Eindverslag
Project .NET interfacing

Student
A. Schlingmann, 99004010
Opleiding
Haagse Hogeschool
Informatica & Informatiekunde
Opleidingsvariant
Ontwikkeling van Software en Technische Infrastructuren (OSTI)
Afstudeerperiode
17 november 2003 t/m 26 maart 2004
Organisatie
Reflecta Automation B.V.
Dorpsweg 26
2811 KH Reeuwijk-Dorp
Tel: 0182-398130
Fax: 0182-398163
Email: info@reflecta.nl
Bedrijfsmentoren
Dhr. T. Dekker
Dhr. G.A. Lenselink
Examinatoren
Dhr. E.M. van Doorn
Dhr. J.L.A. Schramp

Referaat

Schlingmann, A., onderzoek naar en implementatie van interfacing tussen Progress en Microsoft .NET.
Reeuwijk-Dorp, Reflecta Automation B.V., 2004.

Dit document beschrijft en evalueert de werkzaamheden en opgeleverde producten die de auteur heeft uitgevoerd en opgeleverd tijdens een project dat in opdracht van Reflecta Automation is uitgevoerd.

Het project omvat enerzijds het verrichten van een onderzoek naar de mogelijkheden van interfacing tussen enerzijds Progress en .NET en anderzijds Crystal Reports en .NET.
Anderzijds omvat de opdracht het implementeren van een .NET data- en gebruikersinterface. De data-interface bestaat uit een menu-dataobject, dat een dynamisch opgebouwde menustructuur kan inlezen uit een Progress database en deze kan converteren naar een .NET dataobject. De gebruikersinterface wordt opgebouwd op basis van het menu-dataobject en dient een vergelijkbaar uiterlijk te hebben als de Microsoft Windows Explorer.
Daarnaast omvat de opdracht het implementeren van een .NET dataobject dat als datasource kan dienen voor de reporting tool Crystal Reports.

Het project is uitgevoerd in de periode van 17 november 2003 tot en met 26 maart 2004.

Descriptoren

- Crystal Reports;
- IAD;
- Interfacing;
- Microsoft .NET;
- Progress;
- UML.

Voorwoord

Voor u ligt het eindverslag dat ik heb geschreven naar aanleiding van mijn afstudeeropdracht die ik heb uitgevoerd ter afsluiting van de opleiding Informatica & Informatiekunde aan de Haagse Hogeschool. De opdracht is uitgevoerd bij softwarehuis Reflecta Automation B.V. te Reeuwijk-Dorp.

Bij deze wil ik graag mijn bedrijfsmentoren dhr. T. Dekker en dhr. G.A. Lenselink bedanken voor hun begeleiding tijdens het afstuderen. Verder bedank ik ook mijn collega's voor hun waardevolle kennis en prettige samenwerking. Als laatste wil ik mijn examinatoren van de Haagse Hogeschool dhr. E.M. van Doorn en dhr. J.L.A. Schramp bedanken voor het volgen en sturen van het afstuderen.

Reeuwijk-Dorp, 26 maart 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

1. Inleiding

De opleiding Informatica & Informatiekunde aan de Haagse Hogeschool wordt afgesloten door het zelfstandig uitvoeren van een afstudeeropdracht bij een organisatie. Dit verslag beschrijft het verloop van mijn afstudeeropdracht die ik heb uitgevoerd bij softwarehuis Reflecta Automation B.V. te Reeuwijk-Dorp. Hierbij ga ik in op de totstandkoming van de opgeleverde producten en de hierbij gemaakte keuzes. Vervolgens evalueer ik de gekozen aanpak, de uitgevoerde activiteiten en de opgeleverde producten.

Het doel van dit verslag is de examinatoren en de gecommitteerde die het afstuderen beoordelen, inzicht te geven in de omvang, diepgang en het niveau van de opdracht. Daarnaast dienen zij door middel van dit verslag inzicht te krijgen in de kwaliteit van de opgeleverde producten en de kwaliteit van het proces van uitvoering.
Op basis van dit inzicht dienen zij in staat te zijn het afstuderen te beoordelen.

De opbouw van dit verslag is als volgt. Als eerste beschrijf ik in hoofdstuk 2 de organisatie waar de opdracht is uitgevoerd, gevolgd door een omschrijving van de opdracht in hoofdstuk 3. In hoofdstuk 4 wordt het plan van aanpak beschreven. Vervolgens wordt in de hoofdstukken 5 tot en met 8 de uitvoering van het project beschreven, waarbij elke fase van het project een hoofdstuk vormt. Als laatste worden in hoofdstuk 9 de uitgevoerde activiteiten en opgeleverde producten geëvalueerd.

2. Organisatie

2.1 Inleiding

In dit hoofdstuk wordt de organisatie beschreven waar de afstudeeropdracht is uitgevoerd. Als eerste wordt in paragraaf 2.2 de organisatorische inrichting beschreven. Vervolgens worden in paragraaf 2.3 de producten beschreven die de organisatie levert. Als laatste komt in paragraaf 2.4 mijn plaats binnen de organisatie tijdens het afstuderen aan bod.

2.2 Organisatorische inrichting

De afstudeeropdracht is uitgevoerd bij softwarehuis Reflecta te Reeuwijk-Dorp. Dit bedrijf is sinds 1990 werkzaam binnen de automatisering van groothandelaren, importeurs en detailhandelsbedrijven in de sport-, mode-, schoenen- en tweewielerbranche.
Reflecta telt ongeveer 25 medewerkers, verdeeld over drie takken: Reflecta Automation, Retail Systems en Information.
De kernactiviteiten van het bedrijf worden uitgevoerd voor de groothandelaren en importeurs door Reflecta Automation. Automation is de grootste tak binnen Reflecta, die voornamelijk actief is met de handelsapplicatie RA-Trade.
De automatisering van detaillisten wordt verzorgd door Reflecta Retail Systems, dat sinds begin 2000 het pakket Store-IT levert, een pakket voor de modische detailhandel.
Als laatste is Reflecta Information actief in Research & Development (R & D) voor techniek en software. Deze tak is de basis van waaruit nieuwe ontwikkelingen aan de rest van de organisatie en aan klanten ter beschikking worden gesteld.

2.3 Producten

2.3.1 RA-Trade

Reflecta heeft in eigen beheer het softwarepakket RA-Trade ontwikkeld. RA-Trade is een handelsapplicatie waarin specifieke toepassingen zijn opgenomen voor de administratieve organisatie van groothandelsbedrijven en importeurs in de genoemde branches. Hieronder vallen uitgebreide functionaliteiten die nodig zijn voor het registreren van de beweging van goederen, eventueel met meerdere kleuren en maten. In grote lijnen betreft dit relatiebeheer, inkoop, verkoop, voorraadbeheer en de financiële administratie.

RA-Trade bestaat uit een basis die kan worden uitgebreid met 40 aanvullende modules en eventueel met maatwerk. De functionaliteiten die specifiek voor de branches zijn ontwikkeld, omvatten toepassingen voor goederen met meerdere kleuren en maten, seizoenen, assortimenten en prijsbreuken. Naast de mogelijkheid tot het volgen van de goederenbeweging beschikt RA-Trade over een geïntegreerde financiële administratie. De aanvullende modules zijn gesplitst in de hoofdgroepen Relatiebeheer, Inkoop, Verkoop, Voorraadbeheer, Financieel en Algemeen.

Het pakket wordt bediend vanuit een lokale netwerkomgeving of via het Internet. Hierdoor is het mogelijk om bijvoorbeeld vertegenwoordigers, relaties, klanten of thuiswerkers overal ter wereld contact te laten maken met de centrale bedrijfsapplicatie. Per gebruiker worden de presentatielaag en eventuele beperkingen in toegang bepaald. Het pakket is geschreven in de 4GL Progress, gebruikt de open Progress DBMS en is platformonafhankelijk. De gebruikersinterface van het pakket is character based.

2.3.2 Store-IT

Store-IT is een pakket dat specifiek voor de modische detailhandel is ontwikkeld. Het is geschikt voor het beheer van één of meerdere winkels en is een volledig grafisch pakket dat draait onder Microsoft Windows 95/98 en NT. De software bestaat, evenals RA-Trade, uit een basis die kan worden aangevuld met modules als statistieken, klantenbeheer, spaarsystemen, controletellingen, budgetten en prijzenbeheer.

Store-IT is geschreven in Visual Basic en maakt gebruik van Microsoft Access.

2.3.3 Research & Development

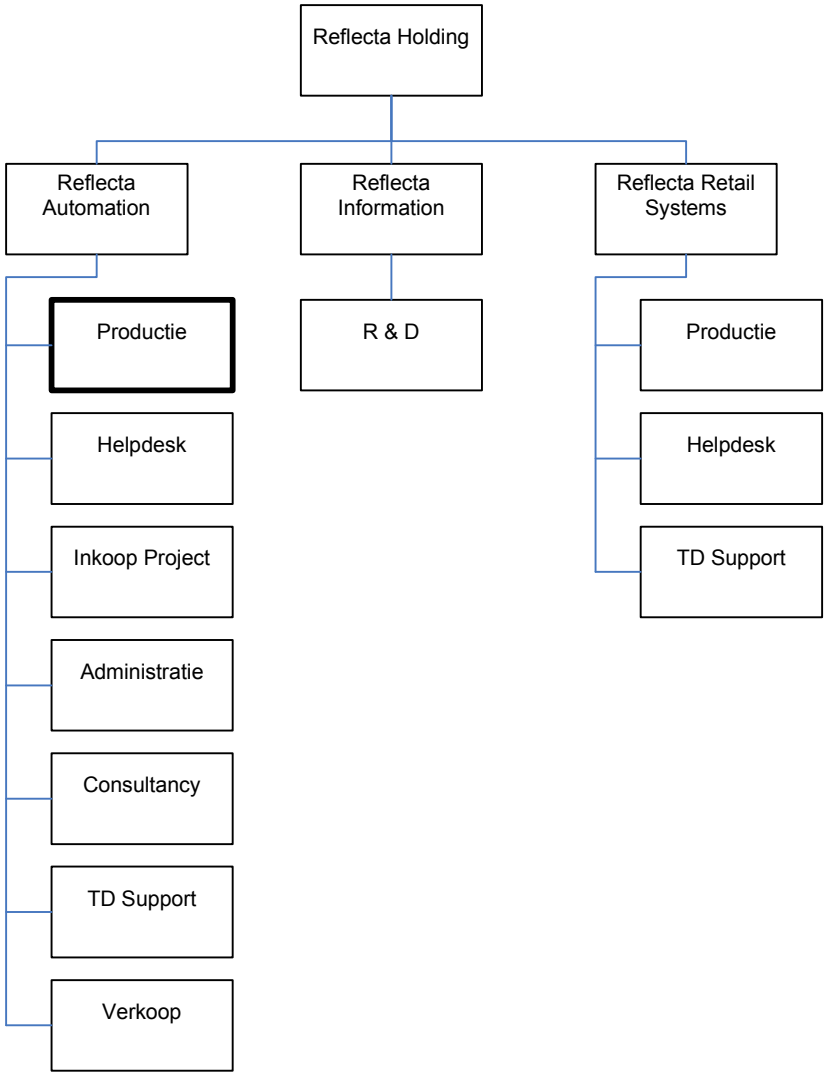
Reflecta Information houdt zich bezig met R & D voor techniek en software. Information is actief met het ontwikkelen van nieuwe grafical user interface (GUI) toepassingen en E-Commerce oplossingen.

Het testen en integreren van nieuwe toepassingen binnen Windows 2000, Office 2000, Linux en Unix in combinatie met de bestaande producten van Reflecta Automation en Reflecta Retail, behoren tevens tot de activiteiten. Daarbij behoren ook het testen van bestaande en nieuwe software op diverse platformen. Information draagt ook zorg voor de technische infrastructuur ten behoeve van remote support bij klanten.

2.4 Plaats binnen de organisatie

De afstudeeropdracht wordt uitgevoerd binnen de tak Automation, die zich bezighoudt met het pakket RA-Trade. Momenteel is een team van drie programmeurs actief met de ontwikkeling van een opvolger van RA-Trade. De opdracht wordt uitgevoerd binnen dit team. Hiervoor is een extra werkplek ingericht.
Het team van programmeurs bestaat uit ervaren Progress ontwikkelaars. De begeleider van het afstudeerproject maakt deel uit van dit team.

Op de volgende pagina is een organigram van de organisatie opgenomen. Hierin is de plaats waar de afstudeeropdracht wordt uitgevoerd, gemarkeerd door een vette rand om de betreffende afdeling.



Figuur 1: *Organigram Reflecta*

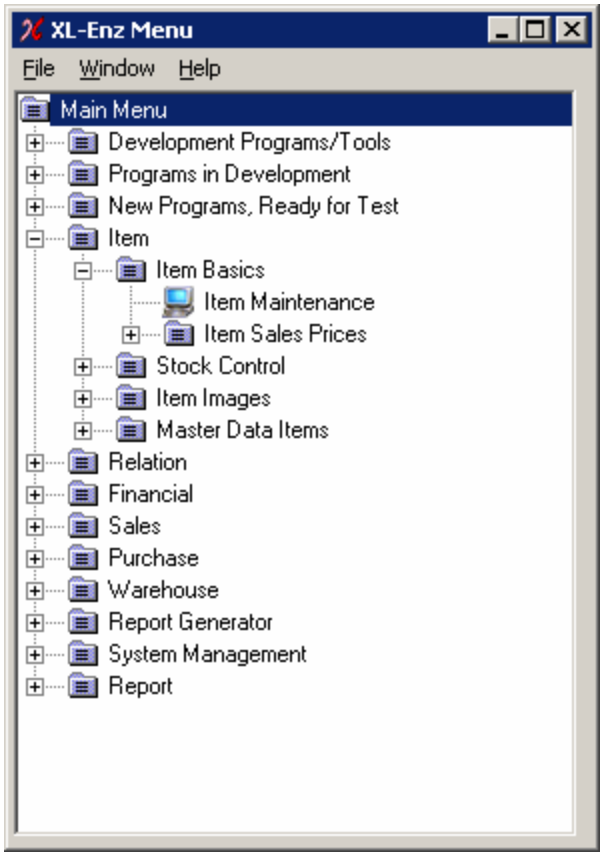
3. Opdracht

3.1 Inleiding

In dit hoofdstuk wordt de afstudeeropdracht beschreven en nader toegelicht. In paragraaf 3.2 wordt de probleemstelling verwoord die ten grondslag ligt aan de opdracht. Vervolgens wordt in paragraaf 3.3 de opdracht omschreven. De doelstellingen van het project worden in paragraaf 3.4 verwoord. Paragraaf 3.5 gaat in op de aanpak van de opdracht. De uit te voeren werkzaamheden en op te leveren producten worden respectievelijk beschreven in de paragrafen 3.6 en 3.7. Als laatste wordt in paragraaf 3.8 de situatie bij aanvang van het project behandeld.

3.2 Probleemstelling

De opvolger van RA-Trade, die de naam XL-Enz heeft gekregen, zal beschikken over een grafische gebruikersinterface.
Omdat XL-Enz volledig in de 4GL Progress wordt geschreven, is de gebruikersinterface afhankelijk van de mogelijkheden die Progress op dit gebied levert.
De gebruikersinterface die reeds is ontwikkeld voor het menu van de applicatie, bestaat uit een menu in de vorm van een boomstructuur. Deze wordt dynamisch opgebouwd, afhankelijk van de beschikbare modules van XL-Enz (zie figuur hieronder).



Figuur 2: *Huidig menu XL-Enz*

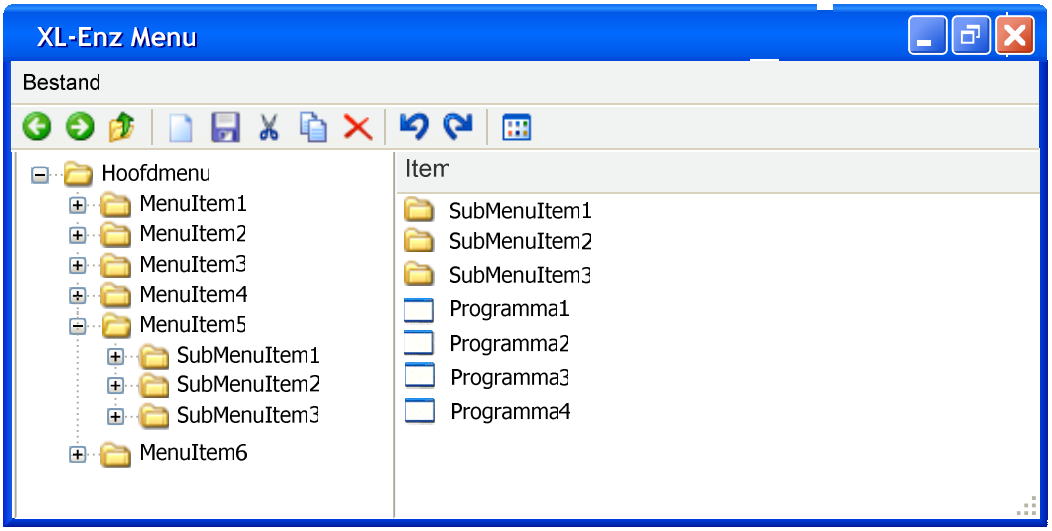
Deze gebruikersinterface is in de ogen van de opdrachtgever niet erg fraai. Het menu is niet aan te passen aan de wensen van de gebruiker, biedt geen extra functionaliteiten en is niet erg aantrekkelijk. De opdrachtgever zou deze gebruikersinterface dan ook anders willen.

De nieuwste versie van Progress (OpenEdge 10) zal volledige ondersteuning bieden met betrekking tot een koppeling met Microsoft .NET technologieën. Deze technologieën bieden mogelijkheden om het menu van XL-Enz te verbeteren.
Reflecta verwacht dat .NET technologieën in de toekomst een steeds belangrijkere rol gaan spelen en wil deze gaan integreren in XL-Enz, om zodoende een voorsprong op de concurrentie te behalen. Op dit moment bezit Reflecta echter geen kennis van de technologieën van .NET.

Reflecta maakt gebruik van de reporting tool Crystal Reports, waarmee gegevens uit databases worden gelezen en vervolgens in een rapport worden gepresenteerd. Momenteel wordt het dataobject dat als datasource dient voor Crystal Reports op een inefficiënte wijze opgebouwd. Dit dataobject bestaat uit een XML bestand, dat veel redundante gegevens bevat. De gegevens in het XML-bestand zijn namelijk opgebouwd uit één tabel, waarvan elk record alle benodigde gegevens bevat. Hierdoor wordt het bestand onnodig groot, wat een nadelige invloed heeft op de performance van het opbouwen van een rapport.
Reflecta wil de opbouw van dit dataobject verbeteren door gebruik te maken van .NET technologieën.

3.3 Omschrijving opdracht

De afstudeeropdracht omvat het verrichten van een onderzoek naar de mogelijkheden van interfacing tussen Progress en .NET om vervolgens in een .NET-omgeving een data- en gebruikersinterface te ontwikkelen voor het menu van XL-Enz.
De data-interface bestaat uit een menu-dataobject, dat een dynamisch opgebouwde menustructuur kan inlezen uit een Progress database.
Het menu wordt in een .NET omgeving opgebouwd op basis van het menu-dataobject en dient een vergelijkbaar uiterlijk te hebben als de Windows Explorer: aan de linkerkant staan de hoofditems van het menu in een boomstructuur en aan de rechterkant staat de inhoud van de hoofditems weergegeven. Wanneer een item uit het menu aangeklikt wordt, dient de achterliggende functionaliteit uitgevoerd te worden. Het menu dient ook functionaliteiten als knippen, slepen en plakken van menu-items te bevatten, waardoor de gebruiker in staat wordt gesteld de opbouw van het menu naar eigen inzicht samen te stellen.
Wijzigingen in de opbouw van het menu dienen te worden opgeslagen in de Progress database.
Hieronder is een prototype opgenomen van het menu zoals dit dient te worden ontwikkeld.



Figuur 3: *Prototype te ontwikkelen menu*

De opdracht omvat ook het onderzoek verrichten naar de mogelijkheden en de implementatie van interfacing tussen Progress, .NET en Crystal Reports.

In Progress OpenEdge 10 is een nieuw dataobject beschikbaar, de ProDataSet. Dit object kan worden gezien als een in-memory database, dat tabellen bevat met hun onderlinge relaties. Dit object kan één op één worden geconverteerd naar een ADO.NET dataobject, dat vervolgens als datasource kan worden gebruikt in Crystal Reports om rapporten op te bouwen, te tonen en te printen.
Doordat deze dataobjecten meerdere tabellen met hun onderlinge relaties kan bevatten, kan hiermee het probleem van het voorkomen van redundante gegevens in het huidige XML-bestand worden opgelost.

3.4 Doelstellingen

Het doel van de opdracht is drieledig:

- Kennis opdoen voor Reflecta over de mogelijkheden van interfacing tussen enerzijds Progress en .NET en anderzijds Crystal Reports en .NET. Deze kennis dient zodanig te worden gedocumenteerd dat deze bruikbaar is voor de medewerkers van Reflecta om .NET te kunnen integreren in XL-Enz;
- Een data- en gebruikersinterface voor het menu van XL-Enz opleveren die aan bovenstaande eisen voldoen;
- Crystal Reports laten werken op basis van een .NET dataobject als datasource.

3.5 Aanpak

Het te verrichten onderzoek naar interfacing tussen Progress en .NET wordt uitgevoerd door het opstellen van een rapport. De volgende onderwerpen worden onderzocht en beschreven:

- Algemene werking van .NET (definitie, doel, (on)mogelijkheden, voorbeelden);
- Mogelijke betekenis van .NET voor Reflecta;
- Werking van interfacing tussen Progress en .NET;
- Werking van interfacing tussen Crystal Reports en .NET.

Het ontwerpen, implementeren en invoeren van het menu van XL-Enz en de datasource van Crystal Reports wordt aangepakt volgens de systeemontwikkelmethode Iterative Application Design (IAD). Hierbij wordt gebruik gemaakt van de technieken van Unified Modeling Language (UML).

3.6 Uit te voeren werkzaamheden

In het kader van de afstudeeropdracht worden de volgende activiteiten verricht:

- Opstellen van een plan van aanpak voor de gehele opdracht;
- Onderzoek verrichten naar interfacing:
  - Kennis vergaren van Progress, Crystal Reports en .NET;
  - Rapport opstellen over interfacing.
- Definitiestudie:
  - Opstellen plan van aanpak;
  - Definiëren ontwikkelscenario;
  - Definiëren van eisen aan het menu van XL-Enz en aan de datasource van Crystal Reports;
  - Bepalen systeemconcept;
  - Beschouwen technische structuur;

  - Beschouwen organisatorische inrichting;
  - Opstellen pilotplan.
- Pilotontwikkeling:
  - Opstellen plan van aanpak;
  - Specificeren globaal-functionele structuur pilot;
  - Specificeren globaal-technische structuur pilot;
  - Specificeren globaal-organisatorische inrichting;
  - Opstellen pilotontwikkelplan;
  - Ontwerpen software-bouweenheden;
  - Bouw software-bouweenheden;
  - Ontwerpen handmatige procedures pilot;
  - Samenstellen handleiding pilot;
  - Opstellen invoeringsprocedure pilot;
  - Integreren bouweenheden;
  - Beoordelen en testen pilot.
- Invoering (per pilot):
  - Opstellen plan van aanpak;
  - Invoeren pilot;
  - Uitvoeren pilotacceptatie;
  - Opstellen van beheerdocumentatie.

3.7 Op te leveren producten

De volgende producten worden opgeleverd:

- Document plan van aanpak voor het gehele project;
- Rapport .NET interfacing;
- Document definitiestudie;
- Document pilotontwikkelrapport (per pilot);
- Menu-dataobject van XL-Enz met bijbehorende gebruikersinterface, dynamisch opgebouwd door middel van .NET-technologieën;
- Datasource van Crystal Reports gekoppeld aan .NET;
- Beheerdocumentatie.

3.8 Uitgangssituatie

Om de opdracht uit te kunnen voeren is een werkplek ingericht bij Reflecta Automation. Deze werkplek bestaat uit de benodigde hardware (PC, mogelijkheid tot printen, Internetverbinding) en software (o.a. ontwikkelomgevingen Progress OpenEdge 10 en Microsoft Visual Studio .NET 2003).
Er wordt binnen Reflecta naar eigen inzicht gebruik gemaakt van systeemontwikkelmethoden. In de praktijk betekent dit dat bij aanvang van een project de specificaties van het op te leveren product worden opgesteld en een planning van de uit te voeren werkzaamheden wordt gemaakt. Tijdens het project wordt deze planning bewaakt en eventueel bijgesteld.

Bij aanvang van het afstuderen bezat Reflecta vrijwel geen kennis over de nieuwe technieken in Progress OpenEdge 10 die benodigd zijn om interfacing naar .NET te realiseren. Er is wel een kleine test uitgevoerd waarbij in een .NET omgeving gegevens uit een Progress database werden gelezen. Deze test is geslaagd.

Tijdens de opleiding Informatica & Informatiekunde heb ik basiskennis opgedaan met het werken in een .NET omgeving. Deze kennis omvatte echter een deel van .NET dat voor deze opdracht weinig relevant leek (namelijk XML-webservices).
Voordat ik aan de opdracht begon had ik geen ervaring met de ontwikkelomgeving Progress.

4. Plan van aanpak

4.1 Inleiding

De eerste activiteit die ik heb uitgevoerd is het opstellen van een plan van aanpak voor het gehele project. Het plan van aanpak wordt gebruikt voor het beheerst uitvoeren van activiteiten voor het bereiken van een eenmalig gesteld doel. In dit document worden afspraken over de opdracht tussen de opdrachtgever en de uitvoerenden gemaakt. Het maken van deze afspraken heeft tot doel om overeenstemming te bereiken met de opdrachtgever over de opdracht en over de uitvoering daarvan.
Door overeenstemming te bereiken wordt de kans verkleind dat de opdrachtgever en de uitvoerenden een verschillend beeld hebben van de uit te voeren opdracht, wat zou kunnen resulteren in een eindproduct dat niet voldoet.

In dit hoofdstuk beschrijf ik de totstandkoming van het plan van aanpak voor dit project. Als eerste beschrijf ik welke methode ik heb gekozen om het document op te stellen. Vervolgens behandel ik de totstandkoming van elk onderdeel van het plan van aanpak. Als laatste ga ik in de paragraaf “Afronding” in op de oplevering en afronding van het document.

4.2 Keuze methode plan van aanpak

Om het plan van aanpak op te stellen, heb ik gebruik gemaakt van de methode die hiervoor wordt aangeleerd op de Haagse Hogeschool. Ik heb voor deze methode gekozen, omdat deze voldoet aan de eisen die worden gesteld aan een plan van aanpak voor een afstudeerproject. Daarnaast wordt binnen Reflecta geen gebruik gemaakt van een methode voor het opstellen van een plan van aanpak, waardoor ik vrij was in het kiezen van een methode.

Als eerste heb ik een hoofdstukindeling gemaakt van het document, waarbij ik de indeling aanhield die de gekozen methode voorschrijft. Vervolgens heb ik per hoofdstuk bekeken of het onderdeel dat in het betreffende hoofdstuk werd behandeld, relevant was voor dit project.
Hieruit bleek dat het hoofdstuk dat de kosten van het project analyseert, niet relevant was. De kosten van het project zijn namelijk wel ingecalculeerd door de opdrachtgever, maar deze werden geacht niet zo hoog te zijn dat het noodzakelijk was hiervoor een budget op te stellen. Er is van uitgegaan dat ik het project grotendeels zelfstandig uit zou voeren en dat de vereiste tijd van begeleiding door de bedrijfsmentor een substantieel deel van zijn tijd zou vergen. Daarnaast was het niet noodzakelijk kosten te maken voor aanschaf van hard- en software, aangezien alle benodigde hard- en software reeds aanwezig was. Als gevolg hiervan heb ik besloten om geen kostenanalyse uit te voeren.

De overige hoofdstukken die de methode voorschrijft, bleken wel relevant. De totstandkoming en inhoud van deze hoofdstukken worden hieronder behandeld, met uitzondering van het hoofdstuk “Opdrachtomschrijving”. De inhoud van dit hoofdstuk is reeds in het hoofdstuk hiervoor beschreven.

Om een zo volledig mogelijk plan van aanpak op te leveren, heb ik vergelijkbare documenten bestudeerd die ik eerder heb opgeleverd. Hieruit heb ik elementen gebruikt en eventueel aangepast aan de situatie van dit project.

4.3 Projectorganisatie

Het hoofdstuk “Projectorganisatie” beschrijft de samenstelling van de projectgroep en de faciliteiten en hulpmiddelen die benodigd zijn om het project uit te kunnen voeren. Daarnaast worden de verwachtingen van de opdrachtgever en de begeleider van het project vastgesteld.
Het doel van dit onderdeel is om verwachtingen op elkaar af te stemmen, waardoor de rollen van de betrokkenen vast komen te liggen. Daarnaast kan door het vaststellen van de benodigde faciliteiten worden bekeken of de faciliteiten reeds beschikbaar zijn of dat er kosten gemaakt moeten worden voor aanschaf van nieuwe faciliteiten. Het beschikbaar hebben van de benodigde faciliteiten voor aanvang van het project bevordert een goede start van het project.

De projectgroep bestond in dit geval uit één projectuitvoerende, de begeleider van het project en de opdrachtgever. De rollen van de projectuitvoerende en de begeleider waren bij aanvang van het project duidelijk. De begeleider stuurt het project door middel van de volgende activiteiten:

- Doornemen en becommentariseren van opgeleverde producten;
- Ondersteuning bieden bij het oplossen van knelpunten;
- Het tijdig melden wanneer in een verkeerde richting wordt gewerkt en sturing in de juiste richting;
- Het juist verwoorden van de functionaliteiten van de te ontwikkelen software.

De rol van de opdrachtgever tijdens het project was echter niet gelijk duidelijk. Directe sturing en begeleiding bleken overbodig, omdat dit reeds werd uitgevoerd door de begeleider. In overleg met de begeleider is besloten dat de opdrachtgever het project slechts volgt en dat er geen directe sturing van hem wordt verwacht. Het volgen van het project bestaat uit rapportage over de voortgang van het project door de uitvoerende of de begeleider en het lezen van relevante opgeleverde documenten, zoals bijvoorbeeld het plan van aanpak.

In het hoofdstuk “Projectorganisatie” heb ik ook de benodigde faciliteiten en hulpmiddelen vastgesteld. Deze heb ik onderverdeeld in hardware, software en literatuur.
De hardware bestaat uit een systeem dat voldoet aan de eisen die worden gesteld door de benodigde software. De benodigde software bestaat uit de ontwikkelomgevingen Progress OpenEdge 10 (beta) en Microsoft Visual Studio .NET 2003 en de reporting tool Crystal Reports. Hieruit volgt dat een besturingssysteem noodzakelijk is waarop deze applicaties kunnen werken. Aangezien Microsoft Windows XP geschikt is voor het gebruik van .NET technologieën, is voor dit besturingssysteem gekozen.
Als laatste is software benodigd om rapporten te kunnen schrijven, diagrammen te ontwerpen en planningen te maken. Hiervoor is Microsoft Office XP, Microsoft Visio 2003 en Microsoft Project gekozen, omdat deze pakketten reeds beschikbaar waren bij Reflecta en ik reeds vertrouwd was met deze producten.
De benodigde literatuur bestaat uit boeken die de gebruikte systeemontwikkelmethode en technieken beschrijven. Daarnaast is documentatie benodigd over de gebruikte

rapportagetechniek om de op te leveren rapporten te laten voldoen aan de eisen van deze techniek. Deze literatuur had ik beschikbaar bij aanvang van het project, omdat ik tijdens mijn opleiding deze literatuur reeds nodig had.
Als laatste is documentatie over Progress, Crystal Reports en .NET benodigd. Deze documentatie kan Reflecta grotendeels ter beschikking stellen. Documentatie over .NET dient echter verkregen te worden door middel van Internet of artikelen uit tijdschriften.

4.4 Projectrisico's

In het hoofdstuk "Projectrisico's" uit het plan van aanpak worden risico's beschreven die zich tijdens het uitvoeren van het project kunnen voordoen. Door projectrisico's te onderkennen, wordt het mogelijk om met negatieve invloeden op het project om te gaan. Voor elk risico wordt een maatregel beschreven die kan worden genomen om kans op het optreden van het risico te verkleinen of de impact van het opgetreden risico te minimaliseren.
De volgende risico's met bijbehorende maatregelen zijn voor dit project gedefinieerd:

- Tijdsvertraging door ziekte of afwezigheid van de uitvoerende van het project.
  Maatregel: Het project inkorten. Deze maatregel is alleen van toepassing bij een afwezigheid gedurende één week of langer. In de planning van het project is een overhead van één week opgenomen.

- Tijdsvertraging door ziekte of afwezigheid van de begeleider van het project.
  Maatregel: Wanneer er niet verder kan worden gewerkt zonder feedback van de afwezige begeleider, wordt er een persoon aangesteld als (tijdelijk) begeleider, die in staat is om de benodigde feedback te geven.

- Het project is te groot in omvang om te kunnen uitvoeren in de beschikbare tijd.
  Maatregel: In overleg met de begeleider en de opdrachtgever de opdracht inkorten.

- Problemen met hardware en/of software.
  Maatregel: Dagelijks back-ups maken van de applicatie en de documentatie.

- De technieken van .NET zijn te complex om binnen de gestelde periode te beheersen.
  Maatregel: Een verkenningstraject naar .NET doorlopen tijdens het onderzoek naar interfacing. Op basis van de bevindingen uit dit onderzoek kan de opdracht eventueel worden aangepast.

4.5 Standaarden en richtlijnen

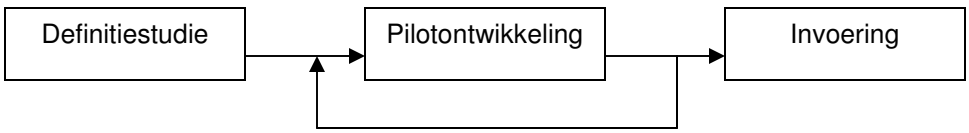
Om de kwaliteit van het verslag over interfacing te waarborgen, heb ik gekozen om gebruik te maken van de rapportagetechnieken die als standaard gelden binnen de Sector Informatica van de Haagse Hogeschool. De keuze voor deze technieken lag voor de hand, aangezien de overige op te leveren rapporten voor dit project ook volgens deze technieken worden opgebouwd. Het zou inconsistent zijn wanneer ik hiervoor een andere techniek had gekozen. Daarnaast heb ik geen ervaring met andere rapportagetechnieken en was het aanleren van nieuwe rapportagetechnieken geen doelstelling van het project. Ten slotte heb ik voor deze techniek gekozen, omdat deze

voldoet aan de eisen die aan rapporten worden gesteld die worden opgeleverd in het kader van een afstudeeropdracht binnen de Haagse Hogeschool.

Om de kwaliteit van de op te leveren software te waarborgen, heb ik gebruik gemaakt van een systeemontwikkelmethode. Door te werken volgens een methode wordt het project gestructureerd en zal het opgeleverde product moeten voldoen aan de gestelde eisen van de ontwikkelmethode. Door het gebruik van een methode wordt het tevens mogelijk een planning van de uit te voeren activiteiten te maken. Hierdoor wordt de kans op overschrijden van de deadline verkleind.

Om het menu-dataobject met de bijbehorende gebruikersinterface en de koppeling van Crystal Reports met .NET te ontwerpen en te bouwen, heb ik gekozen om gebruik te maken van de systeemontwikkelmethode Iterative Application Design (IAD). Tijdens de opleiding Informatica aan de Haagse Hogeschool heb ik echter nauwelijks ervaring met deze methode opgedaan. Alleen in het laatste project ter voorbereiding op het afstuderen heb ik volgens deze methode gewerkt. De overige projecten die ik heb uitgevoerd tijdens mijn opleiding heb ik aangepakt volgens de methode System Development Methodology (SDM).
De reden waarom ik toch voor IAD heb gekozen is vanwege haar iteratieve ontwikkelstrategie. Door een ontwikkeltraject meerdere malen te doorlopen, is het mogelijk om kennis die tijdens het project wordt opgedaan, alsnog te verwerken in het op te leveren product.
Aangezien bij aanvang van het project geen kennis beschikbaar was van de technieken van .NET, leek iteratief ontwikkelen erg waardevol. Het is algemeen bekend dat de technieken van .NET zeer complex zijn. Het was dus waarschijnlijk dat er gedurende de doorlooptijd van het project nog kennis over .NET zou worden opgedaan. IAD maakt het mogelijk om deze kennis alsnog te verwerken in het product, in tegenstelling tot SDM.

IAD deelt het systeemontwikkeltraject op in drie fasen: definitiestudie, pilotontwikkeling en invoering. In de fase definitiestudie worden de eisen aan het systeem vastgesteld. Vervolgens wordt het systeem in de fase pilotontwikkeling opgedeeld in zelfstandig te ontwikkelen pilots. In de fase invoering worden de pilots uiteindelijk ingevoerd in de organisatie.
IAD kent vier iteratiestrategieën om deze fasen te doorlopen. Voor dit project heb ik de iteratiestrategie “incrementeel ontwikkelen” gekozen. Hierbij wordt de fase definitiestudie eenmalig doorlopen. De pilotontwikkeling wordt in iteraties uitgevoerd en de invoering van de pilots vindt in één keer plaats.



Figuur 4: *Iteratiestrategie*

Ik heb voor deze iteratiestrategie gekozen, omdat de eisen aan het op te leveren product reeds voor een groot deel vast stonden bij aanvang van het project. Als gevolg hiervan kon worden volstaan met het eenmalig doorlopen van de definitiestudie.
De keuze om de fase pilotontwikkeling iteratief te doorlopen, komt voort uit het feit dat alle betrokkenen bij het project onbekend waren met de technieken van .NET. Door de pilots iteratief te ontwikkelen kon zodoende later opgedane kennis alsnog worden verwerkt in de pilots.

De invoering van de pilots kon eenmalig plaatsvinden, omdat er geen behoefte was om delen van het systeem zo snel mogelijk operatief te maken.

Binnen de systeemontwikkelmethode wordt gebruik gemaakt van technieken om het systeem te ontwerpen. De technieken die ik heb gekozen zijn die van de Unified Modeling Language (UML). De reden waarom ik voor deze technieken heb gekozen, komt voort uit het feit dat deze gericht zijn op toepassing in een objectgeoriënteerde omgeving (zoals bijvoorbeeld .NET).
Tijdens de opleiding Informatica is het gebruik van deze technieken wel aangeleerd, maar slechts nauwelijks toepast tijdens projecten. De technieken die vrijwel altijd werden gebruikt, waren die van Yourdon. Ik heb echter niet voor deze technieken gekozen. In het project ter voorbereiding op het afstuderen heb ik deze technieken namelijk gebruikt in een .NET omgeving. Dit bleek geen succes te zijn, omdat Yourdon niet is gericht op objectgeoriënteerde omgevingen. Achteraf werd mij dan ook aangeraden om in een dergelijke situatie UML te gebruiken.

Tijdens het opstellen van het plan van aanpak heb ik onderzocht welke diagrammen van UML relevant zouden kunnen zijn voor dit project. De onderstaande diagrammen leken bruikbaar:

- Use-case-diagram: om de functionaliteiten van het systeem te kunnen modelleren;
- Klassediagram: om de elementen waaruit het systeem bestaat te kunnen modelleren;
- Sequencediagram: om de interactie tussen objecten te kunnen modelleren;
- Toestandsdiagram: om het gedrag van objecten te kunnen modelleren;
- Componentdiagram: om de afhankelijkheden tussen componenttypen te kunnen modelleren.

De onderstaande diagrammen leken niet relevant in het kader van dit project:

- Objectdiagram: dit diagram is vergelijkbaar met een klassediagram (toont een instantie van het klassediagram) en leek geen toegevoegde waarde te bieden;
- Collaboratiediagram: dit diagram is vergelijkbaar met een sequencediagram;
- Activiteitsdiagram: dit diagram leek geen toegevoegde waarde te bieden na het modelleren van het sequence- en toestandsdiagram;
- Deploymentdiagram: dit diagram is alleen toepasbaar bij de bouw van gedistribueerde systemen. Gezien de opdracht leek dit diagram niet relevant.

In overleg met de begeleider heb ik besloten om de software die in .NET wordt ontwikkeld, te schrijven in de programmeertaal C#, ondanks dat ik geen ervaring met deze taal had. De keuze voor deze taal komt echter voort uit het feit dat C# specifiek voor .NET is ontwikkeld, wat in de praktijk voordelen op zou kunnen leveren. Andere talen hebben namelijk uitbreidingen of aanpassingen nodig om goed te kunnen werken in een .NET omgeving.
Het ontbreken van ervaring met C# leek geen bezwaar, omdat ik wel ervaring had met de talen C en Java. De syntax van C# is op deze talen gebaseerd, waardoor het aanleren van C# waarschijnlijk geen groot obstakel zou vormen.

Als laatste heb ik in het hoofdstuk “Standaarden en richtlijnen” de standaard voor de lay-out van de op te leveren rapporten vastgesteld. Om aan de eisen van de Haagse Hogeschool met betrekking tot rapportage te kunnen voldoen, heb ik gekozen om

gebruik te maken van de rapportagetechnieken die als standaard worden gebruikt binnen de Sector Informatica van de Haagse Hogeschool.
Om deze eisen SMART (Specifiek, Meetbaar, Acceptabel, Realistisch en Tijdgebonden) te maken, heb ik in het plan van aanpak een bijlage opgenomen, waarin deze eisen in een checklist zijn opgenomen. Deze checklist is bruikbaar om te controleren of rapporten aan de eisen van de techniek voldoen. Daarnaast heb ik het lettertype vastgesteld van de verschillende elementen uit de rapporten.

4.6 Planning

Om het project beheersbaar te maken en om een schatting te kunnen maken van de benodigde tijd, heb ik een planning opgesteld. Aangezien ik gebruik maakte van een systeemontwikkelmethode, kon ik de planning baseren op de activiteiten die deze methode voorschrijft.
Om de planning te kunnen maken, heb ik als eerste een schatting van de benodigde tijd per fase van het project gemaakt. Deze schatting heb ik gedeeltelijk gebaseerd op eerdere ervaringen die ik heb opgedaan met het werken volgens een systeemontwikkelmethode. Daarnaast heb ik de activiteiten bestudeerd die IAD voorschrijft, om vervolgens een schatting te maken van de duur van deze activiteiten. Hier is de onderstaande schatting uit voortgekomen:

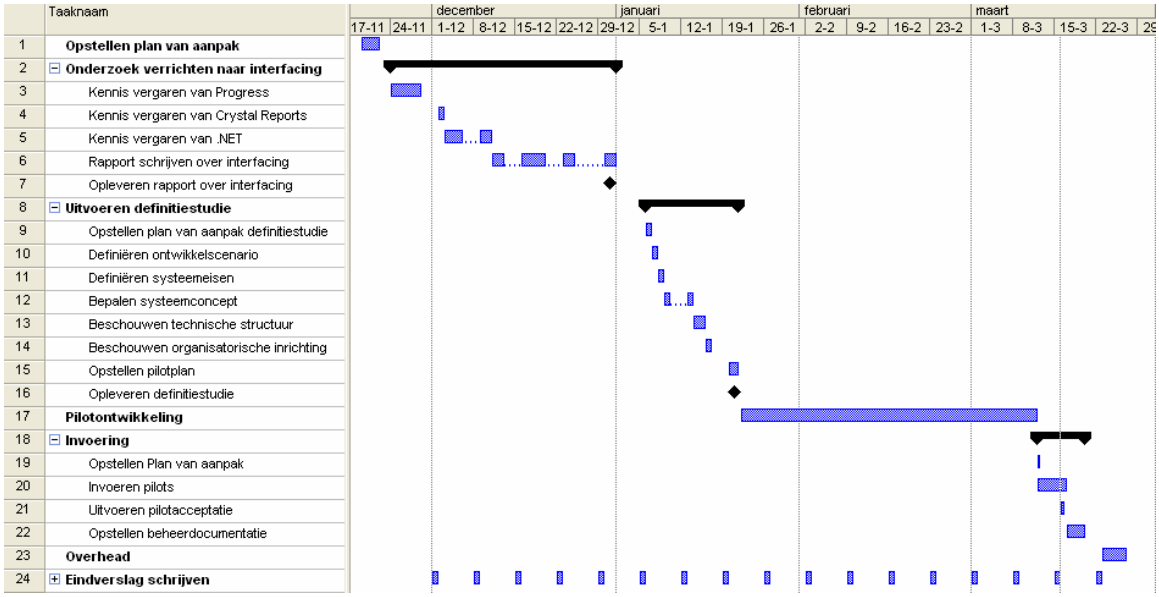
Fase	**% van totale project**
Plan van aanpak	5 %
Verslag interfacing	25 %
Definitiestudie	10 %
Pilotontwikkeling	35 %
Invoering	5 %
Totaal:	**80 %**

Figuur 5: *Schatting van de verdeling van de fasen*

Naast deze fasen heb ik tijd ingepland voor uitloop van het project (5 %) en voor activiteiten die worden uitgevoerd in het kader van het afstuderen (15 %).

Op basis van deze globale schatting heb ik een gedetailleerde planning gemaakt in dagen. Hierbij heb ik gebruik gemaakt van een planningsalgoritme dat mij is aangeleerd op de Haagse Hogeschool. Dit algoritme gaat uit van het minimale, het maximale en het gemiddelde aantal dagen dat geschat wordt benodigd te hebben om een activiteit uit te voeren. Door middel van de formule “gepland aantal dagen = (minimaal + maximaal + 2 * gemiddeld) / 4” wordt vervolgens het aantal dagen berekend dat gepland wordt voor deze activiteit.

Nadat ik het benodigde aantal dagen per fase had gepland, heb ik een GANNT diagram opgesteld, waarin ik de uit te voeren activiteiten verdeeld heb over het gepland aantal dagen. Deze planning is op de volgende pagina opgenomen.



Figuur 6: *Planning bij aanvang project*

Omdat de fase pilotontwikkeling tijdens de fase definitiestudie wordt verdeeld in meerdere pilots en iteratief zou worden doorlopen, was het niet mogelijk om bij aanvang van het project een gedetailleerde planning voor deze fase te maken. De verdeling van de te ontwikkelen pilots was namelijk nog niet bekend op dat tijdstip. Een gedetailleerde planning voor de fase pilotontwikkeling is gemaakt tijdens de activiteit “opstellen pilotplan”, die tijdens de fase definitiestudie is uitgevoerd.
Indien nodig wordt deze planning na afloop van elke fase bijgesteld aan nieuwe ontwikkelingen en inzichten. De aanpassingen in de planning die tijdens dit project noodzakelijk waren, worden in dit document beschreven in de paragrafen “Afronding”. Deze paragrafen zijn opgenomen bij elk hoofdstuk dat een fase uit het project behandelt.

4.7 Testaanpak en kwaliteitscontrole

Om te garanderen dat de opgeleverde software voldoet aan de eisen die opgesteld zijn tijdens het uitvoeren van de definitiestudie, wordt deze getest. In het plan van aanpak heb ik als teststrategie “black box testing” gekozen. Deze keuze komt voort uit het feit dat ik het meest bekend ben met deze teststrategie. De geïmplementeerde pilots worden getest op een correcte werking, op het bevatten van de juiste functionaliteiten en op performance.

Daarnaast heb ik een acceptatietest gepland, waarin de opdrachtgever en/of begeleider het systeem test aan de hand van de opdrachtomschrijving. Deze test is noodzakelijk om een oordeel te kunnen verkrijgen over de opgeleverde producten.

De paragraaf “Kwaliteitscontrole” uit het plan van aanpak had tot doel om de wijze van controle op kwaliteit van de opgeleverde producten vast te leggen. Hierbij heb ik onderscheid gemaakt in kwaliteitscontrole van de rapportopmaak en van de op te leveren software.
De controle van de rapportopmaak vindt plaats door gebruik te maken van de checklist die als bijlage bij het plan van aanpak is opgenomen.

De kwaliteitscontrole van de opgeleverde software vindt plaats door gebruik te maken van een systeemontwikkelmethode en enkele technieken om het systeem te ontwerpen. Daarnaast heb ik de kwaliteit van de lay-out van de source code gewaarborgd door deze op te maken volgens aangeleerde standaarden van de Haagse Hogeschool. Als laatste heb ik vermeld dat kwaliteitscontrole van de software ook plaats vindt door middel van het uitvoeren van de eerder beschreven teststrategie.

4.8 Afronding

Om een voorlopige versie van het plan van aanpak op te kunnen leveren, heb ik drie dagen benodigd gehad, zoals gepland. Na het bespreken van deze versie met de begeleider heb ik de opdracht specifieker omschreven op basis van de gegeven feedback.
De begeleider gaf te kennen dat hij het planningsalgoritme zelf niet op de toegepaste wijze zou gebruiken, omdat ik het maximaal aantal benodigde dagen hoger heb gepland dan dat er maximaal beschikbaar is. Uit eerdere ervaringen heeft hij geleerd dat het werkelijk aantal benodigde dagen vaak dichterbij het maximaal geplande aantal dagen ligt, dan bij het vooraf geplande aantal dagen. Dat betekent dat de kans groot is dat het project uitloopt op de planning. Uitloop was voor dit project echter niet mogelijk, omdat de doorlooptijd niet aanpasbaar was.
Om dit probleem op te lossen zou de begeleider ervoor kiezen om het maximaal aantal benodigde dagen vast te stellen op het aantal beschikbare dagen.
Ik heb ervoor gekozen om de planning niet hierop aan te passen, omdat ik reeds rekening had gehouden met uitloop van werkzaamheden. Hierdoor was wel enige speling in de planning mogelijk.

Na de aanpassingen op basis van de feedback van de begeleider heb ik een definitieve versie van het plan van aanpak opgeleverd. Deze heb ik overhandigd aan de begeleider en de opdrachtgever. Deze zijn vervolgens akkoord gegaan met het plan van aanpak. De laatste versie van het plan van aanpak is als externe bijlage opgenomen bij dit verslag.

5. Onderzoek naar interfacing

5.1 Inleiding

Nadat ik de voorlopige versie van het plan van aanpak had opgeleverd, ben ik begonnen aan de fase “Onderzoek verrichten naar interfacing”. Deze fase had tot doel om kennis op te doen voor Reflecta over interfacing tussen Progress en .NET. Deze kennis diende zodanig te worden gedocumenteerd dat deze bruikbaar is voor de medewerkers van Reflecta om .NET te kunnen integreren in XL-Enz. Het resultaat van deze fase was dan ook een verslag over interfacing tussen Progress en .NET.

Dit hoofdstuk beschrijft de werkzaamheden die ik heb uitgevoerd tijdens deze fase. Als eerste ga ik in op hoe ik kennis heb vergaard van Progress en .NET. Deze kennis was benodigd om het rapport over interfacing te kunnen schrijven. Vervolgens beschrijf ik de totstandkoming van het rapport. De beschrijving van de oplevering van het rapport vormt de afsluiting van dit hoofdstuk.

5.2 Kennis vergaren van Progress

Om het rapport over interfacing te kunnen schrijven, was het noodzakelijk om kennis te vergaren van Progress. Met name kennis van de werking van de ProDataSet was essentieel, aangezien de interfacing naar .NET op basis van deze techniek verloopt.
Ik had nog niet eerder met Progress gewerkt, waardoor het noodzakelijk was om eerst basiskennis van de taal op te doen, voordat ik de werking van de ProDataSet kon onderzoeken.

5.2.1 De ontwikkelomgeving Progress

Omdat ik onbekend was met Progress, heb ik mij eerst op deze omgeving georiënteerd. Mijn begeleider heeft mij de mogelijkheden van Progress duidelijk gemaakt en ik heb op Internet gezocht naar beschrijvingen van deze omgeving.
Progress bleek een Relational Database Management System (RDBMS) en een 4GL te zijn, waarmee applicaties kunnen worden ontwikkeld. Deze applicaties kunnen zowel een character based als grafische gebruikersinterface hebben.
Door middel van de Progress DataServer Architecture kan naast de Progress RDBMS een verscheidenheid aan andere RDBMS-en worden gebruikt.
De Progress Application Development Environment biedt een aantal middelen om in een grafische omgeving applicaties te ontwikkelen. De Progress 4GL is een procedurele taal die is geoptimaliseerd voor “online transaction processing”. Progress is vooral geschikt voor bedrijfskritische applicaties, waarbij gegarandeerde performance en gegevensintegriteit vereist is.
Progress maakt geen gebruik van standaard SQL om te communiceren met het RDBMS, maar heeft hiervoor een eigen vergelijkbare taal.
Progress applicaties kunnen op verschillende besturingssystemen opereren, waaronder Windows, UNIX, OS/2 en DOS.

De applicatie XL-Enz wordt gebouwd voor een gedistribueerde omgeving. Het DBMS, de applicatielogica en de gebruiker zijn logisch (en eventueel fysiek) gescheiden. Het DBMS bevindt zich op een database server, de applicatielogica bevindt zich op de Progress Application Server (AppServer) en de gebruiker (client) communiceert met deze server. De client communiceert in dit geval dus nooit rechtstreeks met het DBMS.

Nadat ik bekend was met de eigenschappen van Progress, was de volgende stap om kennis op te doen van de Progress 4GL.

5.2.2 Progress TempTables

Mijn begeleider had bij aanvang van het afstuderen een stappenplan opgesteld, dat ik in overleg met hem heb gevolgd om de benodigde kennis van de Progress 4GL op te doen. Ik ben begonnen met het maken van een klein klant-order-artikel systeem op basis van zogenaamde TempTables. In Progress is een TempTable een databasetabel in het geheugen, die alle eigenschappen kan bezitten van een fysieke databasetabel. De begeleider heeft voor deze introductie gekozen, omdat het ProDataSet-object ook gebruik maakt van TempTables. De kennis die ik tijdens de introductie opdeed, zou dus later bruikbaar zijn, omdat het ProDataSet-object ten grondslag ligt aan interfacing tussen Progress en .NET.

Met behulp van de Language Reference Help van Progress heb ik als eerste een statische versie van het systeem gemaakt. Hierbij worden alle tabellen compiletime gegenereerd, wat relatief eenvoudig te programmeren is en daardoor goed als een introductie kon dienen.
In XL-Enz worden veel schermen met de daarbij behorende data dynamisch (runtime) opgebouwd. In de afstudeeropdracht komt het dynamische aspect ook terug in de vorm van het dynamisch opgebouwde menu-dataobject en het dynamisch opgebouwde dataobject voor Crystal Reports. De begeleider stelde daarom voor om van de statische versie van het klant-order-artikel systeem een dynamische versie te maken.
Dit bleek moeilijker dan het maken van de statische versie, omdat het abstracter was. Wanneer ik niet verder kwam, gaf de begeleider hints waardoor ik in staat was het systeem stap voor stap geheel dynamisch te maken.
Het resultaat was een scherm waarin drie ”browsers” stonden (klant, order, artikel). Wanneer een klantnaam werd aangeklikt, werden de orders van die klant getoond, met de daarbij behorende artikelen per order.

5.2.3 Progress ProDataSet

Na het maken van de dynamische versie van het klant-order-artikel systeem, was de volgende stap uit het stappenplan om dit systeem te laten werken op basis van een ProDataSet-object. Hiermee kon de werking van deze techniek worden onderzocht en in praktijk worden gebracht, waardoor de (on)mogelijkheden van de techniek duidelijk werden. Dit zou kennis opleveren dat in het verslag over interfacing kon worden verwerkt.

Om de werking van de ProDataSet te onderzoeken, had ik de beschikking over bèta documentatie van Progress, waarin de functionele specificatie van de ProDataSet werd beschreven. Dit bleek de enige beschikbare documentatie te zijn op dat moment.

Na bestudering van dit document werd duidelijk dat een ProDataSet een object is dat TempTables bevat. Hierin zijn de onderlinge relaties tussen de tabellen bekend. Na de definitie van de TempTables die de ProDataSet dient te bevatten, worden deze gevuld met behulp van een query op de database. Vervolgens kan de data die de ProDataSet bevat, worden gelezen en gewijzigd.
Het idee achter de ProDataSet is dat deze kan worden gevuld door middel van slechts één query op de “parent-tabel” (bijvoorbeeld de klantentabel). De bijbehorende “childs” (bijvoorbeeld orders van een klant, en daarvan weer de artikelen) worden vervolgens automatisch herkend door middel van de gedefinieerde onderlinge relaties en in de ProDataSet geladen.

Met behulp van de opgedane kennis heb ik vervolgens een implementatie van het ProDataSet-object gemaakt. Daarbij heb ik het eerder gemaakte klant-order-artikel systeem als uitgangspunt gebruikt. Tijdens de implementatie bleek de beschikbare informatie erg beperkt te zijn, aangezien deze alleen een formele beschrijving van de syntax van de onderdelen van de ProDataSet bevatte, zonder veel verdere uitleg of voorbeelden. De enige optie was programmeren volgens “trial and error”. Na een dag resulteerde dit in een werkende ProDataSet.
Deze ProDataSet bevatte twee TempTables, “Customer” en “Order”. Door middel van een query op de databasetabel “Customer” werden deze tabellen gevuld. Bij elke klant werden automatisch de bijbehorende orders in de TempTable “Order” gezet.

Toen ik echter de query probeerde uit te breiden, liep ik tegen een belangrijke beperking bij het vullen van de ProDataSet aan. Wanneer in de query een “where”-constructie op de childtabel (“Order”) werd opgenomen, bleek dit niet het gewenste resultaat te hebben.
De query bleek alleen toegepast te worden op de parenttabel (“Customer”), waarna automatisch álle bijbehorende childs werden geladen in de childtabel, ongeacht de query die op de childtabel was gezet.
Navraag op een forum van de website van Progress resulteerde in reacties van gebruikers die tegen hetzelfde probleem waren gestuit. Een geboden oplossing van een gebruiker bleek na overleg niet afdoende, aangezien de functionaliteiten van de ProDataSet hierbij vrijwel niet werden gebruikt. Ook een expert van Progress kon hiervoor geen oplossing bieden en beaamde het probleem.

Deze beperking had tot resultaat dat Reflecta besloot voorlopig af te zien van een brede implementatie van ProDataSets in XL-Enz. Voor de afstudeeropdracht kon de techniek echter wel voldoen.

Na deze implementatie van een ProDataSet leek de opgedane kennis voldoende om de medewerkers van Reflecta de werking van de ProDataSet uit te kunnen leggen in het rapport over interfacing.

5.3 Kennis vergaren van .NET

Naast kennis van Progress was het noodzakelijk om kennis van .NET te vergaren, voordat ik het rapport over interfacing kon schrijven. Tijdens mijn opleiding heb ik gedurende de module voorbereidend op het afstuderen reeds kennis opgedaan van .NET. Deze kennis was echter zeer globaal en omvatte een slechts een deel van .NET, namelijk het deel over XML-webservices. Om de medewerkers van Reflecta een

compleet overzicht van .NET te kunnen geven, was het noodzakelijk om meer kennis van .NET te vergaren.
Gezien het enorme aanbod aan informatie en de complexiteit van .NET, was het noodzakelijk structuur en prioriteiten aan te brengen in de delen van .NET die ik zou onderzoeken en beschrijven in het rapport over interfacing. Ten tijde dat ik zelf tijdens mijn opleiding met .NET te maken kreeg, had ik vooral de volgende vragen met betrekking tot .NET:

- Wat is .NET?
- Hoe werkt het?
- Waar kan het zinvol worden toegepast?
- Welke plaats gaat .NET in de toekomst innemen?

Tot het moment dat ik aan deze opdracht begon, zijn bovenstaande vragen nooit duidelijk beantwoord. Om de betekenis van .NET voor mijzelf én voor de medewerkers van Reflecta duidelijk te krijgen, leek het mij noodzakelijk om in het te verrichten onderzoek in ieder geval bovenstaande vragen beantwoord te krijgen.
In overleg met de begeleider heb ik besloten om tijdens het onderzoek naar .NET de volgende onderwerpen te onderzoeken en te beschrijven:

- Definitie van .NET;
- Technieken van .NET;
- Beveiliging binnen .NET;
- Toekomst van .NET.

Per onderwerp heb ik een inventarisatie van informatiebronnen gemaakt. Vooral het Internet bleek hierbij een grote bron van informatie. Microsoft besteedt een groot deel van haar website aan een beschrijving van .NET. Daarnaast leverde een zoekopdracht naar .NET door Google enkele tienduizenden internetpagina's op waar uiteenlopende informatie over .NET te vinden was. Ook heb ik navraag gedaan of er binnen Reflecta documentatie over .NET beschikbaar was. De tijdschriften Software Release Magazine (een vakblad voor software ontwikkeling) en het Windows & .NET Magazine bleken artikelen over (delen van) .NET te bevatten.
Per onderwerp heb ik tien tot vijftien bronnen opgeslagen, die voor mij waardevol leken in het kader van dit project. Elke categorie bevatte bronnen die afkomstig waren van Microsoft. Vooral het Microsoft Developers Network (MSDN) bevatte een grote hoeveelheid informatie over .NET.
Daarnaast bevatten de bronnen publicaties van instanties die zich bezighouden met ontwikkelingen in de ICT sector, zoals ZDNet, W3C, USAToday.com, VeriSign, ComputerWeekly.com en WebWereld.

Na twee dagen informatiebronnen over .NET zoeken, doornemen en categoriseren, had ik een beeld van de wijze waarop ik deze informatie zou kunnen verwerken in het rapport over interfacing. Het leek mij dan ook verstandig om dit beeld te concretiseren door een hoofdstukindeling van het rapport te maken en deze te bespreken met de begeleider.
In de planning had ik vijf dagen opgenomen om kennis te vergaren van .NET. De twee dagen die ik had besteed om informatiebronnen te inventariseren boden mij echter genoeg mogelijkheden om reeds te beginnen met het schrijven van het rapport over interfacing. Hiermee ben ik dan ook begonnen, met de gedachte dat het waarschijnlijk noodzakelijk zou zijn de activiteit "Kennis vergaren van .NET" tijdens het schrijven van het rapport alsnog uit voeren, wanneer kennis over een specifiek onderdeel van .NET vereist was.

5.4 Rapport schrijven over interfacing

5.4.1 Doelgroep en opbouw rapport

Als eerste heb ik een opbouw van de inhoud van het rapport over interfacing gemaakt. Het rapport was bedoeld als een interne publicatie over .NET. De doelgroep bestond uit alle medewerkers van Reflecta. Het doel van het rapport was drieledig:

- Alle medewerkers kennis op laten doen van .NET in het algemeen;
- Het management van Reflecta door middel van het rapport kennis op laten doen over de mogelijkheden van .NET voor Reflecta;
- De software ontwikkelaars van Reflecta kennis op laten doen over de implementatie van interfacing tussen Progress en .NET.

Hierom heb ik het rapport zo opgebouwd dat er een duidelijke scheiding is tussen de verschillende onderwerpen.
De opbouw van het rapport is als volgt. Als eerste heb ik beschreven wat .NET is. Vervolgens ben ik ingegaan op de standaarden en technieken die .NET gebruikt. Daarna heb ik de mogelijkheden van .NET voor Reflecta geanalyseerd. Als laatste heb ik de technische implementatie van interfacing tussen Progress en .NET beschreven.

De opbouw van het rapport heb ik besproken met mijn begeleider. Deze ging hiermee akkoord.
Vervolgens ben ik begonnen met het schrijven van de hoofdstukken. De onderstaande paragrafen beschrijven de totstandkoming van elk hoofdstuk uit het rapport. Citaten uit het rapport zijn te herkennen doordat deze in een grijze rechthoek zijn geplaatst.

5.4.2 Definitie Microsoft .NET

In dit hoofdstuk heb ik beschreven wat .NET is. Dit hoofdstuk was bedoeld voor alle medewerkers van Reflecta. Men wist ongeveer wat .NET is, maar kon het moeilijk onder woorden brengen en er iets concreets bij voorstellen. Tijdens het verzamelen van informatiebronnen over .NET was mij reeds opgevallen dat een groot deel van het bedrijfsleven dezelfde moeite had met .NET als de medewerkers van Reflecta. Ook ikzelf had nog geen duidelijk beeld van de mogelijkheden en reikwijdte van .NET. Om deze redenen heb ik veel aandacht besteed aan een heldere en complete definitie van .NET.
Als eerste heb ik naar definities gezocht die Microsoft zelf geeft van .NET. Deze definities varieerden sterk en gaven geen heldere beschrijving van .NET. Eén van de vier definities die ik heb opgenomen in het rapport is hieronder opgenomen.

> “Microsoft .NET is een reeks softwaretechnologieën van Microsoft om verbinding te maken in uw wereld van informatie, mensen, systemen en apparaten. Met .NET kunt u software op een ongekende manier integreren via het gebruik van XML-webservices: kleine, onopvallende toepassingen die als bouwstenen op elkaar - en op andere, grotere toepassingen - aansluiten via Internet.” (Bron: www.microsoft.com)

De essentie van .NET bleek moeilijk te beschrijven in enkel natuurlijke taal. In vrijwel elke definitie kwamen de termen XML en XML-webservices terug. Omdat dit twee elementaire onderdelen van .NET bleken te zijn, heb ik deze onderdelen toegelicht. Hierbij heb ik echter gekozen om deze onderdelen zonder technische details te beschrijven. Hierdoor zou het hoofdstuk namelijk minder goed leesbaar worden voor de

doelgroep die een minder technische achtergrond heeft. Voor technische details over de onderdelen heb ik echter wel verwezen naar het hoofdstuk dat hierop ingaat. Vanwege mijn onbekendheid met XML heb ik onderzoek hiernaar moeten verrichten om hierover te kunnen schrijven. Het Internet bleek hierbij wederom een waardevolle informatiebron. Na onderzoek heb ik XML bijvoorbeeld als volgt beschreven.

> Om een standaard te kunnen bieden voor data-uitwisseling maakt .NET gebruik van eXtensible Markup Language (XML).
> XML wordt op dit moment al veelvuldig gebruikt bij data-uitwisseling op Internet. De informatie die een WAP-telefoon bijvoorbeeld inleest, is opgesteld in XML. En de kans is groot dat het nieuws van een online krant is samengesteld in XML en naar HTML is geconverteerd.
> In een XML-document zijn alleen de gegevens zelf en de structuur van de gegevens opgeslagen. Het document hoeft slechts één keer aangemaakt te worden en de gegevens kunnen vervolgens via allerlei media getoond worden: op een computermonitor, een mobiele telefoon, een televisie, enzovoort. XML is overal te gebruiken waar informatie opgeslagen en gepresenteerd wordt. Het kan dus niet alleen gebruikt worden om informatie te presenteren, maar ook voor de uitwisseling van informatie.

Ondanks het onderzoeken en beschrijven van meerdere definities van .NET, bleef het idee achter .NET vaag. Nader onderzoek op de website van Microsoft leverde een overzicht van de reikwijdte van .NET op, waarbij .NET werd verdeeld in vijf delen. Dit overzicht maakte mij de reikwijdte van .NET aanzienlijk duidelijker, waardoor ik heb besloten deze verdeling op te nemen in het rapport. Het overzicht ziet er als volgt uit.

> Om zicht te krijgen op de reikwijdte van .NET heeft Microsoft het .NET platform onderverdeeld in vijf delen:
> 1. Hulpmiddelen voor de ontwikkelaar;
> 2. Servers;
> 3. Basisset XML-webservices;
> 4. Clients;
> 5. Gebruikerservaringen.

Elk deel heb ik nader onderzocht en toegelicht. Zo bleek bij nader onderzoek de ontwikkeling van het derde onderdeel, “Basisset XML-webservices”, vrijwel te zijn stopgezet, wegens een tegenvallende ontvangst ervan door het bedrijfsleven. Microsoft zelf prijst het product op haar website echter nog steeds aan, ondanks dat het sinds twee jaar vrijwel niet verder is ontwikkeld en slecht ontvangen is.

Na de definitie en reikwijdte van .NET beschreven te hebben zoals Microsoft die ziet, heb ik onderzocht hoe Microsoft de toekomst van .NET ziet. De website van Microsoft bleek ook hier een goede bron van informatie. De beschrijving van dit onderdeel heb ik dan ook grotendeels hierop gebaseerd. Microsoft bleek grote toekomstverwachtingen van .NET te hebben:

> Volgens Microsoft zal .NET de manier waarop gedacht wordt over en gebruik gemaakt wordt van computers drastisch veranderen. Momenteel wordt de computerwereld overheerst door twee concepten: de server en de client. .NET breidt dit model uit tot een gedistribueerd computermodel van flexibel gekoppelde services.
> In plaats van het gebruikelijke onderscheid tussen pc en server, vindt de verwerking plaats waar dit het beste uitkomt, ongeacht of het hierbij gaat om een server, pc, handheld apparaat of ander apparaat. “Dit is intelligent computergebruik voor een nieuwe generatie intelligente apparaten”, aldus Microsoft.

Omdat de beschrijving van een product door de ontwikkelaar van het product meestal niet objectief is, heb ik vervolgens onderzocht hoe het bedrijfsleven .NET definieert en ervaart. Hierbij heb ik het Internet wederom als belangrijkste informatiebron gebruikt. Onderzoek naar meningen over .NET leverde een overvloed aan artikelen op, variërend van de resultaten van een onafhankelijk onderzoek naar .NET door Forrester Research tot meningen van individuele programmeurs.
Het bleek moeilijk om deze verschillende meningen te verwoorden tot een compact verhaal dat de hoofdgedachte uit deze meningen weergeeft. Na het doornemen van ongeveer tien tot vijftien artikelen, bleek echter dat een afwachtende houding ten opzichte van .NET overheerst. Samengevat bleken de belangrijkste redenen hiervoor:

- Men weet na drie jaar nog steeds niet wat .NET precies is. Het gebruik van ".NET" voor veel verschillende producten en diensten werkt verwarrend;
- Men ziet geen zinvolle toepassing van .NET in de praktijk;
- Men vraagt zich af of het doel van .NET om vergaande integratie van systemen te realiseren, wel gewenst is;
- .NET is in complexe situaties nog niet stabiel genoeg om erop te kunnen vertrouwen.

Onderzoek naar de mening van het bedrijfsleven over de toekomst van .NET leverde een minder positief beeld op dan de verwachtingen van Microsoft. Het bedrijfsleven ziet obstakels die eerst dienen te worden opgeruimd om .NET tot een succes te maken.

Na een beschrijving van verschillende meningen over .NET, wilde ik het principe zo concreet mogelijk maken door enkele praktijkvoorbeelden van .NET te geven. Microsoft noemt zelf het .NET Passport en .NET Alerts als twee initiatieven die zij zelf heeft ontwikkeld. Deze toepassingen heb ik dan ook als voorbeeld genomen.

> NET Passport is een online service waarbij een consument zich met één e-mailadres en één wachtwoord kan aanmelden bij alle aan de .NET Passport-service deelnemende websites en services.
> …
> NET Alerts is een service van Microsoft dat gebruikers voorziet in berichtgeving op maat. Gebruikers kunnen Alerts ontvangen op hun computer door middel van MSN of Windows Messenger, per e-mail of op een mobiel apparaat als een telefoon of PDA.
> Bedrijven kunnen gebruik maken van deze service om klanten persoonlijke informatie te bieden. Zo is het voor een bank bijvoorbeeld mogelijk om door middel van deze service een klant erop te attenderen dat het saldo van zijn bankrekening onder een bepaald minimum is geraakt.

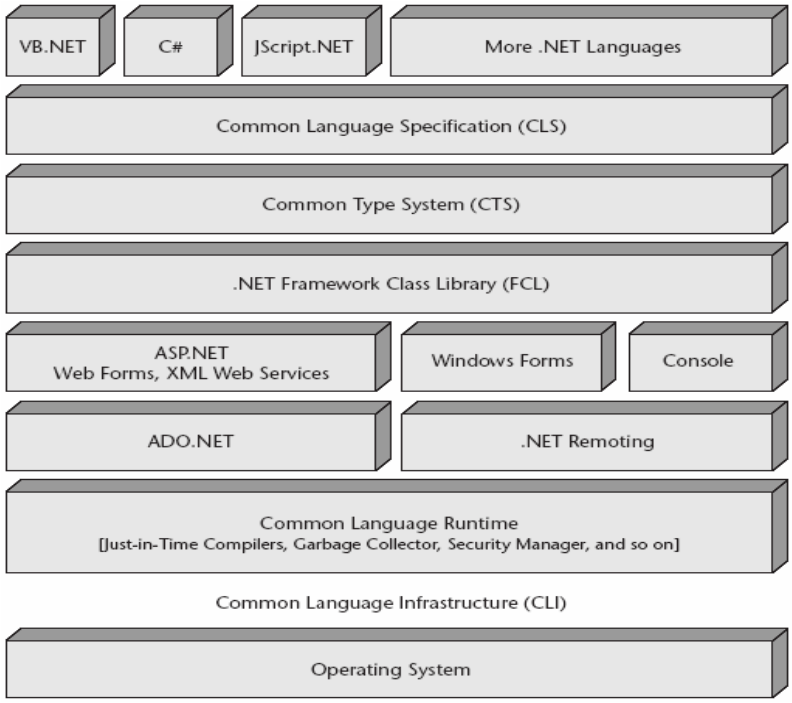
Naast deze toepassingen heb ik gezocht naar enkele praktijkvoorbeelden van XML-webservices. Microsoft bleek op haar website een aantal voorbeelden geplaatst te hebben van implementaties van XML-webservices. Deze voorbeelden heb ik kunnen gebruiken in het rapport.
Ik heb het hoofdstuk afgerond door een conclusie te schrijven over de beschreven onderdelen uit het hoofdstuk.

Het schrijven van dit hoofdstuk heeft ruim twee dagen in beslag genomen. Nadat ik het hoofdstuk had afgerond, heb ik het laten lezen aan mijn begeleider en enkele andere collega's. Deze vonden het stuk verhelderend: .NET was niet langer een vage term. Ook ikzelf had een veel duidelijker beeld van de definitie en reikwijdte van .NET.

5.4.3 Standaarden en technieken van .NET

Om de medewerkers van Reflecta inzicht te bieden in de technische werking van .NET heb ik een hoofdstuk opgenomen, waarin ik de architectuur van het .NET Framework en de belangrijkste standaarden en technieken van .NET heb beschreven.
Omdat ik tijdens mijn opleiding reeds globaal kennis had gemaakt met .NET, had ik een idee welke standaarden en technieken van belang waren om te beschrijven. Tijdens het inventariseren van informatiebronnen was ik deze standaarden en technieken ook al tegengekomen. Inhoudelijke kennis van deze onderwerpen had ik echter vrijwel niet, zodat het noodzakelijk was om eerst onderzoek hiernaar te verrichten, voordat ik erover kon schrijven. Vrijwel alle benodigde informatie kon ik op Internet of in de genoemde tijdschriften vinden.

Als eerste heb ik het .NET Framework onderzocht en beschreven. Het .NET Framework vormt de infrastructuur voor het .NET platform. Kennis van de architectuur van het .NET Framework is noodzakelijk om de werking van .NET te kunnen begrijpen. Door middel van deze kennis wordt inzicht verkregen in hoe .NET programmeertaal- en platformonafhankelijkheid creëert.
Zoeken op Internet leverde enkele documenten op waarin het Framework schematisch werd weergegeven en waarin elk onderdeel werd uitgelegd. Het schematische overzicht van het Framework is hieronder opgenomen.



Figuur 7: *Architectuur .NET Framework*

Elk onderdeel uit het Framework heb ik onderzocht en de werking ervan toegelicht. Hierbij was het soms noodzakelijk om dieper onderzoek te verrichten naar een specifiek onderdeel. Zo werd er bijvoorbeeld regelmatig over zogenaamde “managed code” en “unmanaged code” gesproken, wat voor mij onbekende termen waren. Na onderzoek op Internet heb ik deze termen als volgt verwoord.

.NET maakt onderscheid in zogenaamde *managed* en *unmanaged* programmeertalen. Managed talen zijn Visual Basic .NET en C#. Code die in deze talen is geschreven wordt gecompileerd naar Intermediate Language (IL) en niet naar machine code. De IL wordt opgeslagen in een bestand, de assembly, samen met metadata dat de klassen, methoden en attributen beschrijft die zijn gecreëerd door de ontwikkelaar.
De IL draait in de Common Language Runtime (CLR). Deze laadt en verifieert de assembly om te verzekeren dat de IL correct is. Wanneer methoden worden aangeroepen compileert de CLR deze in machine code die geschikt is voor de machine waar de assembly op draait (Just In Time compilatie). Wanneer de assembly eenmaal draait, verzorgt de CLR services als beveiliging en geheugenbeheer. De applicatie wordt dus ge-*managed* door de CLR.

Code die is geschreven in unmanaged talen worden direct gecompileerd naar machine code. Alle talen die zijn ontwikkeld vóór het ontstaan van .NET zijn unmanaged.
.NET kan deze talen compileren naar managed code, waardoor deze talen alsnog gebruik maken van de CLR.

Na het beschrijven van het Framework heb ik de drie belangrijkste standaarden van .NET, SOAP, UDDI en WSDL, onderzocht en beschreven. Tijdens het inventariseren van informatiebronnen was mij reeds duidelijk geworden dat deze drie standaarden de basis vormden in het standaardisatieproces van .NET.
Van elke standaard heb ik het doel en de werking onderzocht en beschreven, met eventueel een voorbeeld ter verduidelijking. Zo heb ik SOAP (Simple Object Access Protocol) bijvoorbeeld als volgt beschreven.

SOAP definieert een standaard XML-berichtstructuur. Een SOAP-bericht kan worden gezien als een envelop. De envelop bevat het adres (URL) waarnaar het bericht moet worden verstuurd: de header van het SOAP-bericht. De inhoud van de envelop bevat het werkelijke bericht in XML formaat. Dit bericht bestaat onder meer uit de naam van de functionaliteit die wordt aangeroepen, alsmede de parameters die vereist zijn om deze functionaliteit uit te kunnen voeren.

Tijdens de inventarisatie van informatiebronnen kwam ik vaak artikelen tegen die problemen meldden over de beveiliging binnen .NET. Omdat beveiliging van gegevens een belangrijk onderdeel vormt binnen applicatieontwikkeling, heb ik besloten hier ook aandacht aan te besteden. Onderzoek wees uit dat er sinds de lancering van .NET meerdere malen problemen zijn geweest met gegevensbeveiliging. Vooral Microsoft Passport bleek een probleemgeval. Gegevens van gebruikers bleken vrij toegankelijk. De beveiligingsproblemen werden zelfs als een van de oorzaken genoemd van het uitblijven van het succes van .NET.
Ik heb de beveiligingsproblemen alleen genoemd en niet verder onderzocht, omdat dit teveel tijd zou gaan kosten en niet relevant zou zijn. Wel heb ik onderzoek verricht naar de maatregelen die zijn genomen om de problemen te verhelpen. Eén van de resultaten van dat onderzoek was het onderstaande.

Een initiatief dat onafhankelijk ontwikkeld is, is de Security Assertion Markup Language (SAML). SAML is een taal die ontwikkeld is om op een eenduidige manier autorisatie-informatie tussen verschillende beveiligingssystemen, waaronder webservices, te delen. SAML bevat een aantal standaardprotocollen en messaging frameworks als XML Signature, XML Encryption en SOAP.
SAML is als standaard aangenomen door het E-business consortium Organization for the Advancement of Structured Information Standards (OASIS).

Naast een objectieve beschrijving van de standaarden en technieken van .NET wilde ik beschrijven hoe deze standaarden en technieken werden ervaren in de praktijk. Dit zou Reflecta kunnen helpen om een oordeel te vormen over .NET en het gebruik ervan in de praktijk.
Het nieuwe concept van XML-webservices bleek enkele serieuze risico's met zich mee te brengen. Het bedrijfsleven vraagt zich af of Microsoft rekening heeft gehouden met deze risico's en hoe daar mee wordt omgegaan. Ik heb een opsomming gemaakt van de vijf grootste risico's en elk risico kort toegelicht.
Eén van de risico's van XML-webservices is bijvoorbeeld het afhankelijk zijn van de aanbieder van de webservice.

- Afhankelijkheid van aanbieder van de webservice
  De gebruiker van een webservice is afhankelijk van de aanbieder van de webservice. Wanneer de aanbieder de webservice wijzigt of stopt, is de gebruiker daarvan de dupe. Op dit moment zijn tegen dit risico nog geen maatregelen genomen. Waarschijnlijk worden deze pas genomen wanneer dergelijke situaties zich in de praktijk voordoen.

Als laatste ben ik in dit hoofdstuk ingegaan op de standaarden en technieken op het gebied van applicaties ontwikkelen in .NET. Hierbij heb ik onderzocht wat de benodigdheden zijn voor een applicatieontwikkelaar om .NET applicaties te kunnen bouwen.
Microsoft heeft gelijktijdig met de lancering van .NET de nieuwe ontwikkelomgeving Visual Studio .NET uitgebracht. Om .NET programmeertaalonafhankelijk te maken, ondersteunt deze omgeving ongeveer 20 programmeertalen. Op dit moment blijken veel talen geschikt te worden gemaakt voor gebruik binnen .NET, zoals bijvoorbeeld de talen COBOL, Eiffel, Perl, Smalltalk en Python. De meest gebruikte en best ondersteunde .NET talen bleken C++, Visual Basic, ASP en de nieuwe taal C# ("C-Sharp").

Omdat ik tijdens de opdracht gebruik zou maken van de taal C#, heb ik hiernaar extra onderzoek verricht.
De objectgeoriënteerde taal C# bleek momenteel de meest gebruikte taal waarin .NET applicaties worden ontwikkeld. De oorzaak hiervoor is dat C# het beste functioneert in een .NET omgeving, omdat de taal speciaal voor .NET is ontwikkeld. Andere talen hebben uitbreidingen of aanpassingen nodig om goed te kunnen werken in een .NET omgeving. Dit levert in de praktijk vaak problemen op.
De syntax van C# is gebaseerd op die van C++, maar bevat een aantal functionaliteiten die het gebruik eenvoudiger en productiever maakt dan C++, zoals bijvoorbeeld garbage collection.

Na drie dagen heb ik dit hoofdstuk afgerond en ben ik begonnen aan het volgende hoofdstuk, "Mogelijke betekenis van .NET voor Reflecta".

5.4.4 Mogelijke betekenis van .NET voor Reflecta

Eén van de doelen van het rapport over interfacing was om onderzoek te verrichten naar de mogelijkheden van .NET voor Reflecta. Omdat er nog weinig inzicht was in deze mogelijkheden, heb ik een hoofdstuk in het verslag opgenomen dat hierop ingaat.
Als eerste heb ik de mogelijke betekenis van XML-webservices voor Reflecta onderzocht. Gezien de huidige problemen met XML-webservices, heb ik geadviseerd

om nog te wachten met de implementatie van webservices. In de toekomst zouden webservices wel hun dienst kunnen bewijzen voor Reflecta op het gebied van communicatie met draadloze apparaten.

Vervolgens heb ik de nieuwe mogelijkheden van Progress OpenEdge 10 beschreven.

Tot Progress OpenEdge 10 was de gebruikersinterface van een Progress applicatie afhankelijk van de functionaliteiten die Progress bood op dit gebied. De Progress gebruikersinterfaces staan bekend als functioneel, maar weinig aantrekkelijk.
Eén van de doelen die Progress wil bereiken met OpenEdge 10 is “User Interface Independence”: Progress wil de ontwikkelaar zijn eigen gebruikerinterface kunnen laten kiezen.
…
Voor Reflecta biedt dit de mogelijkheid om de gebruikersinterface van Progress applicaties te verbeteren. Door middel van de integratie van OpenEdge 10 met .NET kan een gebruikersinterface geïmplementeerd worden die is gebaseerd op .NET, waardoor een Windows look-and-feel wordt verkregen. De business logica van de applicatie kan (onafhankelijk van de gebruikersinterface) in een Progress-omgeving worden uitgevoerd, op de wijze zoals dit reeds plaatsvindt. (Voor meer technische details hierover wordt verwezen naar hoofdstuk 5 van dit document).

Het implementeren van een .NET gebruikersinterface kent enkele voor- en nadelen ten opzichte van Progress gebruikersinterfaces.
Voordelen zijn:

- Gebruikers zijn reeds vertrouwd met de .NET (Windows) interface, wat de gebruikersvriendelijkheid van een Progress applicatie verhoogt;
- De .NET interface biedt meer mogelijkheden dan een Progress gebruikersinterface;
- Met relatief weinig extra inspanning kan een applicatie een sterk gemoderniseerd uiterlijk krijgen.

Een nadeel is dat onbekendheid met .NET tot gevolg heeft dat eerst onderzoek naar de wijze van technische implementatie moet worden gedaan voordat de .NET gebruikersinterface kan worden geïmplementeerd.

Als laatste mogelijke toepassing van .NET heb ik het verbeteren van de opbouw van de datasource voor Crystal Reports beschreven. Hierbij heb ik met behulp van een voorbeeld uitgelegd op welke wijze de huidige datasource redundante gegevens bevat en dat dit nadelig is voor de performance van het opbouwen van rapporten. Vervolgens heb ik beschreven dat een op .NET dataobject gebaseerde datasource dit probleem kan verhelpen.
De bovenstaande toepassingen van .NET worden in dit project onderzocht en geïmplementeerd door het menu van XL-Enz in een .NET omgeving te implementeren en een .NET dataobject voor Crystal Reports te bouwen.

Wegens tijdsdruk heb ik geen onderzoek verricht naar mogelijke andere toepassingen van .NET. Het zoeken van andere toepassingen van .NET was ook geen doel van de opdracht, zodat ik heb besloten mij te richten op het volgende hoofdstuk van het verslag.

5.4.5 Interfacing tussen Progress en .NET

In het laatste hoofdstuk van het verslag heb ik enerzijds de implementatie van interfacing tussen Progress en .NET en anderzijds de implementatie van interfacing tussen Progress, .NET en Crystal Reports beschreven. Dit heb ik gedaan door middel van voorbeeldimplementaties. Het hoofdstuk is bedoeld voor de ontwikkelaars van Reflecta Automation en heeft tot doel hen op de hoogte te brengen van de wijze waarop .NET samenwerkt met Progress en met Crystal Reports.
In overleg met de begeleider heb ik besloten om dit hoofdstuk van het verslag aan te vullen gedurende het verloop van het project. Dit vanwege het feit dat er tijdens de implementatiefase nieuwe kennis zou worden opgedaan op het gebied van interfacing. Deze kennis zou zodoende nog verwerkt kunnen worden in het rapport.

Als eerste heb ik in dit hoofdstuk de nieuwe mogelijkheden van Progress OpenEdge 10 onderzocht en beschreven. Hiervoor heb ik documentatie gebruikt die Progress beschikbaar stelde op haar website. Deze documentatie bestond voornamelijk uit “Beta Training” in de vorm van Microsoft PowerPoint presentaties, waarin de mogelijkheden van interfacing naar .NET werden behandeld. Tijdens een bijeenkomst van de Progress User Group (PUG) eind 2003 zijn deze mogelijkheden ook behandeld. De presentaties die toen zijn gegeven had ik ook ter beschikking als informatiebron.
Ik heb onderzocht en beschreven wat de achterliggende strategie van Progress is om de mogelijkheid van interfacing naar .NET te bieden in OpenEdge 10. Deze mogelijkheid bleek een onderdeel te zijn van de “User Interface Independence” strategie: naast de Progress gebruikersinterfaces is het in OpenEdge 10 mogelijk om zogenaamde “Open Clients” te gebruiken zoals Java, Webservices en .NET gebruikersinterfaces.
De ondersteuning voor .NET gebruikersinterfaces komt voort uit het feit dat Microsoft gebruikersinterfaces de meest bekende zijn:

> “We are focused on the Microsoft .NET client integration at this time because it is the leading user interface in the industry. At this point in time it is clear that Microsoft, through their operating systems and products, define the most used look and feel.”
> Bron: OpenEdge 10 Beta Training, PowerPoint Presentatie “.NET Open Client”, slide 14

Vervolgens heb ik voor- en nadelen van de implementatie van een .NET gebruikersinterface onderzocht en beschreven.

Het gebruik van een Open Client gebruikersinterface heeft enkele voor- en nadelen.
Voordelen zijn:

- Flexibiliteit in het inzetten van een gebruikersinterface naar keuze;
- Applicaties kunnen een moderne look-and-feel krijgen;
- Wanneer de eisen en trends veranderen op het gebied van gebruikersinterfaces, hoeft slechts de interface zelf aangepast te worden, zonder dat dit consequenties heeft voor de business logica.

Nadelen zijn:

- Implementatie van een Open Client gebruikersinterface kost extra tijd, omdat er onderzoek dient te worden verricht naar de werking van Open Clients. Daarnaast kan implementatie in bijvoorbeeld een Java of .NET omgeving kennis vereisen die niet in huis is;
- De applicatie krijgt een extra laag (bijvoorbeeld een .NET of Java laag), wat onwenselijk kan zijn, omdat dit vertragend kan werken.

Na deze mogelijkheden van OpenEdge 10 te hebben beschreven, heb ik de twee belangrijkste dataobjecten beschreven die worden gebruikt voor interfacing tussen Progress en .NET. In Progress is dat het ProDataSet-object en in .NET is dat het ADO.NET DataSet-object. Zoals reeds beschreven is een ProDataSet-object één op één te converteren naar een ADO.NET DataSet-object, wat het mogelijk maakt om data uit een Progress database beschikbaar te hebben in een .NET omgeving.

Het ProDataSet-object heb ik beschreven op basis van de kennis die ik heb opgedaan bij de testimplementatie van dit object tijdens de activiteit “Kennis vergaren van Progress”. Als eerste heb ik de opbouw en het doel van het object beschreven. Vervolgens heb ik stapsgewijs de implementatie van een ProDataSet-object beschreven door middel van een voorbeeld. In dit voorbeeld heb ik in vijf stappen de Progress broncode gegeven waarmee een ProDataSet-object wordt aangemaakt dat twee tabellen “ttCustomer” en “ttOrder” bevat. Hierbij heb ik het definiëren van de tabellen, de onderlinge relaties en het vullen van het ProDataSet-object beschreven. Bij elke stap heb ik aandachtspunten en verduidelijking op de gebruikte statements beschreven. Tijdens de testimplementaties ben ik tegen enkele beperkingen aangelopen (zie paragraaf 5.2.3 “Progress ProDataSet”). Om de ontwikkelaars van Refecta op de hoogte te brengen van deze beperkingen, heb ik deze beperkingen (en enkele andere beperkingen die Progress zelf meldt) beschreven. Zo meldt Progress bijvoorbeeld dat OpenEdge 10 geen batching ondersteunt bij het doorgeven van ProDataSets tussen een server en een client:

> “Batching is not currently available in this release. Be advised that if you select a large ProDataSet it will all be passed at once. If you want to do batching it will have to be handled programmatically at this point. Development is working on a strategy. Look for future white papers on this topic.”
> Bron: OpenEdge 10 Beta Training, PowerPoint Presentatie “ProDataSet”, slide 19

Het ADO.NET DataSet-object heb ik op vrijwel gelijke wijze beschreven als het ProDataSet-object, echter zonder een voorbeeldimplementatie in broncode. Ik heb hiervoor gekozen omdat tijdens de implementatiefase van het project bleek dat het niet noodzakelijk was om de volledige syntax van de definitie van een ADO.NET DataSet te kennen. In plaats van een voorbeeld met broncode heb ik de onderdelen van het DataSet-object schematisch weergegeven en uitgelegd.

Tijdens de implementatiefase van het project liep ik tegen een probleem aan bij het gebruik van samengestelde primaire sleutels in ADO.NET DataSets. Om de ontwikkelaars van Reflecta op de hoogte te brengen hiervan, heb ik het probleem als volgt omschreven.

Implementatie van de ADO.NET DataSet, gebaseerd op een Progress ProDataSet, heeft een beperking van deze techniek aan het licht gebracht, waarvoor tot op heden nog geen afdoende oplossing voor is gevonden.
In Progress werd een ProDataSet gedefinieerd waarbij een tabel werd opgenomen met een samengestelde primaire sleutel. De sleutel bestond uit een combinatie van twee velden uit de tabel. De tabel bevatte meerdere records.
Wanneer deze ProDataSet naar een ADO.NET DataSet werd geconverteerd, bleek de genoemde tabel slechts één record te bevatten.
Dit bleek te worden veroorzaakt doordat de tabel een samengestelde primaire sleutel bevatte. Wanneer de sleutelgegevens werden opgevraagd van de tabel uit de ADO.NET DataSet, bleek dat de primaire sleutel van de tabel wel een samengestelde sleutel was. Deze definitie was dus correct overgenomen uit de Progress ProDataSet.
Wanneer de samengestelde primaire sleutel werd veranderd naar een enkelvoudige sleutel (in de Progress ProDataSet), trad het probleem niet op.

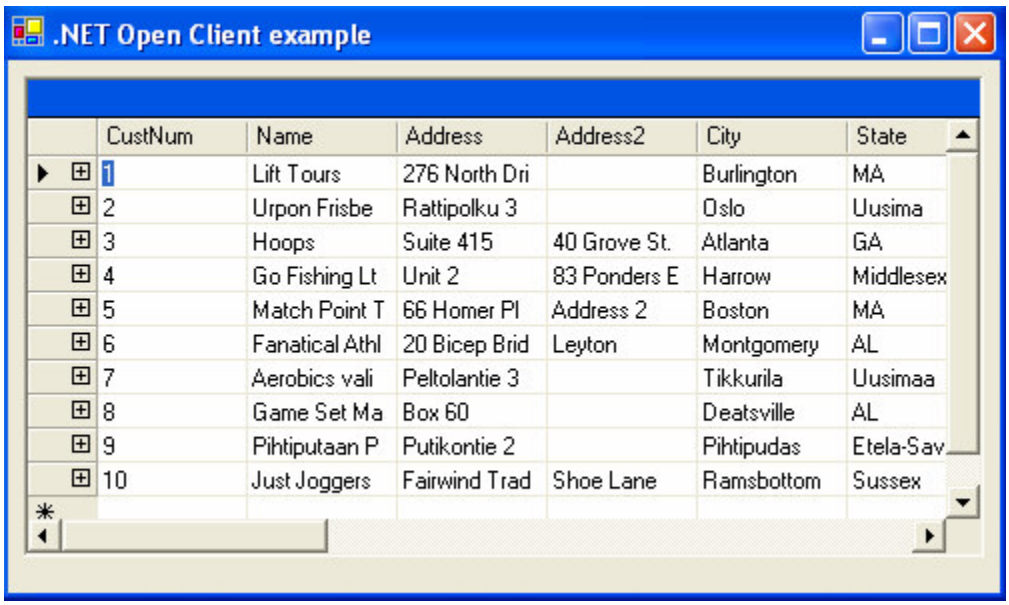
Na de twee belangrijkste dataobjecten uit beide omgevingen te hebben beschreven, heb ik aan de hand van een voorbeeld stapsgewijs de implementatie van interfacing tussen Progress en .NET beschreven.
Met behulp van de documentatie die Progress hierover beschikbaar stelt, heb ik als eerste een globaal stappenplan opgezet van de activiteiten die dienen te worden uitgevoerd om interfacing tussen Progress en .NET te implementeren.
Dit stappenplan bestond uit de volgende stappen:

1. Bouwen Progress procedures;
2. Genereren proxy tussen Progress en .NET;
3. Bouwen .NET client applicatie;
4. Invoeren .NET client applicatie.

Het stappenplan heb ik in eerste instantie niet verder uitgewerkt in het rapport. De kennis die ik had van de werking van interfacing leek na het maken van het stappenplan voldoende om het te ontwikkelen menu te ontwerpen. In overleg met de begeleider heb ik besloten om de uitwerking van het stappenplan verder aan te vullen gedurende het project, wanneer meer kennis over de implementatie van interfacing wordt opgedaan. Dit heb ik dan ook gedaan na het ontwikkelen van de pilot waarin het menu is geïmplementeerd.
Per stap uit het stappenplan heb ik toen met behulp van broncode en voorbeeldschermen een voorbeeldimplementatie beschreven van een .NET client applicatie waarin gegevens uit een Progress database worden getoond. Per stap heb ik tevens de problemen beschreven die ik had ondervonden, om zodoende de ontwikkelaars van Reflecta op de hoogte te brengen van bugs en problemen bij de implementatie van interfacing.
Zo liep ik bijvoorbeeld bij stap 2 “Genereren proxy tussen Progress en .NET” tegen een probleem aan. Om interfacing tussen Progress en .NET te implementeren is het noodzakelijk om een proxy-object te genereren. Dit kan worden gedaan door middel van de tool ProxyGen die wordt meegeleverd met Progress OpenEdge 10. Het proxy-object fungeert als een vertaler tussen de omgevingen. Het object neemt de data en communicatie vanuit de Progress of .NET omgeving in ontvangst en vertaalt deze naar begrijpelijke data voor de andere omgeving.
Omdat Progress OpenEdge 10 Alfa nog niet beschikbaar was, heb ik in eerste instantie gebruik gemaakt van een betaversie van OpenEdge 10. De versie van ProxyGen die bij OpenEdge10 Beta werd meegeleverd, bleek niet bruikbaar. Dit heb ik als volgt verwoord.

Wanneer de versie van ProxyGen wordt gebruikt die bij OpenEdge 10 Beta wordt meegeleverd, kan het genereren van een proxy mislukken. De reden die hiervoor wordt genoemd in de Progress Knowlegde Center (ID: P36138) is dat de Microsoft Java SDK niet is geïnstalleerd.
Deze SDK wordt echter niet meer aangeboden door Microsoft, waardoor het verkrijgen van de SDK een probleem vormt. Dit probleem is op te lossen door de versie van ProxyGen te gebruiken die bij OpenEdge 10A wordt meegeleverd. In deze versie komt dit probleem niet voor.

Het resultaat van de voorbeeldimplementatie was een .NET client applicatie, waarmee gegevens uit een Progress database worden opgevraagd en getoond in een zogenaamd “DataGrid”. Hieronder is een afbeelding opgenomen van deze applicatie.



Figuur 8: *Resultaat voorbeeldimplementatie .NET client applicatie*

5.4.6 Interfacing tussen Progress, .NET en Crystal Reports

Na het opstellen van het genoemde stappenplan voor interfacing tussen Progress en .NET werd duidelijk dat deze interfacing altijd via een proxy-object verloopt, waarmee de “vertaling” tussen de twee omgevingen wordt afgehandeld.
Na overleg met het ontwikkelteam van Reflecta Automation werd besloten dat deze manier van interfacing niet geschikt is voor het implementeren van het .NET dataobject dat als datasource dient voor Crystal Reports. Het feit dat alle data voor een rapport via een proxy-object dient te worden ingelezen in Crystal Reports, maakt deze manier te omslachtig en waarschijnlijk te traag.

Als gevolg van dit besluit was het noodzakelijk om een alternatief te vinden om een .NET datasource voor Crystal Reports te implementeren.
Binnen Reflecta was al bekend dat het mogelijk was om in Crystal Reports een ADO.NET XML bestand te gebruiken als datasource voor rapporten. Het was voor zowel de ontwikkelaars van Reflecta als voor mij onbekend hoe een dergelijk bestand is opgebouwd. Daarom heb ik op Internet gezocht naar meer informatie over ADO.NET

XML bestanden. Hieruit bleek dat een ADO.NET DataSet als XML kan worden opgeslagen. De definitie van de DataSet (tabellen, kolommen, datatypen, relaties etc.) wordt opgeslagen in een XML Schema Definition (XSD) bestand. De data uit de DataSet wordt opgeslagen in een XML bestand, waarin een verwijzing is opgenomen naar het XSD bestand dat de definitie van de data bevat.

Het was bekend dat een Progress ProDataSet één op één te converteren is naar een ADO.NET DataSet en dat Progress functionaliteiten biedt om XML bestanden te genereren. Het alternatief dat door de ontwikkelaars van Reflecta werd voorgesteld was dan ook het volgende:

1. Bouw een Progress ProDataSet-object dat de rapportdata bevat;
2. Exporteer de definitie van de ProDataSet naar een XSD bestand;
3. Exporteer de data uit de ProDataSet naar een XML bestand;
4. Neem in het XML bestand een verwijzing op naar het XSD bestand;
5. Lees het XML bestand in Crystal Reports in als ADO.NET XML datasource.

Op deze manier zou er dus geen proxy-object benodigd zijn als “vertaler”. In overleg met de begeleider is besloten om dit alternatief te ontwikkelen. Ik was echter onbekend met zowel Crystal Reports, XML en XSD als met de mogelijkheden van Progress om data naar XML te exporteren.
Om hier kennis van op te doen heb ik als eerste op Internet gezocht naar XML en XSD bestanden, waarin DataSets waren opgenomen die meerdere tabellen bevatten. Zonder kennis te hebben van de opbouw van dergelijke bestanden, heb ik deze als test ingelezen als ADO.NET XML datasource in Crystal Reports. De XML bestanden werden wel ingelezen en de tabellen uit de DataSet waren ook correct weergegeven, maar de definitie van de tabellen zoals die was opgenomen in het XSD bestand werden niet correct weergegeven. Zo hadden bijvoorbeeld alle velden het datatype “string” met een maximale lengte van 65534 karakters, terwijl in het XSD bestand andere datatypen waren gedefinieerd.
Ik heb onderzocht wat daarvan de reden zou kunnen zijn door op Internet te zoeken naar dit probleem. Dit leverde echter geen oplossing of verklaring op. Tests met andere XML en XSD bestanden hadden hetzelfde resultaat als hierboven beschreven is. Wanneer dit probleem niet kon worden opgelost, zou het implementeren van een dergelijke .NET datasource niet zinvol zijn, omdat het noodzakelijk is dat elk veld het juiste datatype heeft (om bijvoorbeeld berekeningen uit te kunnen voeren).

De begeleider kwam daarop met het voorstel om een test uit te voeren waarbij de definitie (XSD) en de data (XML) in één XML bestand was opgenomen. De datasource die reeds wordt gebruikt voor Crystal Reports werkt ook op deze manier, zij het dan dat deze datasource niet meerdere tabellen kan bevatten in het XML bestand.
Om deze test uit te kunnen voeren was het noodzakelijk om kennis van XML en XSD op te doen. Dit heb ik gedaan door op Internet informatie te zoeken over de opbouw van dergelijke bestanden en door de documentatie van Visual Studio .NET 2003 over dit onderwerp te bestuderen. Ook heb ik de opbouw van XML bestanden bestudeerd die reeds worden gebruikt als datasource voor Crystal Reports.
Door middel van “trial and error” heb ik vervolgens in Visual Studio .NET 2003 een XML bestand gemaakt waarin zowel de definitie van een DataSet als de data uit een DataSet was opgenomen. Als test had ik een DataSet gemaakt met twee tabellen, “Customer” en “Order”, die een onderlinge relatie hadden.

Na twee dagen lukte het om dit XML bestand in Crystal Reports als ADO.NET XML datasource in te lezen. De datatypen werden nu wel correct weergegeven, evenals de relatie tussen de tabellen die ik had gedefinieerd in het XML bestand.
De test was dus geslaagd, waarop werd besloten om het .NET dataobject op deze manier te ontwikkelen.
Na deze beslissing heb ik in het rapport over interfacing een stappenplan beschreven voor de ontwikkeling van een .NET dataobject als datasource voor Crystal Reports.

1. Bouw een Progress ProDataSet-object dat de rapportdata bevat;
2. Exporteer de definitie en de data uit het ProDataSet-objext naar een XML bestand dat is opgebouwd als een ADO.NET XML bestand;
3. Lees het XML bestand in Crystal Reports in als ADO.NET XML datasource.

Het stappenplan heb ik in eerste instantie niet verder uitgewerkt in het rapport. De kennis die ik had van de wijze waarop het dataobject moest worden opgebouwd, leek voldoende om het dataobject te kunnen ontwerpen. In overleg met de begeleider heb ik besloten om de uitwerking van het stappenplan verder aan te vullen gedurende het project, wanneer meer kennis over de implementatie van het dataobject wordt opgedaan.
Dit heb ik dan ook gedaan na het ontwikkelen van de pilot waarin het dataobject is geïmplementeerd. Hierbij heb ik dezelfde werkwijze gehanteerd als voor het beschrijven van de werking van interfacing tussen Progress en .NET. Per stap uit het stappenplan heb ik met behulp van broncode en voorbeeldschermen een voorbeeldimplementatie beschreven van een .NET dataobject in de vorm van een XML bestand.

Zo heb ik bij stap 2 een voorbeeld gegeven van hoe vanuit Progress een XML bestand wordt opgebouwd. Daarbij heb ik de statements toegelicht die daarvoor benodigd zijn en heb ik een “template” in het rapport opgenomen van de vereiste opbouw van het XML bestand. Het maken van een “template” vond ik noodzakelijk omdat de opbouw van het XML bestand aan zeer specifieke eisen moet voldoen, wil het bestand als geldige ADO.NET XML datasource worden herkend in Crystal Reports. Door middel van de “template” kan worden gecontroleerd of het XML bestand aan deze eisen voldoet.

Als laatste heb ik enkele problemen beschreven die ik ben tegengekomen tijdens de implementatiefase. Het grootste probleem was de slechte performance van het inlezen van rapporten waarbij data uit verschillende tabellen benodigd is.

> Wanneer rapporten worden opgebouwd waarbij data uit verschillende tabellen benodigd is, daalt de performance sterk.
> Bij XML bestanden met een grootte tot 1 MB is de performance nog vergelijkbaar met het inlezen van data uit één tabel (getest tot bijna 2000 records).
> Wanneer echter XML bestanden worden ingelezen met een grootte van enkele MB’s (getest tot ruim 5 MB) en daarbij meerdere tabellen bevatten (getest tot 5 tabellen), daalt de performance in veel gevallen sterk.
> De mate van daling van de performance verschilt per combinatie van tabellen. Op dit moment is het onduidelijk wat daarvan de reden is.
> In het slechtste geval worden nog geen 50 records per seconde ingelezen (getest op de combinatie van de tabellen Customer en Order uit de Sports2000 database).

Dit probleem is tot op heden niet opgelost en staat een succesvolle implementatie van het .NET dataobject in de weg. Ik kom hier nog op terug in hoofdstuk 7, waar de ontwikkeling van het .NET dataobject wordt beschreven.

5.5 Afronding

In overleg met de begeleider heb ik met de afronding en oplevering van het rapport gewacht tot het eind van de projectperiode. Gedurende het gehele project heb ik zodoende nieuw opgedane kennis alsnog kunnen verwerken in het rapport.

In de conclusie van het rapport heb ik vermeld dat eerste implementaties van interfacing tussen Progress en .NET hebben uitgewezen dat deze interfacing naar tevredenheid functioneert. Het is mogelijk om een .NET gebruikersinterface te implementeren voor een Progress applicatie. Daarnaast heb ik vermeld dat interfacing tussen Progress, .NET en Crystal Reports technisch realiseerbaar is, maar dat momenteel enkele knelpunten een succesvolle implementatie van het dataobject in de weg staan.

Na het controleren van de rapportopmaak heb ik het rapport opgeleverd. De begeleider heeft het rapport vervolgens doorgenomen en is akkoord gegaan met de inhoud ervan. Het aantal benodigde dagen voor de fase “Onderzoek verrichten naar interfacing” kwam vrijwel overeen met het aantal geplande dagen. De fase heeft totaal ongeveer 25 dagen in beslag genomen. Hiervan heb ik ongeveer 15 dagen aan het schrijven van het rapport over interfacing besteed, verdeeld over de gehele projectperiode.
Het rapport is als externe bijlage opgenomen bij dit verslag.

6. Definitiestudie

6.1 Inleiding

Nadat het besluit werd genomen om het laatste hoofdstuk uit het rapport .NET interfacing open te laten gedurende het project, ben ik begonnen met de eerste fase uit het IAD ontwikkeltraject, de definitiestudie. Deze fase had tot doel de eisen aan de data- en gebruikersinterface van het menu van XL-Enz en het dataobject van Crystal Reports vast te stellen en een systeemconcept hiervoor te ontwikkelen.
Ook had deze fase tot doel om het systeemconcept te clusteren tot eenheden die kunnen dienen als zelfstandig te implementeren pilots. De definitiestudie vormt de basis voor de volgende fase van IAD, de pilotontwikkeling, waarin de pilots op iteratieve wijze worden ontworpen en geïmplementeerd.
Het resultaat van deze fase was een document definitiestudie. Dit hoofdstuk beschrijft de totstandkoming van dit document.
De opbouw van dit hoofdstuk is als volgt. Als eerste wordt in paragraaf 6.2 de totstandkoming van het plan van aanpak voor deze fase beschreven met daarbij het ontwikkelscenario. In paragraaf 6.3 ga ik in op de systeemeisen, waarna ik in paragraaf 6.4 de totstandkoming van het systeemconcept beschrijf. Vervolgens wordt in paragraaf 6.5 het ontwerpen van de technische structuur beschreven. Hierna wordt in paragraaf 6.6 de totstandkoming van het pilotplan behandeld. Paragraaf 6.7 vormt de afsluiting van dit hoofdstuk, waarin ik de afronding en oplevering van het document definitiestudie beschrijf.

6.2 Plan van aanpak en ontwikkelscenario

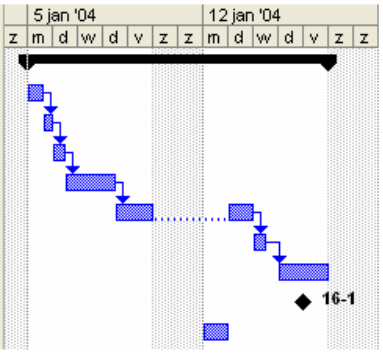
IAD schrijft als eerste activiteit binnen de fase definitiestudie voor om een plan van aanpak op te stellen voor deze fase. Omdat ik weinig ervaring had met IAD, heb ik mij eerst georiënteerd op de fase definitiestudie. Om een planning van de uit te voeren activiteiten te kunnen maken, heb ik het boek “IAD – Het evolutionair ontwikkelen van informatiesystemen” door R.J.H. Tolido bestudeerd. Bij elke activiteit binnen de fase definitiestudie die hierin wordt voorgeschreven heb ik gekeken of deze relevant was voor dit project.
Vervolgens heb ik een overzicht gemaakt van de uit te voeren activiteiten tijdens de fase definitiestudie. Voor de activiteiten die niet relevant bleken, heb ik een reden gegeven waarom ik deze niet relevant vond.

De volgende activiteiten die IAD voorschrijft worden tijdens dit project niet uitgevoerd:

- Voorbereiden pilotplan-workshop
  Reden: Het project is te klein om een uitgebreide workshop te houden. Daarnaast zijn er te weinig betrokkenen bij het project om een workshop zinvol te laten zijn. De systeemeisen en het systeemconcept kunnen in overleg met de begeleider van het project worden vastgesteld.
- Evalueren pilot
  Reden: er zijn geen eerdere pilots om te evalueren. Gekozen is om de definitiestudie eenmalig uit te voeren, waardoor deze activiteit overbodig wordt.
- Beschouwen organisatorische inrichting
  Reden: De invoering van de pilots heeft geen impact op de organisatie, waardoor deze activiteit overbodig wordt.

Van de relevante activiteiten heb ik op basis van de beschrijving uit het boek van Tolido een inschatting gemaakt van de benodigde tijd. Hierop is de onderstaande planning gebaseerd.

	Taaknaam	Duur	Begindatum	Einddatum
1	**Uitvoeren definitiestudie**	**80 uur**	**ma 5-1-04**	**vr 16-1-04**
2	Orientatie IAD / Definitiestudie	6 uur	ma 5-1-04	ma 5-1-04
3	Opstellen plan van aanpak definitiestudie	2 uur	ma 5-1-04	ma 5-1-04
4	Definiëren ontwikkelscenario	4 uur	di 6-1-04	di 6-1-04
5	Definiëren systeemeisen	16 uur	di 6-1-04	do 8-1-04
6	Bepalen systeemconcept	20 uur	do 8-1-04	di 13-1-04
7	Beschouwen technische structuur	4 uur	wo 14-1-04	wo 14-1-04
8	Opstellen pilotplan	16 uur	do 15-1-04	vr 16-1-04
9	Opleveren definitiestudie	1 uur	vr 16-1-04	vr 16-1-04
10	**Eindverslag schrijven**	**8 uur**	**ma 12-1-04**	**ma 12-1-04**



Figuur 9: *Faseplanning definitiestudie*

De doorlooptijd van de fase definitiestudie is enkele dagen korter gepland dan bij de planning die bij aanvang van het project is opgesteld. Dit vanwege het ontbreken van enkele activiteiten die bij nader inzien irrelevant waren en vanwege een inschatting op basis van nieuwe inzichten.
In het plan van aanpak heb ik naast een planning ook een ontwikkelscenario opgenomen. Het beschrijven van het ontwikkelscenario had tot doel om zorg te dragen voor de juiste wijze van het gebruik van de iteratieve ontwikkelstrategie. Als eerste heb ik de globale pilotstrategie beschreven. Op basis van het inzicht in het project dat ik op het moment van het schrijven van het ontwikkelscenario had, heb ik het project verdeeld in twee pilots.

- Pilot 1: “Ontwerp en implementatie van het .NET dataobject voor Crystal Reports”;
- Pilot 2: “Ontwerp en implementatie van het menu van XL-Enz”.

Vervolgens heb ik de iteratieve wijze van ontwikkeling van deze pilots globaal beschreven. Ook heb ik de globale teststrategie beschreven, door te vermelden dat elk geïmplementeerd pilotdeel wordt getest volgens de teststrategie die is beschreven in het document plan van aanpak voor het gehele project. De acceptatietest is gepland tijdens de fase invoering en wordt uitgevoerd door de begeleider en/of opdrachtgever.

6.3 Systeemeisen

Na het opstellen van het plan van aanpak heb ik de systeemeisen vastgesteld. Het doel van deze activiteit was het opstellen van een geprioriteerde lijst van eisen die de behoeften aan het menu van XL-Enz en het dataobject voor Crystal Reports weergeeft. Deze lijst zou als basis dienen voor alle verdere ontwerpactiviteiten.
Bij het opstellen van de eisen heb ik het project verdeeld volgens de indeling in pilots die ik tijdens het ontwikkelscenario reeds had opgesteld. Dit vanwege het feit dat het menu van XL-Enz en het dataobject voor Crystal Reports los van elkaar staande toepassingen zijn.

6.3.1 Eisen aan menu XL-Enz

Als eerste heb ik de eisen aan het menu vastgesteld. Hierbij ben ik uitgegaan van de eisen die tijdens het opstellen van de opdrachtomschrijving reeds zijn gedefinieerd.
De eisen heb ik verdeeld in functionele en niet-functionele eisen. De functionele eisen beschrijven de functies die de gebruiker met het menu kan uitvoeren. De niet-functionele eisen zijn de eisen die niet inhoudelijk met het systeem te maken hebben, maar die randvoorwaarden stellen aan de manier waarop de functionele eisen dienen te worden geïmplementeerd.
De eisen heb ik geprioriteerd door middel van een *MoSCoW* analyse. De *MoSCoW* analyse kent de volgende indeling in prioriteiten:

- Must Have (M);
- Should Have (S);
- Could Have (C);
- Would like to but probably cannot have (W).

Tijdens de module voorbereidend op het afstuderen is deze analyse mij aangeleerd en zij bleek in de praktijk zeer goed bruikbaar om systeemeisen te prioriteren.
Nadat ik de eisen die reeds vastgesteld waren in de opdrachtomschrijving had geprioriteerd en verdeeld in functionele en niet-functionele eisen, heb ik het resultaat hiervan besproken met de begeleider. Deze ging akkoord met de opgestelde eisen en had nog enkele aanvullende eisen. Deze eisen lagen op het gebied van autorisatie van gebruikers en enkele niet-functionele eisen met een lage prioriteit. Na toevoeging van deze eisen was het resultaat de onderstaande lijst systeemeisen.

ID	Omschrijving	MoSCoW
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma's van XL-Enz kunnen worden opgestart	M
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer	M
MF3	Een gebruiker kan het menu afsluiten	M
MF4	Een gebruiker kan menu-items en programma's uit het menu knippen, slepen, kopiëren, plakken, hernoemen, sorteren, toevoegen en verwijderen	S
MF5	Een gebruiker kan gemaakte wijzigingen in de opbouw van het menu opslaan	S
MF6	Een gebruiker kan wijzigingen in het menu die nog niet zijn opgeslagen ongedaan maken	W

Figuur 10: *Functionele eisen menu XL-Enz*

ID	Omschrijving	MoSCoW
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database	M
MN2	Het menu-dataobject is te converteren van een Progress object naar een ADO.NET object en vice versa	M
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven	M
MN4	Het menu van XL-Enz bevat een Active X object, dat wordt opgebouwd op basis van een ADO.NET object	M
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress object en vervolgens doorgegeven aan de client, waar het object wordt geconverteerd naar een ADO.NET object	M
MN6	Zolang de gebruiker de wijzigingen in de opbouw van het menu niet opslaat, worden de wijzigingen bijgehouden in het ADO.NET object dat zich op de client bevindt	S
MN7	Bij het opslaan van wijzigingen in de opbouw van het menu worden de wijzigingen opgeslagen in de Progress database	S
MN8	Het menu wordt opgebouwd en weergegeven in minder dan 10 seconden	S
MN9	Het menu bevat autorisatie per gebruikersgroep en per gebruiker	C
MN10	Een gebruiker kan alleen die programma's starten en die bewerkingen uitvoeren waar hij autorisatie voor heeft	C
MN11	Een beheerder (administrator) ziet de menu's van alle gebruikersgroepen en kan deze wijzigen	C
MN12	Een gebruiker beschikt over een "Favorietenfolder" waarin de gebruiker naar eigen inzicht programma's kan plaatsen, bewerken en verwijderen	C

Figuur 11: *Niet-functionele eisen menu XL-Enz*

6.3.2 Eisen aan dataobject Crystal Reports

De eisen aan het dataobject voor Crystal Reports zijn op dezelfde wijze vastgesteld als de eisen aan het menu van XL-Enz. De eisen aan het dataobject omvatten voornamelijk niet-functionele eisen, omdat het dataobject geen (extra) functionaliteiten biedt voor de gebruiker. De volgende systeemeisen zijn vastgesteld.

ID	Omschrijving	MoSCoW
DF1	Een gebruiker is in staat om Crystal Reports een rapport te laten genereren op basis van een .NET dataobject als datasource	M

Figuur 12: *Functionele eis dataobject Crystal Reports*

ID	Omschrijving	MoSCoW
DN1	Het dataobject kan gegevens uit een bestaande Progress database inlezen	M
DN2	Het dataobject is te converteren van een Progress object naar een ADO.NET object	M
DN3	Het dataobject wordt opgebouwd op de server als een Progress object, vervolgens geconverteerd naar een ADO.NET object en daarna verstuurd naar client	M
DN4	De tijd die benodigd is om een rapport op te bouwen op basis van een .NET dataobject als datasource, dient minstens gelijk te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject	M

Figuur 13: *Niet-functionele eisen dataobject Crystal Reports*

Het vaststellen van de performance-eis (DN4) vormde een probleem. De tijd die benodigd is om een rapport op te bouwen is namelijk afhankelijk van de hoeveelheid gegevens die benodigd is voor een rapport. Deze hoeveelheid kan sterk variëren. In overleg met de begeleider heb ik als performance-eis aangehouden dat de benodigde tijd voor de opbouw van een rapport minstens gelijk dient te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject.

6.4 Systeemconcept

Op basis van de systeemeisen heb ik vervolgens een systeemconcept ontworpen. Dit systeemconcept is een eerste functionele, gebruikersgerichte beschrijving van het systeem en dient als basis voor het opstellen van een pilotplan en voor verdere ontwikkeling in de fase pilontwikkeling.
Het systeemconcept heb ik gemodelleerd met behulp van de technieken van UML. Omdat ik geen ervaring had met het toepassen van deze technieken in een IAD ontwikkeltraject, heb ik eerst theorie over UML bestudeerd om vervolgens de verschillende UML diagrammen te plaatsen in het IAD ontwikkeltraject. Hiervoor heb ik het boek “Praktisch UML” (J. Warmer en A. Kleppe) bestudeerd. Daaruit bleek dat use-cases, het use-case-diagram en het klassediagram toepasbaar waren in de definitiefase. Om die reden heb ik deze diagrammen, voor zover relevant voor dit project, toegepast om het systeemconcept te ontwikkelen.

6.4.1 Systeemconcept menu XL-Enz

Als eerste heb ik de functionaliteiten van het menu van XL-Enz gemodelleerd door middel van UML use-cases en een use-case-diagram. De functionele systeemeisen vormden hiervoor de basis. Door middel van use-cases wordt de interactie tussen de gebruiker en het systeem gemodelleerd. Elke use-case bestaat uit een tekstuele beschrijving van de functie die een gebruiker kan uitvoeren met het systeem.
Voor elke functionele systeemeis heb ik een use-case gemaakt. Zo ziet bijvoorbeeld de use-case voor het opslaan van wijzigingen in het menu er als volgt uit.

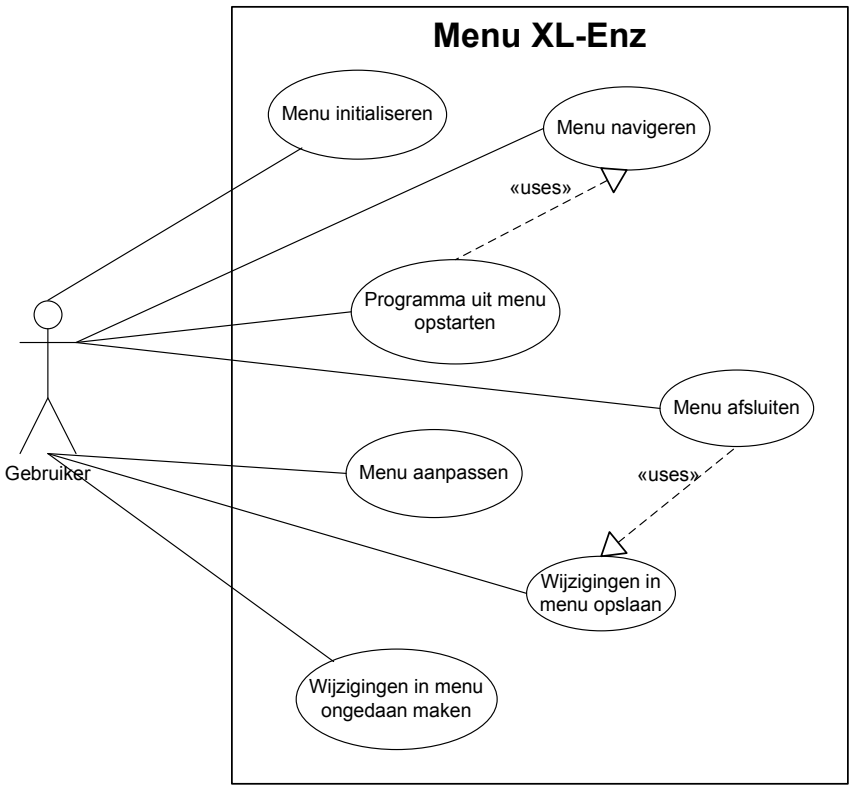
Naam:	Wijzigingen in menu opslaan
Aannamen:	Menu wordt getoond en één of meer menu-items zijn gewijzigd
Beschrijving:	1. De gebruiker geeft aan wijzigingen in het menu op te willen slaan; 2. Het systeem vraagt een bevestiging; 3. Na bevestiging slaat het systeem de wijzigingen op in de database; 4. Het systeem meldt dat de wijzigingen zijn opgeslagen.
Uitzonderingen:	Opslaan mislukt: Het systeem geeft een melding dat het opslaan in de database niet is gelukt. Het opslaan wordt geannuleerd
Resultaat:	De wijzigingen die de gebruiker in het menu heeft gemaakt zijn opgeslagen in de database
Dekt systeemeis:	MF5

Figuur 14: *Use-case "Wijzigingen in menu opslaan"*

Naast deze use-case heb ik use-cases gemaakt voor de volgende functionaliteiten:

- Menu initialiseren;
- Menu navigeren;
- Programma uit menu opstarten;
- Menu afsluiten;
- Menu aanpassen;
- Wijzigingen in menu ongedaan maken.

Met deze use-cases werd elke functionele systeemeis gedekt. Op basis van deze use-cases heb ik vervolgens een use-case-diagram ontworpen. Een use-case-diagram geeft een overzicht van alle functionaliteiten die door een gebruiker (actor) kunnen worden uitgevoerd met het systeem.

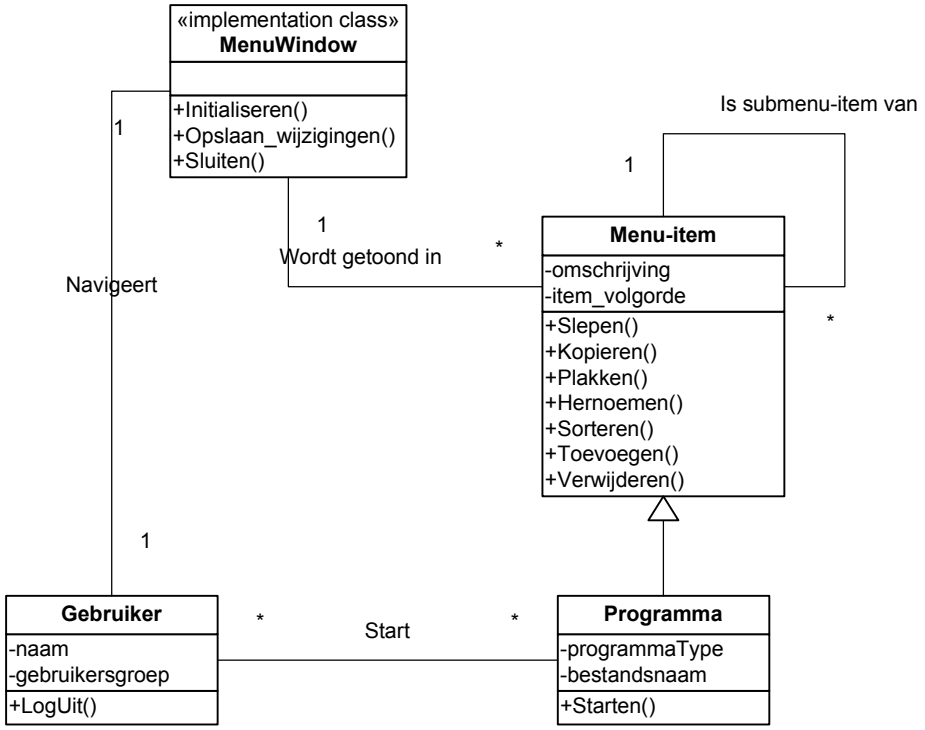


Figuur 15: *Use-case-diagram menu XL-Enz*

Het diagram heeft niet veel informatie toegevoegd, maar geeft wel een overzicht van de mogelijkheden die het menu aan de gebruiker biedt.
De volgende stap was het opstellen van een klassediagram. Door het ontwerpen van dit diagram zou inzicht kunnen worden verkregen in de statische structuur van het menu. Het klassediagram toont de elementen waaruit het menu bestaat en hun onderlinge relaties en kan als basis dienen voor het verdere ontwerp van het menu tijdens de fase pilotontwikkeling.

Omdat het te ontwikkelen menu gebruik maakt van een reeds bestaande menustructuur, heb ik het klassediagram op deze bestaande structuur gebaseerd. Om dit te kunnen realiseren, heeft de begeleider mij de opbouw van het menu in de database uitgelegd. Zo is het menu bijvoorbeeld in de database opgeslagen in meerdere tabellen. Een record uit de tabel die alle menu-items bevat kan óf een menu-item óf een programma zijn. Daarnaast kan een menu-item een submenu-item van een ander menu-item zijn.

Het klassediagram heb ik ontworpen door de werkwijze te volgen die Warmer en Kleppe beschrijven in “Praktisch UML”. Als eerste heb ik de klassen geselecteerd uit de zelfstandige naamwoorden die in de use-cases voorkomen. Vervolgens heb ik associaties, attributen en operaties gedefinieerd.
Het resultaat is het klassediagram dat hieronder is opgenomen.



Figuur 16: *Klassediagram menu XL-Enz*

Bij het klassediagram heb ik een modeldictionary opgenomen, waarin ik elk element uit het klassediagram heb omschreven, om onduidelijkheid over de gebruikte termen te voorkomen.

6.4.2 Systeemconcept dataobject Crystal Reports

Het bleek vrijwel niet mogelijk om het systeemconcept voor het dataobject voor Crystal Reports te ontwerpen door middel van UML technieken. In de definitiefase wordt alleen een gebruikersgericht concept van het systeem ontwikkeld. Omdat het dataobject voor Crystal Reports geen gebruikersgerichte toepassing is, was het vrijwel niet mogelijk om een systeemconcept met behulp van UML technieken te ontwikkelen.
De enige use-case die ik heb ontworpen, is de onderstaande.

Naam:	Rapport genereren in Crystal Reports
Aannamen:	De gebruiker heeft aangegeven welke informatie het rapport dient te bevatten en heeft opdracht gegeven het rapport te genereren
Beschrijving:	Het systeem zoekt in de database de benodigde gegevens om het rapport te kunnen genereren en geeft deze door aan de gebruiker
Uitzonderingen:	Geen
Resultaat:	Het rapport wordt getoond
Dekt systeemeis:	DF1

Figuur 17: *Use-case "Rapport genereren in Crystal Reports"*

Het bijbehorende use-case-diagram heb ik niet ontwikkeld, omdat dit zou bestaan uit slechts één actor die deelneemt in slechts één use-case.
Om toch een systeemconcept te hebben dat ik tijdens de fase pilotontwikkeling als uitgangspunt kon gebruiken, heb ik besloten een tekstuele omschrijving te geven van het proces dat dient te worden uitgevoerd om een rapport op te stellen.
Deze omschrijving is als volgt:

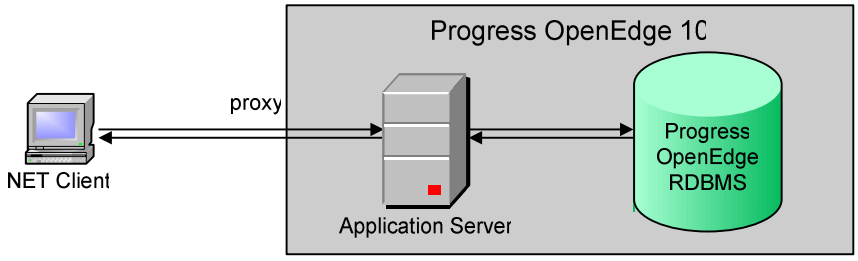
- De gebruiker geeft het systeem opdracht een rapport te genereren;
- De Progress AppServer ontvangt de aanvraag en stelt een query samen;
- De query wordt op de Progress DBMS uitgevoerd;
- De Progress DBMS verzamelt op basis van de query de benodigde data;
- De Progress DBMS verstuurt de data naar de AppServer;
- De AppServer maakt een ProDataSet-object aan dat de ontvangen data bevat;
- Het ProDataSet-object wordt geconverteerd naar een ADO.NET DataSet-object, in de vorm van een ADO.NET XML bestand;
- Het ADO.NET XML bestand wordt naar de client verstuurd;
- Het ADO.NET XML bestand wordt ingelezen in Crystal Reports.

Dit concept heb ik voorgelegd aan de begeleider en deze ging ermee akkoord.

6.5 Technische structuur

De volgende activiteit in de fase definitiestudie was het toetsen of de huidige technische structuur kon worden gebruikt om het ontworpen systeemconcept te realiseren. Dit had tot doel om onvoorziene aanpassingen of uitbreidingen aan de technische structuur te voorkomen en om te definiëren op welke wijze het systeemconcept in de technische structuur kon worden gerealiseerd.
De niet-functionele systeemeisen speelden hierbij een grote rol. Voor dit project zijn namelijk zeer specifieke eisen gesteld aan de technische structuur.
Tijdens het ontwerpen van het systeemconcept had ik reeds rekening gehouden met de gedistribueerde architectuur waar de applicatie XL-Enz gebruik van maakt.

In documentatie van Progress Software Corporation over een koppeling tussen Progress en .NET was beschreven dat deze koppeling te realiseren is door de bestaande architectuur uit te breiden door de client geschikt te maken voor .NET. Deze uitbreiding bestaat uit het installeren van het (gratis) .NET Framework op de client.
De technische architectuur die hierdoor ontstaat, heb ik als volgt schematisch weergegeven:



Figuur 18: *Technische architectuur*

De conclusie was dat het systeemconcept te realiseren was op de bestaande technische architectuur, zij het dat de client geschikt dient te worden gemaakt voor .NET.

6.6 Pilotplan

De laatste activiteit binnen de fase definitiestudie is het opstellen van een pilotplan. Het doel van deze activiteit was het opstellen van een geprioriteerde lijst van pilots, die in de volgende fasen ontwikkeld en ingevoerd gaan worden. Elke pilot biedt na oplevering een zelfstandige, volledig operationele subset van het gehele systeem en kan daardoor als mijlpaal worden gezien.
Als eerste heb ik een pilotstructuur ontworpen. Het menu van XL-Enz en het dataobject voor Crystal Reports zijn los van elkaar staande toepassingen, die apart van elkaar kunnen worden ontwikkeld. Hierdoor lag het voor de hand deze delen van het project te verdelen in twee pilots. Het ontwikkelen van het menu van XL-Enz heb ik vervolgens onderverdeeld in twee pilots:

1. Ontwikkelen “Must Have” functionaliteiten van het menu;
2. Ontwikkelen overige functionaliteiten van het menu (functionaliteiten waaraan een lagere prioriteit is toegekend dan “Must Have”).

Ik heb hiervoor gekozen, omdat na het implementeren van de “Must Have” functionaliteiten het in principe mogelijk is om het menu te gebruiken. Het menu zou eventueel kunnen worden ingevoerd met alleen deze functionaliteiten.
Vervolgens heb ik de pilots geprioriteerd. Omdat de implementatie van het dataobject voor Crystal Reports kennis zou opleveren die zou kunnen worden gebruikt bij het implementeren van het menu, heb ik voorgesteld om het dataobject als eerste te ontwikkelen. De begeleider ging hiermee akkoord.
Na deze pilot heb ik de pilot gepland waarin de “Must Have” functionaliteiten van het menu worden ontwikkeld. Als laatste is de pilot gepland waarin de functionaliteiten met een lagere prioriteit worden ontwikkeld.

Vervolgens heb ik per pilot pilotdelen gedefinieerd. Hierbij heb ik gecontroleerd of elke systeemeis wordt gedekt door minstens één van de pilotdelen. Zo ziet de inhoud van de eerste pilot er bijvoorbeeld als volgt uit.

Pilot 1: Dataobject Crystal Reports		
Pilotdeel	**Omschrijving**	**Dekt systeemeis**
1.1	Ontwikkelen dataobject in Progress OpenEdge 10	DN1
1.2	Converteren Progress dataobject naar .NET dataobject	DN2, DN3
1.3	Invoegen dataobject in Crystal Reports	DF1, DN4

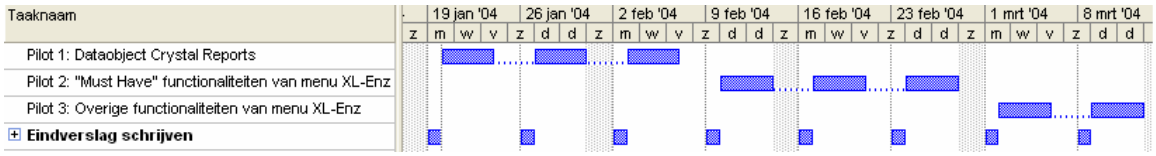
Figuur 19: *Pilotdelen van pilot 1*

Na het definiëren van de pilotdelen, heb ik een schatting gemaakt van de tijd die elke pilot in beslag zal nemen. Voor de fase pilotontwikkeling heb ik totaal 32 dagen gepland.

Pilot	**Geschat percentage**	**Aantal dagen**
Pilot 1: Dataobject Crystal Report	40 %	12
Pilot 2: “Must Have” functionaliteiten van menu XL-Enz	40 %	12
Pilot 3: Overige functionaliteiten van menu XL-Enz	20 %	8
Totaal:	**100 %**	**32**

Figuur 20: *Geschat aantal dagen per pilot*

Op basis van bovenstaande schatting heb ik de onderstaande planning gemaakt van de fase pilotontwikkeling. Hierbij heb ik vermeld dat per pilot een gedetailleerde planning wordt gemaakt voor aanvang van de pilot.



Figuur 21: *Globale planning fase pilotontwikkeling*

Vervolgens heb ik een globaal pilotacceptatieplan opgesteld, waarin ik heb vermeld dat een pilot na elke iteratieslag wordt getest op de wijze die is beschreven in het hoofdstuk “Testaanpak en kwaliteitscontrole” uit het plan van aanpak voor het gehele project.
Als laatste heb ik een globaal invoeringsplan opgesteld, waarin ik de wijze van invoering van de pilots heb beschreven.

Nadat alle pilots voldoen aan de gestelde eisen en normen, kan worden overgegaan tot invoering van de pilots.
Pilot 1: Dataobject Crystal Reports kan worden ingevoerd door het huidige dataobject te vervangen door het .NET dataobject en door de client geschikt te maken om met een .NET omgeving te kunnen werken.
Het menu van XL-Enz kan worden ingevoerd indien het menu zoveel functionaliteiten bevat dat het zinvol is om het menu in te voeren. Deze beslissing wordt genomen door de begeleider en/of de opdrachtgever.
Invoering van het menu bestaat uit het vervangen van het huidige menu met het nieuwe menu. Daarnaast dient de client geschikt te worden gemaakt om met een .NET omgeving te kunnen werken.
Na invoering vindt een acceptatietest plaats, waarbij de pilots aan de hand van de gestelde eisen en normen getest worden op hun werking. Deze test wordt uitgevoerd door de begeleider en/of de opdrachtgever.

6.7 Afronding

Na het schrijven van een samenvatting van de inhoud van het rapport definitiestudie en het controleren van de rapportopmaak, heb ik het rapport opgeleverd. De begeleider heeft het rapport vervolgens doorgenomen en is akkoord gegaan met de inhoud ervan. Het document definitiestudie is als externe bijlage opgenomen bij dit verslag.
Voor het doorlopen van de fase definitiestudie heb ik enkele dagen minder nodig gehad dan de geplande twee weken. Doordat ik echter drie dagen ziek ben geweest, is de tijd die ik had gewonnen weer verloren gegaan.

7. Pilotontwikkeling pilot 1

7.1 Inleiding

Na oplevering en goedkeuring van het rapport definitiestudie is de fase pilotontwikkeling gestart. Deze fase volgt in het IAD systeemontwikkeltraject op de fase definitiestudie en heeft tot doel om de pilots die tijdens de definitiestudie zijn gedefinieerd, gedetailleerd te ontwerpen in een pilotontwikkelrapport en vervolgens te implementeren en te testen. Tijdens de definitiestudie is het ontwikkelen van het .NET dataobject voor Crystal Reports gedefinieerd als pilot die het eerst wordt ontwikkeld. Dit hoofdstuk beschrijft de activiteiten die zijn uitgevoerd tijdens de ontwikkeling van deze pilot.
De opbouw van dit hoofdstuk is als volgt. Als eerste wordt in paragraaf 7.2 de totstandkoming van het plan van aanpak voor de ontwikkeling van de pilot beschreven. In de paragrafen 7.3 en 7.4 beschrijf ik respectievelijk het ontwerpen van de functionele en de technische structuur van de pilot. Vervolgens wordt in paragraaf 7.5 de totstandkoming van het pilotontwikkelplan beschreven, waarin software-bouweenheden zijn gedefinieerd. Het ontwerp van deze bouweenheden wordt besproken in paragraaf 7.6. In paragraaf 7.7 wordt het bouwen van de bouweenheden beschreven. Vervolgens wordt in paragraaf 7.8 de totstandkoming van de invoeringsprocedure van de pilot beschreven. Paragraaf 7.9 behandelt het proces van integratie van de bouweenheden en paragraaf 7.10 beschrijft hoe de pilotdelen zijn beoordeeld en getest. Als laatste wordt in paragraaf 7.11 de afronding van de fase pilotontwikkeling van deze pilot beschreven.

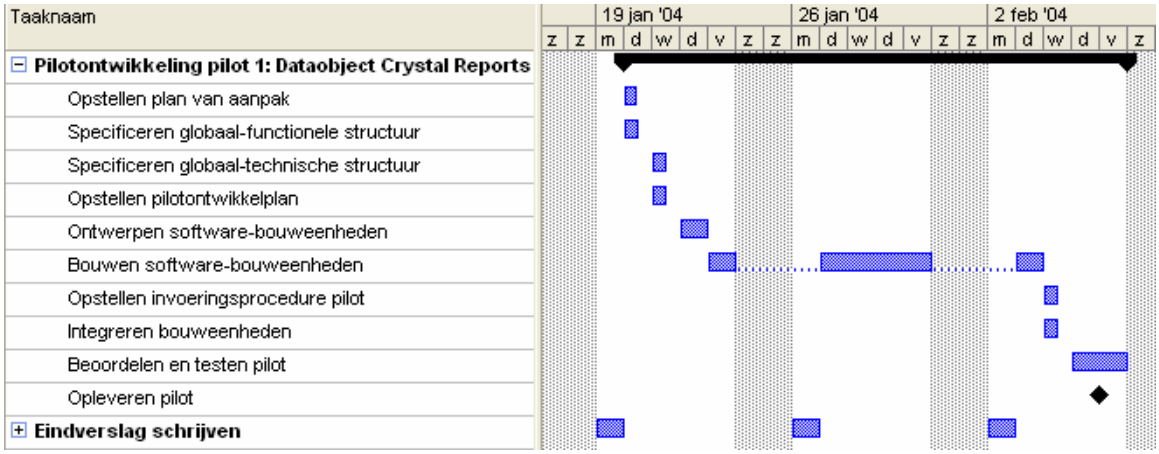
7.2 Plan van aanpak

IAD schrijft als eerste activiteit binnen de fase pilotontwikkeling voor om een plan van aanpak op te stellen voor de te ontwikkelen pilot. Na bestudering van de activiteiten die tijdens de pilotontwikkeling kunnen worden uitgevoerd, heb ik in het plan van aanpak een overzicht gemaakt van de uit te voeren activiteiten. Voor de activiteiten die mij niet relevant leken voor dit project, heb ik een reden gegeven waarom ik deze niet relevant vond.

De volgende activiteiten die IAD voorschrijft worden tijdens dit project niet uitgevoerd:

- Voorbereiden pilotontwerp-workshop
  Reden: De pilot is te klein om een workshop zinvol te laten zijn.
- Specificeren globaal-organisatorische inrichting
  Reden: De pilot heeft geen invloed op organisatorische aspecten.
- Aanpassen externe componenten
  Reden: Er is geen noodzaak tot aanpassing van externe componenten.
- Wijzigen andere informatiesystemen
  Reden: Er is geen noodzaak tot aanpassing van andere informatiesystemen.
- Bouwen conversie-tools
  Reden: Er zijn geen conversie-tools benodigd voor deze pilot.
- Ontwerpen handmatige procedures
  Reden: Er is geen sprake van handmatige procedures bij deze pilot.
- Samenstellen opleidingsmateriaal pilot
  Reden: Er is geen behoefte aan opleidingsmateriaal voor deze pilot.
- Samenstellen handleiding pilot
  Reden: Deze activiteit is gepland tijdens de fase invoering

Van de relevante activiteiten heb ik op basis van de beschrijving uit het boek van Tolido een inschatting gemaakt van de benodigde tijd. Hierop is de onderstaande planning gebaseerd.

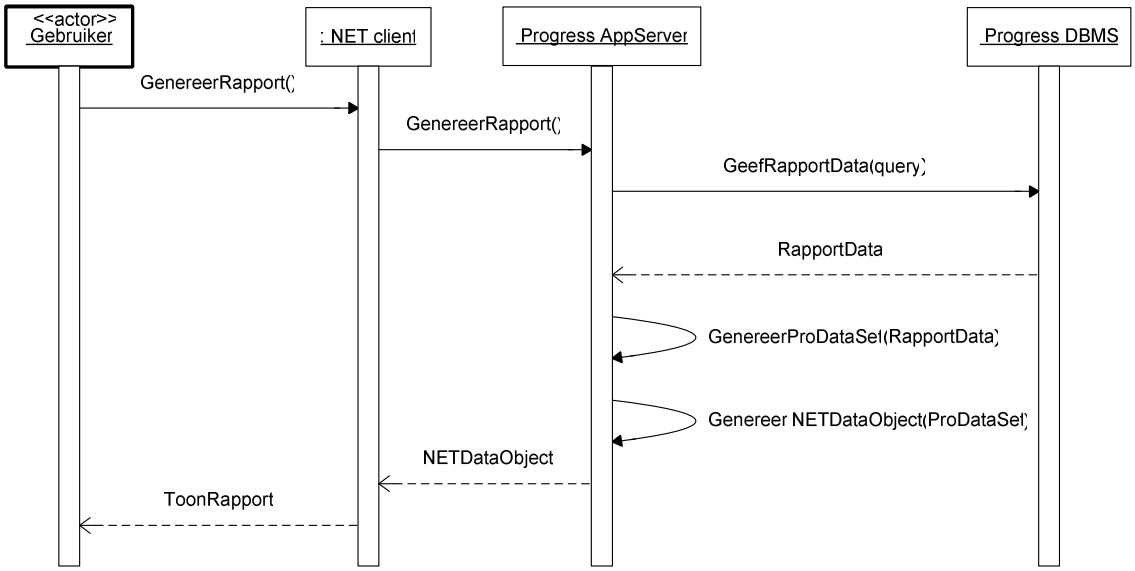


Figuur 22: *Faseplanning ontwikkeling pilot 1: Dataobject Crystal Reports*

Als laatste heb ik de iteratiestrategie van de pilot beschreven. Tijdens de activiteit “Beoordelen en testen pilot” wordt beoordeeld of de pilot voldoet aan de eisen die zijn opgesteld tijdens de definitiestudie. Wanneer dit niet het geval is, dient in overleg met de begeleider en/of opdrachtgever te worden besloten of de fase pilotontwikkeling voor deze pilot nogmaals wordt doorlopen.
Na overeenstemming met de begeleider te hebben bereikt over dit plan van aanpak ben ik begonnen met het ontwerpen van de pilot door het opstellen van een pilotontwikkelrapport.

7.3 Functionele structuur

Als eerste heb ik de functionele structuur van de pilot ontworpen. Het doel van deze activiteit was om de functionele inhoud van de pilot te bepalen. Tijdens de fase definitiestudie had ik reeds geconcludeerd dat deze pilot alleen gevolgen heeft voor de opbouw van het dataobject voor Crystal Reports en niet voor de functionaliteiten voor de gebruiker. In het rapport definitiestudie had ik één use-case opgesteld, die de functionele inhoud van de pilot beschrijft. Ik heb onderzocht hoe ik deze use-case zodanig verder kon uitwerken met behulp van UML diagrammen, dat daarmee duidelijk zou worden hoe het interne proces van het opbouwen van een rapport verloopt.
Volgens het boek “Praktisch UML” (J. Warmer en A. Kleppe) bleken UML sequencediagrammen hiervoor geschikt. Dit diagram heb ik dan ook toegepast op basis van de use-case “Rapport genereren in Crystal Reports”. Het sequencediagram op de volgende pagina is ontstaan door de werkwijze te volgen die in het genoemde boek wordt beschreven.



Figuur 23: *Sequencediagram "Genereren rapport"*

Het diagram geeft een overzicht van de interactie tussen de objecten die benodigd is om een rapport te genereren dat wordt opgebouwd op basis van een .NET dataobject. Met het ontwerpen van dit diagram leek de functionele structuur van deze pilot voldoende duidelijk. De overige activiteiten die IAD voorschrijft om de functionele structuur te ontwerpen, leken mij dan ook niet relevant. Zo is bijvoorbeeld de activiteit "Maak prototype gebruikersinterface" niet relevant, omdat bij deze pilot geen sprake is van een grafische gebruikersinterface.

7.4 Technische structuur

Op basis van het systeemconcept uit de definitiestudie en het ontwerp van de functionele structuur heb ik vervolgens de technische structuur van de pilot ontworpen, met het doel om de technische aspecten van de pilot te specificeren. Hierbij heb ik de verantwoordelijkheden en taken van de objecten uit het sequencediagram beschreven. Zo heb ik bijvoorbeeld de verantwoordelijkheden van de Progress AppServer als volgt omschreven.

- Progress AppServer
  De AppServer bevat alle business logica van de applicatie XL-Enz. Op dit moment bestaat er reeds een functie die de aanvraag voor het samenstellen van een rapport afhandelt. Deze functie stelt op basis van de aanvraag een query samen, die wordt uitgevoerd op de Progress DBMS.
  - Er dient een functie te worden geschreven die het resultaat van de uitgevoerde query in een Progress ProDataSet-object opneemt;
  - Er dient een functie te worden geschreven die het ProDataSet-object converteert naar een ADO.NET dataobject.

Vanwege het feit dat deze pilot wordt geïmplementeerd op een reeds bestaande netwerkarchitectuur, heb ik het modelleren van het netwerkcommunicatiemodel achterwege gelaten.
Als laatste heb ik onderzocht welke technische aanpassingen of uitbreidingen benodigd zijn om een .NET dataobject als datasource voor Crystal Reports te kunnen gebruiken. Deze aanpassingen bleken zich te beperken tot het aanpassen van de configuratie van het type datasource dat wordt gebruikt door Crystal Reports.

7.5 Pilotontwikkelplan

De volgende activiteit was het opstellen van een pilotontwikkelplan op basis van de opgestelde functionele en technische specificaties. Een pilotontwikkelplan beschrijft hoe de pilot gaat worden gebouwd. Het opstellen van het plan heb ik uitgevoerd volgens de werkwijze die wordt voorgesteld in het boek “IAD – Het evolutionair ontwikkelen van informatiesystemen” door R.J.H. Tolido.
Het pilotontwikkelplan dat ik tijdens de definitiestudie had opgesteld voor deze pilot, heb ik hierbij als uitgangspunt gebruikt. Voor elk pilotdeel uit dit plan heb ik bouweenheden gedefinieerd, die afzonderlijk konden worden gebouwd. De volgende bouweenheden heb ik gedefinieerd:

1. Opzetten ontwikkelomgeving;
2. Bouwen functie voor het opnemen van de rapportdata in een Progress ProDataSet-object;
3. Converteren ProDataSet-object naar ADO.NET dataobject;
4. Invoegen dataobject in Crystal Reports.

Enkele pilotdelen uit het pilotontwikkelplan dat ik tijdens de definitiestudie had opgesteld, bleken niet op te delen in meerdere bouweenheden. Bijvoorbeeld het pilotdeel “Invoegen dataobject in Crystal Reports” omvatte reeds zo’n specifieke activiteit, dat ik het niet noodzakelijk vond om hiervoor deelactiviteiten te specificeren. Dit pilotdeel heb ik dan ook als één bouweenheid gedefinieerd.
Vanwege het feit dat ik het ontwikkelen van de pilot alleen zou uitvoeren, was het niet mogelijk enkele bouweenheden parallel te ontwikkelen. Daarom heb ik in het pilotplan opgenomen dat alle bouweenheden één voor één worden ontwikkeld in de volgorde zoals hierboven is aangegeven.
Na het definiëren van de bouweenheden heb ik de iteratiestrategie voor het ontwikkelen van de bouweenheden beschreven. Om de ontwikkelsnelheid te verhogen, zijn namelijk in overleg met de begeleider de volgende afspraken gemaakt.

Tijdens de eerste iteratie wordt deze pilot op de volgende wijze geïmplementeerd:

- De implementatie vindt plaats in een lokale omgeving.
  Deze keuze komt voort uit het feit dat het gebruik van nieuwe technieken (.NET, ProDataSet-object) onverwachte nadelige gevolgen kan hebben op de bestaande hard- en software. Door eerst de pilot in een lokale omgeving te implementeren, kunnen deze gevolgen worden onderzocht, zonder dat deze verdere gevolgen hebben voor de hard- en software die wordt gebruikt voor de ontwikkeling van de applicatie XL-Enz;
- Het Progress dataobject (ProDataSet-object) wordt als een statisch opgebouwd object geïmplementeerd.
  Hierdoor wordt de doorlooptijd van de eerste iteratie verkort, waardoor het mogelijk wordt eerder de pilot te testen (implementatie van een dynamisch opgebouwd dataobject zou namelijk meer tijd kosten dan implementatie van een statisch opgebouwd dataobject).

Vooral de beslissing om de pilot tijdens de eerste iteratie te implementeren op basis van een statisch opgebouwd ProDataSet-object zou de ontwikkelsnelheid sterk kunnen verhogen. Zoals ik reeds had ervaren tijdens de activiteit “Kennis vergaren van Progress” is het implementeren van een dynamisch opgebouwd dataobject meer complex en tijdrovender dan het implementeren van een statisch opgebouwd dataobject. Wanneer de pilot eenmaal is geïmplementeerd op basis van een statisch opgebouwd ProDataSet-object, kan deze worden getest. Wanneer uit deze tests blijkt dat een .NET datasource inderdaad sterke verbeteringen oplevert ten opzichte van de huidige datasource, kan worden besloten om in een volgende iteratieslag het ProDataSet-object dynamisch op te bouwen.

Als laatste heb ik in het pilotontwikkelplan de teststrategie voor de pilot beschreven. Hierin heb ik beschreven dat elke bouweenheid na implementatie wordt getest op de wijze en op de onderdelen zoals dat beschreven is in het hoofdstuk “Testaanpak en kwaliteitscontrole” van het plan van aanpak voor het gehele project.

7.6 Ontwerp software-bouweenheden

Na het definiëren van de bouweenheden was de volgende activiteit het ontwerpen van deze bouweenheden. Dit ontwerp dient zodanig te zijn opgesteld, dat het mogelijk moet zijn om op basis hiervan de bouweenheden te implementeren.

7.6.1 Ontwerpen bouweenheden

Het ontwerp van de bouweenheden heb ik gedurende de fase pilotontwikkeling steeds verder aangevuld naar aanleiding van nieuw opgedane kennis en inzicht. Het eerste ontwerp van de bouweenheden heb ik gemaakt op basis van de kennis en het inzicht dat ik had na het uitvoeren van het onderzoek naar interfacing. Het stappenplan dat ik had opgesteld voor de ontwikkeling van het dataobject (zie paragraaf 5.4.6) heb ik daarbij als uitgangspunt gebruikt. Dit resulteerde in een ontwerp dat bestond uit een tekstuele beschrijving van de te ontwikkelen bouweenheden. Het ontwerp voor bouweenheid 3 “Converteren ProDataSet-object naar ADO.NET dataobject” zag er bijvoorbeeld als volgt uit.

> Wanneer het (statisch opgebouwde) ProDataSet-object is geïmplementeerd en getest, kan deze worden geconverteerd naar een ADO.NET dataobject.
> Dit dataobject bestaat uit een ADO.NET DataSet, die wordt opgeslagen in de vorm van een ADO.NET XML bestand.
> Progress biedt functionaliteiten om een XML bestand te genereren. Tijdens de implementatie van deze bouweenheid wordt een functie geschreven die gebruik maakt van de genoemde functionaliteiten van Progress. In deze functie wordt de (meta)data uit het eerder gebouwde ProDataSet-object geconverteerd naar een XML bestand dat voldoet aan de eisen van een ADO.NET XML bestand. Deze functie dient een generiek karakter te hebben: elke ProDataSet dient te kunnen worden geconverteerd naar een ADO.NET XML bestand.

Vervolgens heb ik gezocht naar mogelijkheden om UML diagrammen toe te passen in het ontwerp om hiermee het tekstuele ontwerp aan te vullen. De diagrammen leken echter geen toegevoegde waarde te bieden aan het reeds bestaande ontwerp. Dit vanwege het feit dat vrijwel alle diagrammen zich richten op het functionele deel van een systeem (het gedrag). Dit deel was echter reeds voldoende ontworpen en gemodelleerd door middel van een sequencediagram.

Omdat ik na het maken van het tekstuele ontwerp wel duidelijke specificaties had van de te ontwikkelen bouweenheden, heb ik besloten om de bouw van de bouweenheden te starten. Wanneer ik aanvullende relevante kennis zou opdoen tijdens het bouwen, kon ik alsnog het ontwerp uitbreiden met deze kennis.
Deze aanpak wordt ook ondersteund door IAD. Tolido schrijft hierover in het eerder genoemde boek: “Er is een nauwe relatie tussen het ontwerpen en het bouwen van een software-bouweenheid. Binnen de context van een hecht, snel werkend A-team zullen de activiteiten met betrekking tot ontwerpen, bouwen en testen zeer regelmatig worden herhaald, niet noodzakelijkerwijs in de beschreven volgorde.”

Tijdens het bouwen van de bouweenheden heb ik inderdaad aanvullende relevante kennis opgedaan die ik had kunnen opnemen in het ontwerp van de bouweenheden. Ik heb deze kennis echter niet in het pilotontwikkelrapport opgenomen, maar in het rapport over .NET interfacing. Het stappenplan wat ik daarin had opgesteld, heb ik met deze kennis uitgebreid zoals ik heb beschreven in paragraaf 5.4.6. Ik heb hiervoor gekozen, omdat het rapport over interfacing tot doel had om de ontwikkelaars van Reflecta kennis op te laten doen van de wijze van implementatie van interfacing. Door de kennis die ik heb opgedaan tijdens de bouw van de bouweenheden te verwerken in dit rapport, kon ik aan deze doelstelling voldoen.

In het pilotontwikkelrapport heb ik bij het tekstuele ontwerp voor bouweenheid 3 de volgende verwijzing opgenomen naar het rapport over interfacing.

> Voor gedetailleerde informatie over de eisen aan de opbouw van een ADO.NET XML bestand en over de wijze van conversie van een ProDataSet-object naar een ADO.NET XML bestand wordt verwezen naar hoofdstuk 5 van het rapport “Interfacing tussen Progress en Microsoft .NET”.

7.6.2 Opstellen testspecificaties

Bij het ontwerp van de bouweenheden heb ik (zoals voorgeschreven door IAD) per bouweenheid testspecificaties opgenomen. Deze testgevallen dienen te worden uitgevoerd na implementatie van een bouweenheid. In het plan van aanpak voor het gehele project is vastgesteld dat de teststrategie “black box testing” wordt gehanteerd. Daarom heb ik voor elke bouweenheid testgevallen opgesteld, waarbij elke mogelijke waarde van invoer met daarbij de verwachte uitvoer wordt gedefinieerd.
Deze testspecificaties heb ik deels (evenals het ontwerp van de bouweenheden) gedurende het bouwen van de bouweenheden opgesteld. Tijdens de bouw van de bouweenheden kreeg ik meer inzicht in de werking van de bouweenheden. De bijzondere gevallen die ik daarbij tegenkwam, heb ik verwerkt in de testspecificaties.
Op de volgende pagina zijn ter illustratie de testspecificaties opgenomen van bouweenheid 3: “Converteren ProDataSet-object naar ADO.NET dataobject”.

Bouweenheid 3 “Converteren ProDataSet-object naar ADO.NET dataobject” levert een ADO.NET XML bestand op. De opbouw van dit bestand dient aan eisen te voldoen om correct te kunnen functioneren als datasource voor Crystal Reports.
Door middel van het uitvoeren van onderstaande testcases kan worden getest of het ADO.NET XML bestand aan deze eisen voldoet. Om een overzichtelijk beeld te verkrijgen van de opbouw van het ADO.NET XML bestand, kan deze worden ingelezen in Visual Studio .NET 2003.

De onderstaande tests dienen te worden uitgevoerd op een ProDataSet-object dat aan de volgende kenmerken voldoet:

- Het object bevat 2 TempTables: “ttCustomer” en “ttOrder”. Deze tabellen zijn gebaseerd op de tabellen “Customer” en “Order” uit de Sports2000 database;
- Er is een relatie gedefinieerd tussen deze tabellen, op basis van het veld “custnum”;

Testspecificaties bouweenheid 3: Converteren ProDataSet-object naar ADO.NET dataobject			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
3.1	ADO.NET XML bestand bestaat	Controleren of ADO.NET XML bestand in gedefinieerde output directory staat	ADO.NET XML bestand bestaat
3.2	ADO.NET XML bestand bevat een syntactisch correcte ADO.NET DataSet (inclusief XSD schema)	ADO.NET XML bestand openen in Visual Studio .NET 2003	Overzicht van de tabellen en de data die het ADO.NET XML bestand bevat
3.3	Schema in ADO.NET XML bestand bevat correcte definitie van de juiste tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Schema in het ADO.NET XML bestand met gelijke definities van de tabellen “ttCustomer” en “ttOrder” als de definities van deze tabellen in Progress database
3.4	ADO.NET XML bestand bevat juiste data	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Data in het ADO.NET XML bestand met gelijke waarden als de waarden van de tabellen “ttCustomer” en “ttOrder” in Progress database
3.5	ADO.NET XML bestand bevat een correcte definitie van de juiste relaties tussen de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de relatie tussen de tabellen “ttCustomer” en “ttOrder”, op basis van het veld “custnum”
3.6	ADO.NET XML bestand bevat een correcte definitie van de juiste primaire sleutels van de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de primaire sleutels “Custnum” voor “ttCustomer” en “Ordernum” voor “ttOrder”

Figuur 24: *Testspecificaties bouweenheid 3*

7.7 Bouw software-bouweenheden

Na het maken van het eerste ontwerp van de bouweenheden ben ik begonnen met de implementatie van deze bouweenheden. Deze paragraaf beschrijft het implementeren en testen van de bouweenheden.

7.7.1 Opzetten ontwikkelomgeving

In de eerste bouweenheid heb ik de ontwikkelomgeving opgezet. De pilot werd in de eerste iteratie in een lokale omgeving geïmplementeerd. Dit hield in dat de volledige omgeving lokaal moest worden geïnstalleerd. In het pilotontwikkelrapport heb ik een overzicht opgenomen van de software die benodigd is om de pilot te kunnen implementeren. In samenwerking met de begeleider van het project heb ik deze software geïnstalleerd en vervolgens getest volgens de testspecificaties uit het pilotontwikkelrapport. De testgevallen zijn succesvol doorlopen.
De implementatie van deze bouweenheid heeft ongeveer één dag in beslag genomen en heeft geen problemen opgeleverd.

7.7.2 Bouwen functie opnemen rapportdata in ProDataSet-object

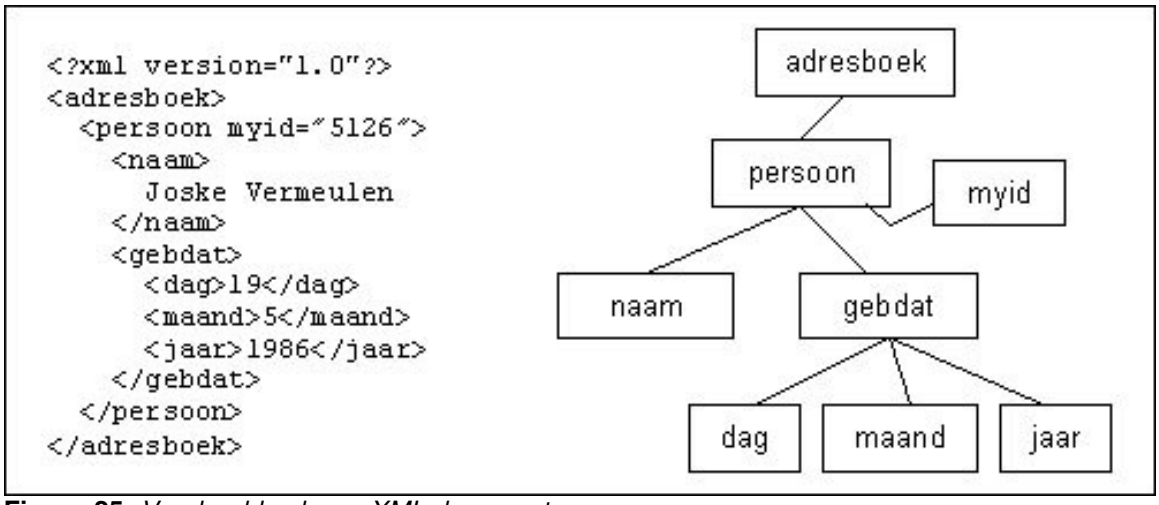
Na de ontwikkelomgeving succesvol te hebben opgezet en getest, ben ik begonnen met de implementatie van bouweenheid 2. Deze bouweenheid omvatte in de eerste iteratieslag de implementatie van een statisch opgebouwd Progress ProDataSet-object, dat willekeurige gegevens uit een Progress database bevatte.
Bij de bouw van deze bouweenheid heb ik gebruik gemaakt van de kennis van Progress die ik heb opgedaan tijdens de activiteit “Kennis vergaren van Progress”.
Hierbij heb ik een ProDataSet-object geïmplementeerd, dat alle gegevens uit de tabellen “Customer” en “Order” van de Sports2000 database bevatte (deze database wordt standaard meegeleverd bij Progress OpenEdge 10 als testdatabase). Ik heb het ProDataSet-object gevuld op basis van de relatie die tussen de tabellen was gedefinieerd.
Dit ProDataSet-object heb ik als uitgangspunt genomen voor het opstellen van de testspecificaties.
Nadat het ProDataSet-object correct leek te functioneren, heb ik deze bouweenheid getest door de opgestelde testgevallen uit te voeren. Deze testgevallen zijn allen succesvol doorlopen.
Vervolgens heb ik de wijze van implementatie van dit ProDataSet-object beschreven in het verslag over interfacing. Hiermee had ik stap 1 uit het stappenplan voor implementatie van een .NET dataobject voor Crystal Reports uitgewerkt (zie paragraaf 5.4.6).
De implementatie van deze bouweenheid heeft ruim één dag in beslag genomen en heeft geen problemen opgeleverd.

7.7.3 Converteren ProDataSet-object naar ADO.NET dataobject

Nadat het ProDataSet-object was geïmplementeerd en succesvol was getest, kon bouweenheid 3 worden ontwikkeld. Deze bouweenheid omvatte de conversie van het ProDataSet-object naar een ADO.NET dataobject in de vorm van een ADO.NET XML document.

Bij aanvang van de bouw van deze bouweenheid had ik onvoldoende kennis om deze bouweenheid gelijk te kunnen implementeren. De ontwikkelaars van Reflecta hadden aangegeven dat het mogelijk was om vanuit Progress output te genereren naar een XML document. De datasource die reeds werd gebruikt voor Crystal Reports bestond namelijk ook uit een XML bestand dat was opgebouwd vanuit Progress. Ik had echter geen ervaring met het werken met XML en ook niet met het genereren van XML documenten met behulp van Progress. De enige kennis die ik had van XML, heb ik tijdens het onderzoek naar interfacing opgedaan (zie paragraaf 5.4.2). Ik wist dat XML voorziet in een gestandaardiseerde wijze van opslag van gestructureerde data.

Om deze bouweenheid toch te kunnen implementeren, heb ik als eerste onderzocht hoe XML documenten zijn opgebouwd. Als informatiebron heb ik het Internet gebruikt, waar talloze voorbeelden en handleidingen te vinden waren over XML. XML documenten bleken als een boom te worden opgebouwd door middel van “tags”, die de semantiek van de data beschrijven. Binnen de “tags” wordt de data geplaatst. Hieronder is een voorbeeld opgenomen van een XML document, waarbij de boomstructuur duidelijk zichtbaar is.



Figuur 25: *Voorbeeld opbouw XML document*

Omdat deze XML documenten als een boom zijn opgebouwd, is het niet mogelijk om in dergelijke bestanden meerdere tabellen op te nemen. Dergelijke documenten bevatten namelijk altijd maar precies één root-element (“adresboek” in het voorbeeld). De datasource van Crystal Reports die reeds wordt gebruikt, bestaat uit een XML document met bovenstaande opbouw.

Na het onderzoek naar interfacing tussen Progress, .NET en Crystal Reports was bekend geworden dat Crystal Reports onderscheid maakt tussen datasources die bestaan uit XML documenten die zijn opgebouwd zoals hierboven is beschreven en tussen zogenaamde ADO.NET XML documenten. Het was echter onbekend hoe ADO.NET XML documenten waren opgebouwd en wat het verschil met “traditionele” XML documenten was. Ik heb hiernaar onderzoek verricht door wederom op Internet te zoeken.
In een ADO.NET XML document bleek een ADO.NET DataSet te zijn opgeslagen. Een ADO.NET DataSet is een dataobject in .NET dat vergelijkbaar is met de ProDataSet in Progress. Het is mogelijk om vanuit een .NET omgeving met behulp van één statement

(WriteToXML()) een DataSet te exporteren naar een ADO.NET XML document. Dit document kan Crystal Reports gebruiken als datasource voor rapporten.

Het verschil tussen “traditionele” XML documenten en ADO.NET XML documenten bleek te zijn dat ADO.NET XML documenten meerdere tabellen kon bevatten, met daarbij definities van deze tabellen en hun onderlinge relaties. ADO.NET XML documenten bleken te zijn opgebouwd uit twee delen: een document waarin de definitie van de data is opgenomen: het XML Schema Definition (XSD) document en een document waarin de data zelf is opgenomen: het XML document. In het XML document is een verwijzing opgenomen naar het XSD document.
Tijdens tests met het inlezen van XSD en XML voorbeeldbestanden als datasource voor Crystal Reports, bleek dat Crystal Reports de definities uit het XSD document niet herkende. De datatypen die bijvoorbeeld in het XSD document waren opgenomen, werden niet correct weergegeven in Crystal Reports: alle velden hadden als datatype “string”.

Zoals beschreven in paragraaf 5.4.6 stelde de begeleider toen voor om een ADO.NET XML document te genereren waarbij de definitie (XSD) en de data (XML) in één XML bestand was opgenomen. Na een geslaagde test om een dergelijk document als datasource te gebruiken in Crystal Reports, heb ik aan de hand van voorbeeld XSD en XML documenten een functie geschreven in Progress die de definitie en data van de eerder gebouwde ProDataSet in één ADO.NET XML document opslaat.
De ontwikkelaars van Reflecta hadden reeds een functie geschreven waarmee een XML document werd aangemaakt. Deze functie heb ik als uitgangspunt gebruikt. Door deze functie en de documentatie van Progress te bestuderen, heb ik mij de statements aangeleerd die binnen Progress worden gebruikt om XML documenten te genereren.

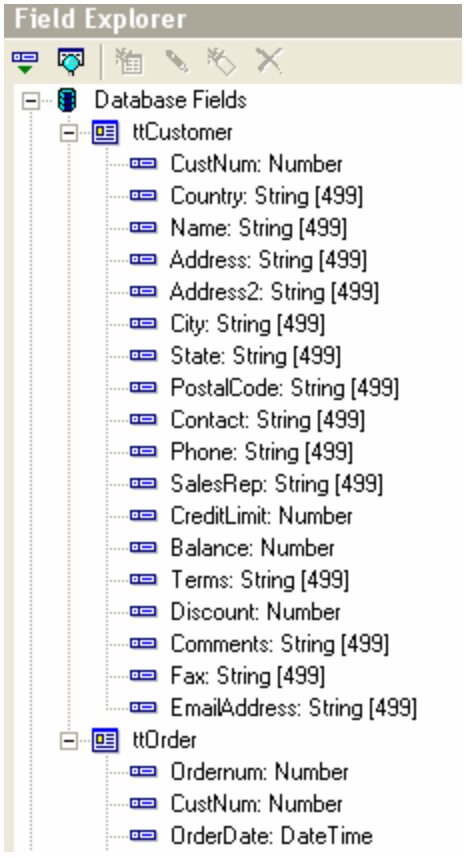
Na enkele dagen had ik een functie gebouwd waarmee een ADO.NET XML document werd aangemaakt dat de definitie en de data bevatte van het eerder gemaakte ProDataSet-object. Het is mogelijk om in Visual Studio .NET 2003 XSD en XML documenten in te lezen en deze te laten controleren op syntactische correctheid. Dit bleek erg waardevol, omdat ik nog onbekend was met de syntax van dergelijke documenten.
Tijdens het bouwen van deze functie liep ik bijvoorbeeld tegen de volgende problemen en vragen aan:

- Incorrecte conversie van datatypen.
  In Progress heeft data van het datatype “boolean” de waarde “true” of “false”. In .NET werden deze waarden niet herkend. De oorzaak hiervoor was dat deze data de waarde “1” of “0” moet hebben. Dit probleem heb ik opgelost door in de functie een conversie op te nemen, waarbij de waarden “true” als “1” in het XML document werden opgeslagen en de waarden “false” als “0”.
- Is het mogelijk om in een XSD bestand maximale lengte van strings op te geven?
  In de Progress database zijn strings opgeslagen met daarbij een maximale lengte van de string. Na onderzoek bleek het mogelijk om in het XSD document een uitbreiding op het standaard datatype “string” op te nemen, waarbij o.a. de lengte van de string kon worden gedefinieerd.
- Hoe kunnen de sleutelvelden en relaties tussen de tabellen uit het ProDataSet-object in het ADO.NET XML document worden opgenomen?
  Bestudering van de documentatie van Visual Studio .NET 2003 leverde een beschrijving van de juiste syntax op om deze elementen in het ADO.NET XML document op te nemen.

Nadat de functie correct leek te werken en een syntactisch correct ADO.NET XML document leek op te leveren, heb ik deze bouweenheid getest door de testgevallen uit te voeren die ik had opgesteld in het pilotontwikkelrapport. Deze testgevallen zijn allen succesvol doorlopen.
Vervolgens heb ik het rapport over interfacing uitgebreid (zie paragraaf 5.4.6) met de kennis die ik heb opgedaan tijdens het bouwen van deze bouweenheid. Hiermee had ik stap 2 uit het stappenplan voor implementatie van een .NET dataobject voor Crystal Reports uitgewerkt.

7.7.4 Invoegen dataobject in Crystal Reports

Nadat het ADO.NET XML document was gegenereerd en getest in bouweenheid 3, kon de laatste bouweenheid worden geïmplementeerd. Deze bestond uit het inlezen van het ADO.NET XML bestand als datasource in Crystal Reports.
Tijdens eerdere tests met het inlezen van dergelijke bestanden had ik reeds ervaring opgedaan met het inlezen van een ADO.NET XML document. Omdat het document in de vorige bouweenheid reeds was getest op syntactische correctheid, vormde het inlezen van het document dan ook geen probleem. De tabellen uit het document leken correct te worden weergegeven.



In het figuur hiernaast is de “Field Explorer” opgenomen van Crystal Reports nadat het ADO.NET XML document is ingelezen. In de “Field Explorer” zijn de tabellen “ttCustomer” en “ttOrder” te zien, die in het ADO.NET XML bestand waren opgeslagen. Vanuit de “Field Explorer” kunnen velden uit de tabellen op het rapport worden geplaatst.
De volgende stap was het testen of de definitie van de data en de data zelf correct werden weergegeven in Crystal Reports. Daarnaast diende getest te worden of aan de performance-eis werd voldaan dat een rapport op basis van een ADO.NET XML datasource minstens even snel werd opgebouwd als op basis van de huidige datasource.
Tijdens het uitvoeren van de hiervoor opgestelde testgevallen bleek dat niet alle tests succesvol werden doorlopen. Enkele tests op de definitie en de weergave van de datatypen faalden. De resultaten van deze tests heb ik in het pilotontwikkelrapport verwoord zoals is te lezen op de volgende pagina.

Figuur 26: *Field Explorer in Crystal Reports*

Mislukte tests:

- Testcase 4.5: Controle op datatype “Decimal” of “Float”
  Het aantal decimalen achter de komma wordt niet correct weergegeven. In het schema van de ADO.NET XML datasource is door middel van het attribuut “fractionDigits” het aantal decimalen achter de komma gedefinieerd. Wanneer de waarden echter in Crystal Reports worden ingelezen, wordt de waarde van dit attribuut genegeerd en wordt hiervoor de waarde gebruikt die is ingesteld in Crystal Reports (deze waarde is standaard “2”).
  Op dit moment is hiervoor nog geen afdoende oplossing voor gevonden.

- Testcase 4.6: Controle op datatype “Date”
  Wanneer in de ADO.NET XML datasource het datatype van een veld “Date” is, wordt deze in Crystal Reports ingelezen als datatype “DateTime”. Dit datatype bestaat uit de elementen “datum” en “tijd”. Dit is niet gewenst, omdat de datums uit het XML bestand meestal geen tijdselement bevatten.
  Dit probleem kan worden opgelost door de weergave van het datatype DateTime in Crystal Reports zodanig aan te passen, dat alleen het datumelement wordt weergegeven. (Zie hiervoor de eigenschappen voor de weergave van het datatype DateTime in Crystal Reports: File – Options – Fields – Date and time.)

Om een verklaring en hopelijk een oplossing voor deze problemen te vinden, heb ik op Internet gezocht of deze problemen bekend waren. Hier was echter niets over te vinden. Navraag bij de online Technical Support van Business Objects (de leverancier van Crystal Reports) leverde eveneens geen oplossing op.
Bij de beoordeling van de pilot (zie paragraaf 7.10) is in overleg (onder meer) besloten om in een eventuele volgende iteratie van de fase pilotontwikkeling het oplossen van deze problemen als doelstelling te definiëren.

Als laatste heb ik de performancetest uitgevoerd. Deze test heb ik in drie deeltesten uitgevoerd:

1. Opbouwen van een rapport op basis van een ADO.NET XML document dat één tabel bevat.
   Voor deze test heb ik ADO.NET XML documenten ingelezen waarin één tabel was opgenomen (getest tot tabellen met bijna 14000 records). De resultaten van deze tests waren zeer goed: in alle gevallen werd het rapport opgebouwd binnen één seconde. Hiermee werd aan de eis voldaan dat de tijd die benodigd is om een rapport op te bouwen op basis van een .NET dataobject als datasource, minstens gelijk dient te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject.

2. Opbouwen van een rapport op basis van een ADO.NET XML document dat twee tabellen bevat.
   Deze test heb ik uitgevoerd met het ADO.NET XML document dat ik had opgebouwd in de voorgaande bouweenheden. Dit document bevatte de tabellen “ttCustomer” en “ttOrder”. De resultaten van deze test voldeden niet aan de opgestelde performance-eis.
   Op de volgende pagina is een overzicht opgenomen van de deze resultaten.

Case #	# velden ttCustomer	# velden ttOrder	Totaal # records	Totale inleestijd (sec)	# records per seconde
1	1	1	3953	~40	~98
2	3	3	3953	~51	~75
3	6	6	3953	~63	~62
4	6	1	3953	~60	~66
5	1	6	3953	~35	~113
6	10	10	3953	~71	~55
7	10	1	3953	~72	~54
8	1	10	3953	~43	~92

Figuur 27: *Resultaten inlezen ADO.NET XML document met twee tabellen*

Zoals hierboven te zien is, varieert de inleestijd bij deze testgevallen van ongeveer 35 tot ongeveer 72 seconden voor nog geen 4000 records. Deze resultaten liggen ver boven de tijd die benodigd is om een dergelijk rapport op te bouwen op basis van de huidige datasource. Deze tijd wordt namelijk geschat op ongeveer vier seconden.
Om een verklaring en hopelijk een oplossing voor deze problemen te vinden, heb ik op Internet gezocht of deze problemen bekend waren. Dit heeft echter weinig resultaten opgeleverd, waarschijnlijk omdat het werken met ADO.NET XML documenten als datasource een vrij nieuwe techniek is die nog niet veel is toegepast. Op een forum van de online Technical Support van Business Objects werd deze slechte performance wel genoemd door een gebruiker die hetzelfde probleem had ervaren, maar hier werd geen verklaring of oplossing geboden.
Ook heb ik in de documentatie van Visual Studio .NET 2003 gezocht naar mogelijkheden om de opbouw van het XML document te verbeteren. Dit heeft echter geen resultaten opgeleverd.
Omdat ik geen oplossing voor het performanceprobleem kon vinden, heb ik op basis van de testresultaten de volgende conclusies getrokken in het pilotontwikkelrapport:

1. De performance daalt naarmate er meer velden uit beide tabellen in een rapport worden opgenomen;
2. Er is vrijwel geen verschil in performance tussen gevallen waarbij veel velden uit beide tabellen in het rapport zijn opgenomen en gevallen waarbij veel velden uit de tabel “ttCustomer” en weinig velden uit de tabel “ttOrder” zijn opgenomen;
3. De performance daalt sterk in gevallen waarbij weinig velden uit de tabel “ttCustomer” en veel velden uit de tabel “ttOrder” zijn opgenomen.

3. Opbouwen van een rapport op basis van een ADO.NET XML document dat vijf tabellen bevat.
   De resultaten van deze tests bevestigden de conclusies die ik reeds had getrokken na het uitvoeren van de vorige test. In het slechtste geval, waarbij alle velden uit alle tabellen op het rapport waren geplaatst, werden nog geen 10 records per seconde ingelezen.
   De resultaten van deze tests heb ik in het pilotontwikkelrapport verwerkt.

Na het bouwen en testen van deze bouweenheid heb ik het rapport over interfacing uitgebreid met de kennis die ik tijdens het bouwen van deze bouweenheid heb opgedaan. Hiermee had ik de laatste stap (stap 3) uit het stappenplan voor implementatie van een .NET dataobject voor Crystal Reports uitgewerkt (zie paragraaf 5.4.6).

7.8 Invoeringsprocedure

De activiteit die na het bouwen van de bouweenheden was gepland, was het opstellen van een invoeringsprocedure voor de pilot. Het opstellen van een dergelijke procedure heeft tot doel om het operationeel maken van de pilot succesvol te laten verlopen.
De invoeringsprocedure heb ik in de vorm van een stappenplan opgesteld. Om het stappenplan op te kunnen stellen, heb ik de activiteiten geanalyseerd die benodigd zijn om een .NET dataobject als datasource te kunnen gebruiken in Crystal Reports.
Het onderstaande stappenplan is het resultaat van deze analyse.

1. Controleren aanwezigheid benodigde hard- en software:
   - Ontwikkelomgeving Progress (OpenEdge 10);
   - Toegang tot de broncode van de applicatie XL-Enz;
   - Toegang tot de database waar XL-Enz gebruik van maakt;
   - Crystal Reports (versie 9 of hoger);
   - Microsoft .NET Framework (versie 1.1 of hoger) is geïnstalleerd op de client.
2. Vervangen huidige functie voor genereren Progress dataobject dat rapportdata bevat met nieuwe functie.
   De functie waarin een TempTable wordt aangemaakt die alle rapportdata bevat, dient te worden vervangen met de nieuwe functie waarin een ProDataSet-object wordt aangemaakt. Er dient een backup te worden gemaakt van de huidige functie.
3. Vervangen huidige functie voor genereren ADO XML bestand met nieuwe functie.
   De functie die op dit moment wordt gebruikt om een ADO XML bestand te genereren, dient te worden vervangen met de nieuwe functie waarmee een ProDataSet-object wordt geconverteerd naar een ADO.NET dataobject dat wordt opgeslagen in een XML bestand. Er dient een backup te worden gemaakt van de huidige functie.
4. Wijzigen configuratie van datasources in Crystal Reports.
   Momenteel is Crystal Reports geconfigureerd op het gebruik van ADO XML datasources (d.m.v. ODBC). Deze configuratie dient te worden gewijzigd in het gebruik van een datasource op basis van ADO.NET XML.

Vervolgens heb ik testspecificaties opgesteld die ter controle op een correcte invoering kunnen dienen. Als laatste heb ik een noodprocedure opgesteld, die kan worden uitgevoerd wanneer de invoeringsprocedure mislukt. Deze procedure heb ik ook in de vorm van een stappenplan opgesteld.

7.9 Integreren bouweenheden

De volgende geplande activiteit was het integreren van de bouweenheden. De bouweenheden bleken echter op zichzelf staande componenten te zijn die elk een eigen functionaliteit leverde, zonder afhankelijk van elkaar te zijn. Dit nam de noodzaak tot integratie van de bouweenheden weg. Om deze reden is deze activiteit dan ook achterwege gelaten.

7.10 Beoordelen en testen

In de activiteit “Beoordelen en testen” heb ik in samenwerking met de begeleider van het project de ontwikkelde pilot beoordeeld. Na de testresultaten van de bouweenheden te hebben beschreven en doorgenomen, is op basis van deze resultaten een beoordeling van de pilot opgesteld.
Daarbij is gecontroleerd of de pilot na de eerste iteratieslag voldoet aan de eisen die in de definitiestudie opgesteld zijn. Hieronder is het resultaat van deze controle opgenomen.

ID	Omschrijving	MoSCoW	Voldaan
DF1	Een gebruiker is in staat om Crystal Reports een rapport te laten genereren op basis van een .NET dataobject als datasource	M	Ja
DN1	Het dataobject kan gegevens uit een bestaande Progress database inlezen	M	Ja
DN2	Het dataobject is te converteren van een Progress object naar een ADO.NET object	M	Ja
DN3	Het dataobject wordt opgebouwd op de server als een Progress object, vervolgens geconverteerd naar een ADO.NET object en daarna verstuurd naar client	M	Nee: dataobject wordt lokaal opgebouwd
DN4	De tijd die benodigd is om een rapport op te bouwen op basis van een .NET dataobject als datasource, dient minstens gelijk te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject	M	Nee: zie resultaten tests

Figuur 28: *Resultaten eerste iteratieslag pilot 1*

Zoals hierboven te zien is, is niet aan alle eisen voldaan. Uiteindelijk is de pilot als volgt beoordeeld.

Aan de eis DN3 is niet voldaan, omdat ervoor gekozen is tijdens de eerste iteratie het dataobject lokaal op te bouwen. Een volgende iteratie kan tot doel hebben om (onder meer) aan deze eis te voldoen.
Bij een ADO.NET XML bestand dat meerdere tabellen bevat, voldoet de performance van het opbouwen van een rapport in Crystal Reports niet aan de eis dat deze gelijk of beter moet zijn dan de performance van het opbouwen van een rapport op basis van het huidige ADO XML dataobject.
Het opbouwen van een rapport op basis van een ADO.NET XML bestand dat één tabel bevat, verloopt zeer snel: tijdens tests waren dergelijke rapporten binnen één seconde opgebouwd.

De slechte performance bij het opbouwen van een rapport dat is opgebouwd uit meerdere tabellen is momenteel de bottleneck voor een succesvolle implementatie van deze pilot. In een eventuele volgende iteratie van de fase pilotontwikkeling dient aandacht te worden besteed aan verbetering van deze performance.

7.11 Afronding

Op het moment dat de pilot werd beoordeeld, was nog niet bekend of de performanceproblemen te verhelpen waren. Om deze reden en vanwege tijdsdruk is besloten de eerste iteratieslag van deze pilot af te ronden en de ontwikkeling van de tweede pilot te starten.

Tijdens de ontwikkeling van de tweede pilot is het performanceprobleem voorgelegd aan de ontwikkelaars van Reflecta en is het probleem genoemd bij de online Technical Support van Business Objects. Tegen het einde van de afstudeerperiode werd bekend dat de slechte performance waarschijnlijk wordt veroorzaakt doordat de data in het ADO.NET XML document niet is geïndexeerd op de velden waar een relatie op is gedefinieerd. Als gevolg hiervan dient het volledige XML document te worden doorlopen om de juiste combinatie van records uit verschillende tabellen te vinden. Dit werkt sterk vertragend. Een oplossing zou kunnen zijn om indexering toe te voegen aan het ADO.NET XML document. Onderzoek heeft echter uitgewezen dat ADO.NET XML documenten geen indexering ondersteunen.
Een andere oplossing voor het probleem zou de nieuwste versie van Crystal Reports kunnen bieden. In een whitepaper waarin een overzicht wordt gegeven van de functionaliteiten van Crystal Reports versie 10 belooft Business Objects een betere performance voor het inlezen van ADO.NET DataSets.

Faster DataSet Processing
In Crystal Reports 10, architectural changes have dramatically improved DataSet performance. Testing on a 6.8MB sample DataSet showed that the new driver is approximately 2.2 times faster than previous versions, uses 74% less private memor, and uses 63% less working set memory. Results may vary for your specific implementation.
(Bron: Business Objects, Whitepaper “Crystal Reports 10 for .NET Feature Overview)

Tijdens de afstudeerperiode was Crystal Reports versie 10 echter nog niet beschikbaar, waardoor deze oplossing nog niet is getest.
Wanneer de performance niet aanzienlijk kan worden verbeterd, wordt waarschijnlijk besloten om geen ADO.NET XML documenten te gebruiken als datasource voor rapporten van Crystal Reports.

Na het schrijven van een samenvatting van de inhoud van het pilotontwikkelrapport en het controleren van de rapportopmaak, heb ik het rapport opgeleverd. De begeleider heeft het rapport vervolgens doorgenomen en is akkoord gegaan met de inhoud ervan. Het pilotontwikkelrapport is als externe bijlage opgenomen bij dit verslag.
Voor het doorlopen van de fase pilotontwikkeling voor deze pilot heb ik ruim drie weken nodig gehad. Dit kwam vrijwel overeen met de drie weken die ik voor deze fase had gepland, zodat er geen noemenswaardige wijzigingen in de planning van het project noodzakelijk waren.

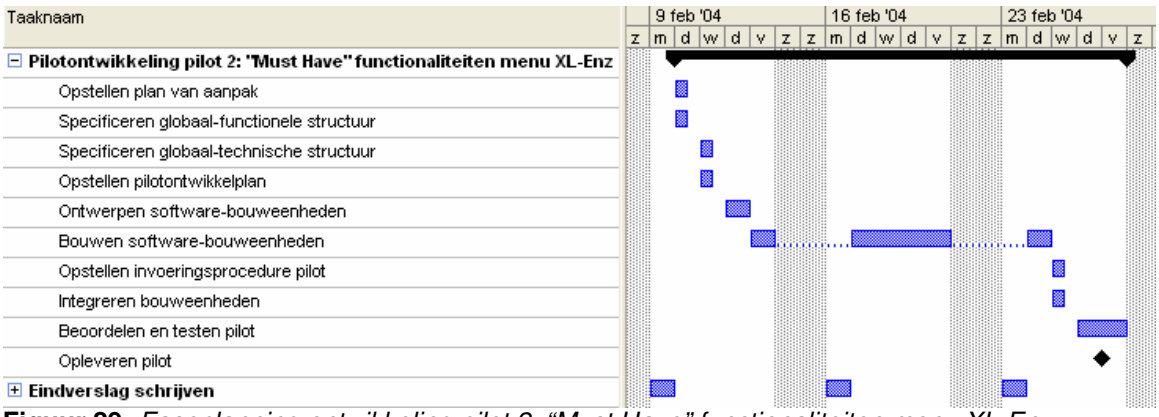
8. Pilotontwikkeling pilot 2

8.1 Inleiding

Na de beoordeling en afronding van de vorige pilot is de ontwikkeling van de tweede pilot gestart. Tijdens de definitiestudie is het ontwikkelen van de “Must Have” functionaliteiten van het menu van XL-Enz gedefinieerd als pilot die als tweede wordt ontwikkeld. Dit hoofdstuk beschrijft de activiteiten die zijn uitgevoerd tijdens de ontwikkeling van deze pilot.
De opbouw van dit hoofdstuk is gelijk aan de opbouw van het vorige hoofdstuk. Als eerste wordt in paragraaf 8.2 de totstandkoming van het plan van aanpak voor de ontwikkeling van de pilot beschreven. In de paragrafen 8.3 en 8.4 beschrijf ik respectievelijk het ontwerpen van de functionele en de technische structuur van de pilot. Vervolgens wordt in paragraaf 8.5 de totstandkoming van het pilotontwikkelplan beschreven, waarin software-bouweenheden zijn gedefinieerd. Het ontwerp van deze bouweenheden wordt besproken in paragraaf 8.6. In paragraaf 8.7 wordt het bouwen van de bouweenheden beschreven. Vervolgens wordt in paragraaf 8.8 de totstandkoming van de invoeringsprocedure van de pilot beschreven. Paragraaf 8.9 behandelt het proces van integratie van de bouweenheden en paragraaf 8.10 beschrijft hoe de pilotdelen zijn beoordeeld en getest. Als laatste wordt in paragraaf 8.11 de afronding van de fase pilotontwikkeling van deze pilot beschreven.

8.2 Plan van aanpak

Net als voor de vorige pilot heb ik eerst een plan van aanpak opgesteld voor de ontwikkeling van deze pilot. Hierbij heb ik ook weer elke activiteit bestudeerd die door IAD wordt voorgeschreven en gekeken of de activiteiten relevant waren in het kader van deze pilot. Dezelfde activiteiten die voor de vorige pilot niet relevant bleken, bleken dat voor deze pilot ook niet en dat ook om dezelfde redenen.
Vervolgens heb ik op basis van een inschatting van de benodigde tijd voor elke activiteit de onderstaande planning opgesteld.



Figuur 29: *Faseplanning ontwikkeling pilot 2: “Must Have” functionaliteiten menu XL-Enz*

Als laatste heb ik de iteratiestrategie van de pilot beschreven. Deze strategie is gelijk aan de strategie die ik heb beschreven voor de vorige pilot.

8.3 Functionele structuur

Bij het ontwerpen van de functionele inhoud van deze pilot heb ik gebruik gemaakt van de kennis die ik heb opgedaan tijdens het uitvoeren van deze activiteit voor de eerste pilot. Als eerste heb ik de functionele reikwijdte van de pilot vastgesteld door een overzicht te geven van de “Must Haves” die zouden worden ontwikkeld in de pilot. Hiermee werden de functionele doelstellingen en begrenzingen van de pilot duidelijk.

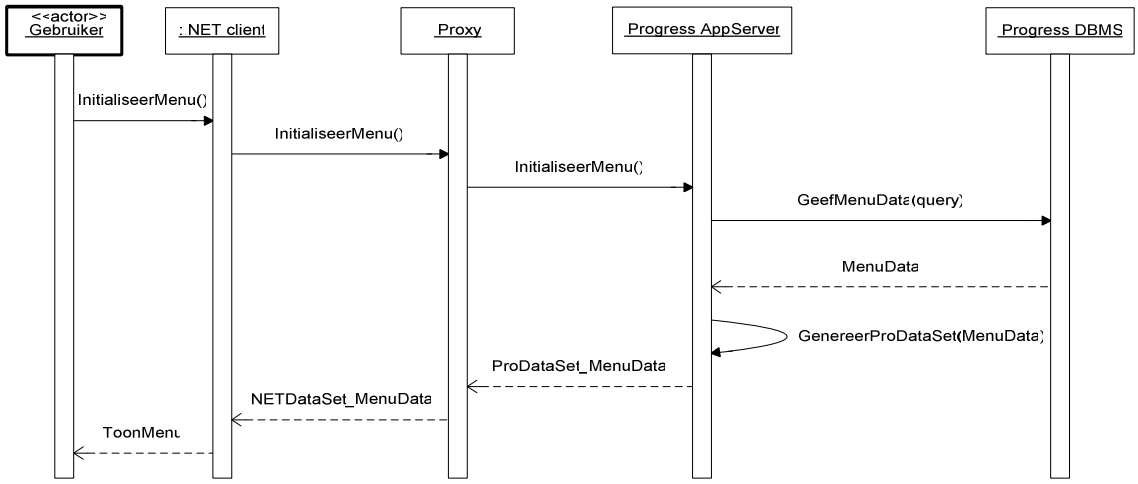
ID	Omschrijving
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma’s van XL-Enz kunnen worden opgestart
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer
MF3	Een gebruiker kan het menu afsluiten
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database
MN2	Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven
MN4	Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress dataobject en vervolgens doorgegeven aan de .NET client, waar het object wordt geconverteerd naar een ADO.NET dataobject

Figuur 30: *Functionele reikwijdte pilot*

Vervolgens heb ik, net als bij pilot 1, voor elke relevante use-case uit de definitiestudie een UML sequencediagram ontworpen, om de afhandeling van deze use-cases in detail te kunnen beschrijven. In deze pilot werden de volgende use-cases geïmplementeerd:

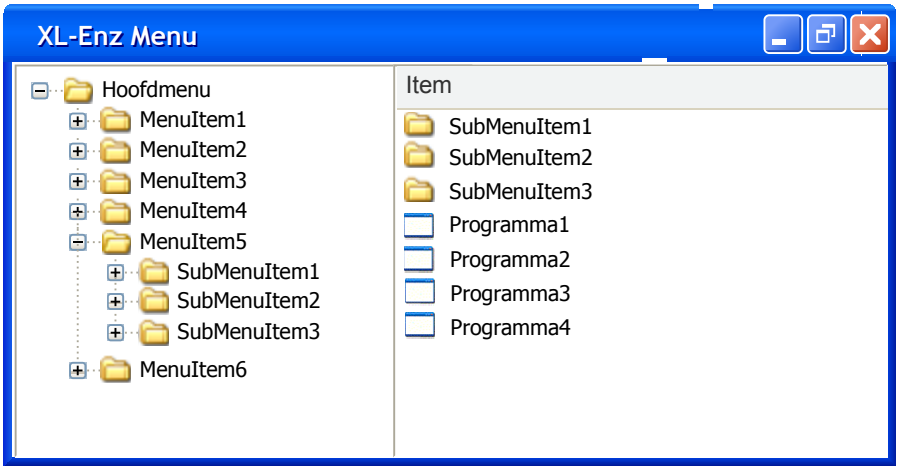
- Menu initialiseren;
- Menu navigeren;
- Programma starten;
- Menu afsluiten.

Ter illustratie is het sequencediagram voor het initialiseren van het menu hieronder opgenomen.



Figuur 31: *Sequencediagram “Menu initialiseren”*

Bij elk sequencediagram heb ik ter verduidelijking een tekstuele beschrijving opgenomen, waarin ik elke interactie uit de sequencediagrammen heb omschreven. Als laatste heb ik een prototype van de gebruikersinterface van het menu ontworpen, dat als uitgangspunt kon fungeren bij de bouw van de gebruikersinterface. Het prototype is hieronder opgenomen.



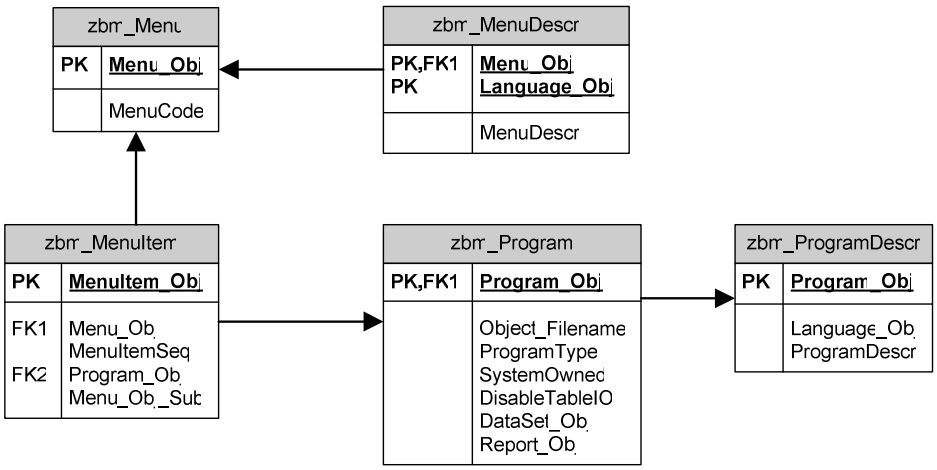
Figuur 32: *Prototype gebruikersinterface menu*

8.4 Technische structuur

Het ontwerpen van de technische structuur heb ik deels op gelijke wijze aangepakt als bij de vorige pilot. Ook hier heb ik de fysieke allocatie ontworpen door de verantwoordelijken van de objecten uit de sequencediagrammen te beschrijven. De taken van het proxy-object heb ik bijvoorbeeld als volgt omschreven.

- Proxy
  Het proxy-object fungeert als een vertaler tussen de .NET omgeving en de Progress omgeving. De proxy neemt de data en communicatie vanuit de Progress of .NET omgeving in ontvangst en vertaalt deze naar begrijpelijke data voor de andere omgeving. De proxy bestaat uit een DLL (Dynamic Link Library).

Vanwege het feit dat het menu gebruik maakt van een reeds bestaande Progress database, heb ik geen gedetailleerd databasemodel ontworpen. Ik heb ervoor gekozen om alleen een overzicht van de reeds bestaande tabelstructuur van het menu in het rapport op te nemen. Daarnaast heb ik vermeld dat de exacte definities van de tabellen, indien gewenst, kunnen worden opgevraagd uit de database van de applicatie XL-Enz. Het databasemodel dat ik in het pilotontwikkelrapport heb opgenomen, is op de volgende pagina weergegeven. Omdat het menu de benodigde tabellen met hun onderlinge relaties weergeeft, leek dit ontwerp mij voldoende om het dataobject voor het menu te kunnen ontwikkelen.



Figuur 33: *Databasemodel menu*

Het model heb ik verduidelijkt door in het pilotontwikkelrapport een beschrijving op te nemen van de functie van elke tabel en van de wijze waarop de tabellen aan elkaar zijn gerelateerd.

8.5 Pilotontwikkelplan

Op basis van de pilotdelen die ik voor deze pilot had gedefinieerd tijdens de definitiestudie, heb ik een pilotontwikkelplan ontworpen. Daarbij heb ik dezelfde werkwijze gebruikt als voor het opstellen van het pilotontwikkelplan voor de eerste pilot. De volgende bouweenheden heb ik gedefinieerd:

1. Opzetten ontwikkelomgeving;
2. Bouwen functie voor het opnemen van de menudata in een Progress ProDataSet-object;
3. Converteren ProDataSet-object naar .NET dataobject;
4. Menu initialiseren op basis van menu-dataobject;
5. Ontwikkelen navigatiemogelijkheid;
6. Ontwikkelen mogelijkheid tot afsluiten menu;
7. Menu compileren en registreren als ActiveX object;
8. Menu als ActiveX object invoegen in Progress;
9. Ontwikkelen mogelijkheid opstarten van programma's.

Het definiëren van de eerste drie bouweenheden kostte weinig moeite, omdat ik tijdens de eerste pilot ongeveer dezelfde activiteiten moest uitvoeren in dezelfde volgorde. Tijdens de eerste pilot bleek deze indeling van bouweenheden goed te functioneren, waardoor ik heb besloten dezelfde indeling voor de tweede pilot te gebruiken.
Het definiëren van bouweenheid 4 tot en met 9 vereiste enig onderzoek. Aanvankelijk had ik bouweenheid 9 na bouweenheid 6 gepland. De begeleider merkte echter tijdens het bespreken van de bouweenheden op, dat het opstarten van programma's alleen vanuit een Progress omgeving kan worden uitgevoerd, en dus niet vanuit een .NET omgeving. Om deze reden was het noodzakelijk eerst het menu als ActiveX object in een Progress omgeving in te voegen, zodat het menu met Progress kon communiceren bij het opstarten van een programma. Daarom is het ontwikkelen van de mogelijkheid van het opstarten van programma's gepland na het invoegen van het menu als ActiveX object.

Vanwege mijn onbekendheid met ActiveX objecten, heb ik eerst onderzoek verricht naar de wijze waarop het menu zou kunnen worden ingevoegd in een Progress omgeving. Binnen Reflecta bleek een dergelijke activiteit eerder te zijn uitgevoerd, waarbij een object dat was geschreven in Visual Basic als ActiveX object in een Progress omgeving was ingevoegd. Om dit te realiseren bleek het object eerst in het Windows register te worden geregistreerd, voordat het kon worden ingevoegd in Progress.
Om die reden heb ik het invoegen van het menu als ActiveX object verdeeld in twee bouweenheden (bouweenheid 7 en 8).

Nadat ik had gecontroleerd of met deze indeling van de bouweenheden alle te ontwikkelen functionaliteiten werden gedekt, heb ik de iteratiestrategie voor het ontwikkelen van de bouweenheden beschreven. Deze strategie was gelijk aan de iteratiestrategie voor de eerste pilot: de pilot zou in de eerste iteratie in een lokale omgeving worden ontwikkeld, om risico's als gevolg van het gebruik van nieuwe technieken te vermijden. Daarnaast heb ik ook hier vermeld dat de pilot na de eerste iteratie wordt getest op een correcte werking, op functionaliteit en op performance. Wanneer hieruit zou blijken dat de pilot verder kan worden ontwikkeld, zou in een volgende iteratie de implementatie plaats kunnen vinden in een gedistribueerde omgeving.

8.6 Ontwerp software-bouweenheden

Na het definiëren van de bouweenheden was de volgende activiteit het ontwerpen van deze bouweenheden. Dit ontwerp dient zodanig te zijn opgesteld, dat het mogelijk moet zijn om op basis hiervan de bouweenheden te implementeren.

8.6.1 Ontwerpen bouweenheden

Het ontwerpen van de bouweenheden heb ik op gelijke wijze aangepakt als bij de eerste pilot. Het ontwerp heb ik ook weer gedurende de fase pilotontwikkeling verder aangevuld naar aanleiding van nieuw opgedane kennis en inzicht. Op basis van de kennis die ik had opgedaan na het uitvoeren van het onderzoek naar interfacing, heb ik in eerste instantie een tekstueel ontwerp van elke bouweenheid gemaakt. Het stappenplan dat ik had opgesteld voor de ontwikkeling van een .NET gebruikersinterface (zie paragraaf 5.4.5) heb ik daarbij als uitgangspunt gebruikt. Het ontwerp voor bouweenheid 3 "Converteren Progress menu-dataobject naar .NET menu-dataobject" zag er bijvoorbeeld als volgt uit.

Wanneer het ProDataSet-object is geïmplementeerd en getest, kan deze worden geconverteerd naar een .NET dataobject. Hiermee wordt de menudata toegankelijk in de .NET omgeving. Ook de functie waarin het Progress dataobject wordt samengesteld dient toegankelijk te zijn in de .NET omgeving.
Deze bouweenheid kan worden geïmplementeerd door gebruik te maken van de functionaliteiten van de tool "Proxy Generator" (ProxyGen), die standaard wordt meegeleverd met Progress OpenEdge 10. Deze tool genereert een proxy-object (Dynamic Link Library (DLL)), dat de communicatie en conversie afhandelt tussen de Progress omgeving en de .NET omgeving.

Net als het ontwerp voor pilot 1 heb ik dit ontwerp uitgebreid tijdens de bouw van de bouweenheden door de relevante kennis die ik opdeed te verwerken in het rapport over .NET interfacing. Het stappenplan wat ik daarin had opgesteld, heb ik uitgebreid zoals ik heb beschreven in paragraaf 5.4.5.
In het pilotontwikkelrapport heb ik bij het tekstuele ontwerp ook weer een verwijzing opgenomen naar het rapport over interfacing.

> Voor gedetailleerde informatie over het genereren van het proxy-object wordt verwezen naar hoofdstuk 5 van het rapport “Interfacing tussen Progress en Microsoft .NET”. Daarnaast biedt de documentatie die Progress beschikbaar stelt over het genereren van proxy-objecten voldoende informatie om deze bouweenheid te kunnen implementeren.

Ook voor deze pilot heb ik gezocht naar mogelijkheden om UML diagrammen toe te passen in het ontwerp om hiermee het tekstuele ontwerp aan te vullen. De diagrammen leken echter ook hier geen toegevoegde waarde te bieden aan het reeds bestaande ontwerp, om dezelfde reden als de reden die ik heb genoemd in het vorige hoofdstuk.

8.6.2 Opstellen testspecificaties

Voor elke bouweenheid heb ik testspecificaties ontworpen in de vorm van “black box” testsets. Hieronder zijn ter illustratie de testspecificaties opgenomen van bouweenheid 5, waarin de navigatiemogelijkheden van het menu worden geïmplementeerd.

Testspecificaties bouweenheid 5: Ontwikkelen navigatiemogelijkheid			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
5.1	Werking navigatie in TreeView	Klik op menu-item	Inhoud van geselecteerd menu-item is zichtbaar in ListView
5.2		Dubbelklik op menu-item in TreeView waarvan inhoud niet zichtbaar is	Submenu(‘s) van geselecteerd item is/zijn zichtbaar in TreeView en submenu(‘s) en programma(‘s) is/zijn zichtbaar in ListView
5.3		Dubbelklik op menu-item in TreeView waarvan inhoud zichtbaar is	Submenu(‘s) van geselecteerd item is/zijn onzichtbaar in TreeView en submenu(‘s) en programma(‘s) is/zijn zichtbaar in ListView
5.4		Klik op een “-“ (collapse) icoon voor menu-item	Inhoud van betreffende menu-item is onzichtbaar
5.5		Klik op een “+“ (expand) icoon voor menu-item	Inhoud van betreffende menu-item is zichtbaar
5.6	Werking navigatie in ListView	Klik op menu-item in ListView	Geen uitvoer
5.7		Dubbelklik op menu-item in ListView	Inhoud geselecteerd menu-item is zichtbaar in ListView, geselecteerd menu-item is ook in TreeView zichtbaar en is geselecteerd
5.8		Dubbelklik op programmanaam in ListView	Geen uitvoer

Figuur 34: *Testspecificaties bouweenheid 5*

Net als de testspecificaties voor de bouweenheden van pilot 1, heb ik deze testspecificaties deels gedurende het bouwen van de bouweenheden opgesteld. Tijdens de bouw kreeg ik namelijk meer inzicht in de werking van de bouweenheden. De bijzondere gevallen die ik daarbij tegenkwam, heb ik verwerkt in de testspecificaties. Zo heb ik de testspecificaties van bouweenheid 7 tijdens de bouw uitgebreid met een gedetailleerde beschrijving van de exacte waarden die in het Windows register dienen te worden geschreven wanneer het menu-object wordt geregistreerd. Ik achtte het noodzakelijk om deze beschrijving in de testspecificaties op te nemen, omdat een correcte registratie van het object essentieel is om het object te kunnen invoegen in een Progress omgeving.

8.7 Bouw software-bouweenheden

Na het maken van het eerste ontwerp van de bouweenheden ben ik begonnen met de implementatie van deze bouweenheden. Deze paragraaf beschrijft het implementeren en testen van de bouweenheden.

8.7.1 Opzetten ontwikkelomgeving

De (lokale) ontwikkelomgeving die ik had gebruikt om de eerste pilot te ontwikkelen, kon ik ook voor deze pilot gebruiken. Deze bouweenheid kon ik dus meteen testen door de testgevallen uit te voeren. Alle benodigde onderdelen van de ontwikkelomgeving bleken correct te functioneren, waardoor de implementatie van deze bouweenheid geen problemen heeft opgeleverd.

8.7.2 Bouwen functie opnemen menudata in ProDataSet-object

Als eerste diende in Progress een functie te worden geschreven, waarmee de menugegevens uit de XL-Enz database werden opgevraagd en in een ProDataSet-object werden opgenomen.
Door gebruik te maken van de ervaring die ik inmiddels had opgedaan met het implementeren van ProDataSet-objecten, was deze bouweenheid binnen een halve dag gebouwd en getest. Het databasemodel dat ik had ontworpen in het pilotontwikkelrapport vormde daarbij een waardevol uitgangspunt: alle benodigde tabellen en relaties waren daarin weergegeven. Het enige nieuwe aspect waarnaar ik onderzoek moest verrichten, was de wijze waarop de functie en het ProDataSet-object konden worden aangeroepen vanuit een .NET omgeving. De documentatie van Progress bevatte echter een heldere beschrijving van de wijze waarop dit kon worden gerealiseerd.
Na de functie succesvol hebben getest, heb ik het bouwen van de functie beschreven in het rapport over .NET interfacing. Hiermee had ik stap 1 (“Bouwen Progress procedures”) uit het stappenplan voor implementatie van een .NET gebruikersinterface uitgewerkt (zie paragraaf 5.4.5).

8.7.3 Converteren ProDataSet-object naar .NET dataobject

Om de menudata uit het ProDataSet-object toegankelijk te maken in de .NET omgeving, diende deze data te worden geconverteerd naar een .NET dataobject. Uit het onderzoek naar interfacing was reeds gebleken dat deze conversie wordt uitgevoerd door een

proxy-object dat als “vertaler” tussen de twee omgevingen fungeert. Bij Progress OpenEdge 10 Beta was een tool genaamd “Proxy Generator” meegeleverd, waarmee dergelijke proxy-objecten konden worden gegenereerd. De werking van deze tool heb ik onderzocht door de documentatie van Progress hierover te bestuderen en door vervolgens enkele tests uit voeren met de tool. Doordat het genereren van proxy-objecten in eerste instantie mislukte (zie paragraaf 5.4.5), kon ik dit niet geheel testen. Gelukkig kwam op dat moment Progress OpenEdge 10A (Alfa) beschikbaar, waarbij de tool wel correct functioneerde.
Bij OpenEdge 10A waren enkele voorbeeldapplicaties meegeleverd, waarmee ik de werking van de tool “Proxy Generator” uitgebreid kon onderzoeken en testen. Het werken met proxy-objecten was geheel nieuw voor mij. Door de werking van dergelijke objecten in de praktijk te onderzoeken, heb ik de kennis opgedaan die benodigd was voor dit project.
Na het uitvoeren van de tests wist ik namelijk hoe ik een proxy-object moest genereren en hoe ik vervolgens een Progress-functie via het proxy-object kon aanroepen vanuit de .NET omgeving. De proxy bleek een ProDataSet-object te converteren naar een ADO.NET DataSet-object. In de .NET omgeving was het Progress ProDataSet-object dus benaderbaar als een ADO.NET DataSet.

Nadat ik het proxy-object voor het menu had gegenereerd en getest volgens de opgestelde testspecificaties, heb ik stap 2 uit het stappenplan (“Genereren proxy tussen Progress en .NET”) beschreven aan de hand van een voorbeeldimplementatie van een proxy-object (zie paragraaf 5.4.5).

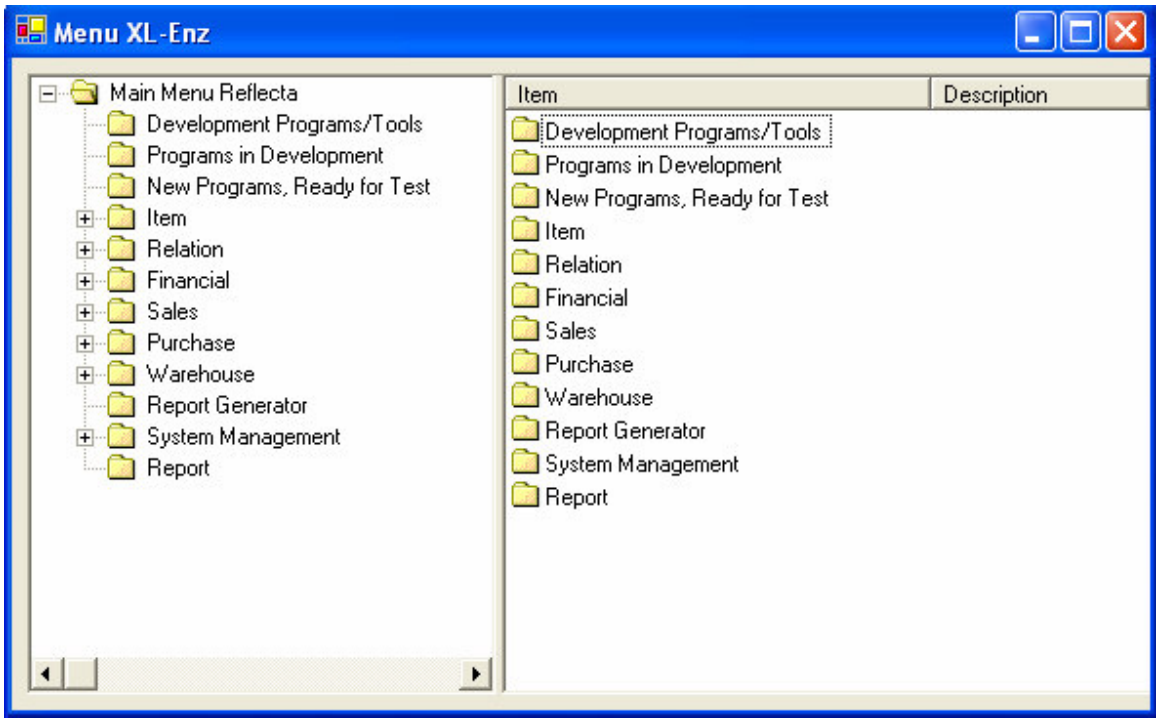
8.7.4 Menu initialiseren op basis van menu-dataobject

Na het succesvol testen van het proxy-object kon de gebruikersinterface van het menu en de initialisatie ervan worden gebouwd in Visual Studio .NET 2003, door middel van de taal C#. Omdat ik vrijwel geen ervaring had met het werken in deze complexe ontwikkelomgeving en met deze programmeertaal, heb ik eerst de werking van de voorbeeldapplicaties bestudeerd die waren meegeleverd met OpenEdge 10A. In deze applicaties werden voorbeelden gegeven van het opvragen en tonen van gegevens uit een Progress database in een .NET applicatie.
Door de meegeleverde documentatie van Progress en de documentatie van Visual Studio .NET 2003 te bestuderen, heb ik de kennis opgedaan waarmee ik een .NET applicatie kon bouwen die via het proxy-object communiceerde met de Progress omgeving.
Omdat het menu als een ActiveX object zou worden ingevoegd in Progress, mocht het menu niet als een “Windows applicatie” worden gebouwd, maar als een “UserControl”. Een UserControl bestaat na compilatie uit een DLL bestand, dat kan worden ingevoegd als control (ActiveX object) in een applicatie. Wegens mijn onbekendheid met UserControls was het noodzakelijk ook hier documentatie over te bestuderen.

Het bestuderen van documentatie en de implementatie van het menu vergde meer tijd dan gepland wegens onbekendheid met zowel de ontwikkelomgeving als de taal C#. Zo liep ik tijdens de bouw bijvoorbeeld tegen problemen aan bij het gebruik van “references”. Aan de .NET applicatie kunnen references worden toegevoegd, waardoor klassen, methoden en events uit die reference kunnen worden aangeroepen in de applicatie. Zo maakt de toevoeging van de reference “System.Windows.Forms” het mogelijk om de functionaliteiten van een Windows Form te gebruiken in de applicatie.

Om de Progress functies aan te kunnen roepen vanuit de .NET applicatie, was het (o.a.) noodzakelijk om het proxy-object als reference toe te voegen aan de applicatie. De problemen met references ontstonden toen ik deze reference toevoegde. Bij compilatie van de applicatie trad een foutmelding op: de cache van het systeem bleek verschillende versies van de reference te bevatten, wat niet was toegestaan.
De documentatie van Visual Studio .NET 2003 gaf een verklaring voor dit probleem. Deze verklaring bevatte echter zoveel voor mij onbekende termen (“versioning”, “assemblies”, “namespaces” etc.), dat ik mij eerst deze termen eigen moest maken, wilde ik de oplossing voor het probleem kunnen begrijpen. Uiteindelijk bleek het probleem oplosbaar door alle versies van de reference te verwijderen en de reference opnieuw te compileren.

De begeleider van het project heeft het concept uitgelegd van de wijze waarop de boomstructuur van het menu wordt gebouwd. Zo wordt de boom bij initialisatie bijvoorbeeld niet in zijn geheel opgebouwd, om de performance van het opbouwen van de boom te verhogen. Alleen de menu-items die zichtbaar zijn, worden aan de boom toegevoegd. Aan de menu-items (“nodes”) die submenu’s bevatten, wordt een zogenaamde “dummy node” toegevoegd. Wanneer de gebruiker de submenu’s wil zien, wordt deze “dummy node” verwijderd, en worden de submenu-items aan de boom toegevoegd.
Daarnaast bleken de menu-items op een vastgelegde volgorde aan de boomstructuur te moeten worden toegevoegd. Deze volgorde was vastgelegd in de database.
Na ongeveer vier dagen had ik een .NET UserControl gebouwd waarin de menugegevens uit de Progress database werden opgevraagd en in een boomstructuur werden getoond. Het resultaat is hieronder opgenomen.

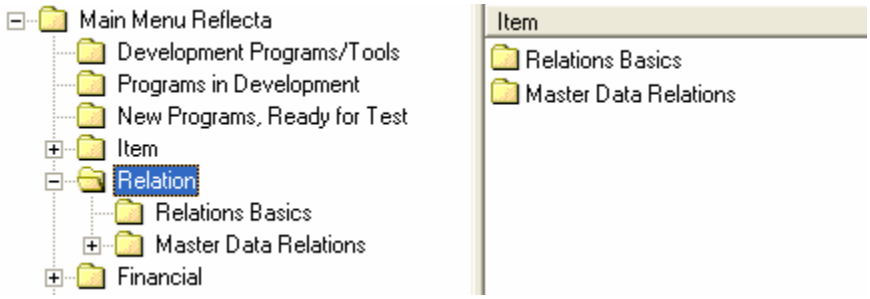


Figuur 35: *Geïnitialiseerd menu*

Nadat ik de initialisatie van het menu had getest volgens de opgestelde testspecificaties, heb ik stap 3 uit het stappenplan (“Bouwen .NET client applicatie”) beschreven aan de hand van een voorbeeldimplementatie van een .NET applicatie (zie paragraaf 5.4.5).

8.7.5 Ontwikkelen navigatiemogelijkheid

De wijze van navigatie door het menu diende gelijk te zijn aan de wijze waarop door de Windows Explorer kan worden genavigeerd. Deze navigatiemogelijkheden heb ik in bouweenheid 5 ontwikkeld. Deze bouweenheid bestond voornamelijk uit het implementeren van “events”. Ook hiervoor ontbrak mij aanvankelijk de benodigde kennis, zodat ik eerst weer de documentatie van Visual Studio .NET 2003 heb bestudeerd.
De boomstructuur van het menu is geïmplementeerd met behulp van de klasse “TreeView”. Deze klasse biedt functionaliteiten om alle aspecten van een menu in de vorm van een boom te implementeren. Zo biedt deze klasse “events” die worden afgevuurd wanneer de gebruiker een actie uitvoert op de boomstructuur. De event “BeforeExpand” wordt bijvoorbeeld uitgevoerd nadat de gebruiker “dubbelklikt” op een menu-item dat submenu-items bevat (bijvoorbeeld het menu-item “Relation” uit het figuur hierboven). Voordat de submenu’s van “Relation” worden getoond, wordt een “BeforeExpand” event afgevuurd. Tijdens dit event dient de “dummy node” te worden verwijderd en de werkelijke submenu’s van “Relation” te worden toegevoegd.
Nadat dit event is uitgevoerd, wordt een “AfterSelect” event afgevuurd, waarmee de submenu’s worden getoond in de boom en in de “ListView”. De ListView is het rechterdeel van het menu, waarin de inhoud van het geselecteerde menu-item wordt getoond.
Het resultaat van deze actie is hieronder te zien: de submenu’s van “Relation” worden getoond.



Figuur 36: *Fragment menu na navigatie*

Het implementeren van de navigatiemogelijkheden vergde evenals de implementatie van de vorige bouweenheid meer tijd dan gepland wegens de extra tijd die benodigd was voor bestudering van documentatie, voor het aanleren van de programmeertaal en voor het leren werken met de ontwikkelomgeving.
Na ongeveer vier dagen werkte de navigatie volgens de testspecificaties en kon deze bouweenheid als succesvol geïmplementeerd worden beschouwd.
Het project begon op dit moment achter op de planning te lopen: de anderhalve week die ik had gepland voor de bouw van de bouweenheden was reeds verstreken, terwijl er nog vier bouweenheden geïmplementeerd moesten worden. In overleg met de begeleider is hierop besloten om de doorlooptijd van de pilot met één week te verlengen. Dit had tot gevolg dat de beschikbare tijd voor de overige activiteiten van het project

werd verkort. In overleg is toen besloten om de fase invoering eventueel niet meer binnen de gestelde projectperiode uit te voeren.

8.7.6 Ontwikkelen mogelijkheid tot afsluiten menu

De bouw van de mogelijkheid tot het afsluiten van het menu bleek weinig moeite te kosten. In deze bouweenheid was het bouwen van een functie gepland waarmee de gebruiker werd uitgelogd en waarmee de verbinding met de Progress AppServer werd verbroken. Een dergelijke functie bleek na bestudering van de documentatie van Visual Studio .NET 2003 overbodig te zijn: in .NET worden activiteiten als verbindingen verbreken en uitloggen automatisch afgehandeld door de omgeving. In de broncode van de applicatie was al automatisch een functie opgenomen die wordt uitgevoerd bij het afsluiten van de applicatie. Deze functie verbreekt alle openstaande verbindingen en verzorgt “garbage collection”.
Het afsluiten van het menu heb ik getest volgens de opgestelde testspecificaties. Na alle testgevallen succesvol te hebben doorlopen, heb ik deze bouweenheid als afgerond beschouwd.

8.7.7 Menu compileren en registreren als ActiveX object

Om het menu als een ActiveX object te kunnen invoegen in een Progress omgeving, diende het menu op de juiste wijze te worden gecompileerd en vervolgens te worden geregistreerd in het Windows register. Hiermee zou het menu bekend worden binnen het systeem en als een ActiveX object kunnen worden ingevoegd in elke ontwikkelomgeving.
Wegens mijn onbekendheid met de wijze van registratie van ActiveX objecten, heb ik wederom de documentatie van Visual Studio .NET 2003 bestudeerd. Hieruit leerde ik dat een .NET object moet worden ge”wrapped” naar een niet-.NET object, om in een andere omgeving dan .NET te kunnen worden gebruikt. De “managed” .NET code kan namelijk niet worden gebruikt door “unmanaged” omgevingen (zie ook paragraaf 5.4.3). “Unmanaged” objecten maken gebruik van het “Component Object Model” (COM), terwijl .NET objecten communiceren met de .NET Common Language Runtime (CLR).

.NET objecten worden met behulp van enkele tools ge”wrapped” naar COM objecten. Deze tools worden meegeleverd met Visual Studio .NET 2003. Zo kan de Assembly Registration Tool worden gebruikt om het .NET object te registreren in het Windows register.
Tijdens het gebruik van de tools liep ik tegen problemen aan, waardoor het object niet kon worden geregistreerd. De foutmeldingen die werden gegeven, maakte het noodzakelijk om wederom extra studie te verrichten naar de betekenis van nieuwe termen. Zo diende het object (dat bestond uit een DLL bestand) te worden “ondertekend” (signed) met een “strong name”. Na het bestuderen van documentatie hierover, bleek dat bij het DLL bestand informatie over het bestand moest worden opgenomen, waardoor het bestand uniek wordt gemaakt binnen het systeem.

Nadat het registreren van het object leek te zijn gelukt, heb ik geprobeerd het menu in een Progress omgeving in te voegen. In de Progress omgeving stond het menu echter niet in de lijst met beschikbare ActiveX objecten. Ik heb toen geprobeerd het object aan te roepen vanuit Progress code, door een functie te schrijven waarmee enkele methoden van het object werden opgevraagd. Dit lukte wel: ik kon bijvoorbeeld vanuit

Progress de methode uitvoeren waarmee in het object verbinding met de Progress AppServer wordt gemaakt. Het leek er dus op dat het object wel was geregistreerd in het Windows register.
Het feit dat het object in Progress niet in de lijst met beschikbare ActiveX objecten voorkwam, duidde er echter op dat de registratie van het object niet geheel correct was verlopen. Om de oorzaak hiervan te achterhalen, heb ik op Internet gezocht naar een verklaring of oplossing. Ook is het probleem gemeld bij de online Technical Support van Progress.
Na enkele dagen zoeken naar oplossingen en na het proberen van meerdere suggesties, bleek dat er een verschil bestaat tussen de registratie van een .NET object als een “Automation object” en als een “COM control”. Een object dat als “Automation object” is geregistreerd, kan alleen via code worden benaderd en heeft geen grafische gebruikersinterface. Een “COM control” is echter een grafisch object, dat kan worden benaderd via de grafische gebruikersinterface.

Omdat de functionaliteiten van het menu wel via Progress code aan te spreken waren, leek het er dus op dat het object als “Automation object” was geregistreerd. Het menu is echter een grafisch object, waardoor het als “COM control” geregistreerd zou moeten worden. Toen dat bekend was, heb ik onderzoek verricht naar de wijze waarop het object als “COM control” geregistreerd zou kunnen worden.
Omdat de documentatie van Visual Studio .NET 2003 hier niets over vermeldde, ben ik op Internet gaan zoeken. Hieruit bleek dat Microsoft het registreren van objecten als “COM control” niet meer ondersteunt in .NET.
Enkele gebruikers die tegen hetzelfde probleem aanliepen, boden een stuk C# code aan als “workaround”, waarmee tijdens de compilatie van het object de juiste waarden in het Windows register worden toegevoegd, zodat het object wordt geregistreerd als COM control.
Na implementatie van deze code bleek het object inderdaad geregistreerd als COM control: in Progress stond het menu in de lijst met beschikbare ActiveX objecten. Wanneer ik echter het menu uit deze lijst selecteerde, liep de Progress ontwikkelomgeving vast of werd deze zonder melding afgesloten.
Progress bleek dus nog niet met het object overweg te kunnen. Dit probleem is ook voorgelegd bij de Technical Support van Progress. Toen echter bekend werd dat Microsoft het registreren van .NET objecten als “COM controls” niet meer ondersteunt, meldde de Technical Support dat Progress daarom ook geen ondersteuning kon bieden voor dit probleem:

> “The solution you found to get the .NET control registered as a COM control fixes only a part of the issue - the generated COM object will look like a control on the outside, but no more than that. It will still lack any internal support code the COM object requires to actually function as a control instead of an Automation object - and therefore it can't be expected to work.
>
> To get the .NET control to actually work as a COM control you'd need to know what additional code would be required.
> Unfortunately, only Microsoft has this information and as they do not support generating a COM Control from a .NET Control they are unlikely to assist with this.”

Om uit te sluiten dat het probleem niet veroorzaakt werd door een stuk code dat ik zelf had geschreven, heb ik het voorbeeld geïmplementeerd dat ik als “workaround” had gevonden op Internet. In dit voorbeeld werd een .NET UserControl zonder functionaliteiten gecompileerd en geregistreerd als COM control. Ook dit voorbeeld

bleek echter niet in Progress te kunnen worden ingevoegd: de ontwikkelomgeving liep vast of werd afgesloten.

Inmiddels was de beschikbare tijd voor het project vrijwel verstreken. In overleg met de begeleider is toen besloten om de documentatie voor de pilot bij te werken met de kennis die was opgedaan tijdens het project. Ondertussen zou nog onderzoek kunnen worden verricht of navraag kunnen worden gedaan naar een oplossing voor het bovenstaande probleem.
Ik heb bouweenheid 7 afgerond door de testspecificaties voor deze bouweenheid uit te voeren. Ondanks dat het menu niet in te voegen was in Progress, is (voor zover bekend) de registratie van het object wel correct verlopen. De testgevallen zijn dus ook succesvol doorlopen.

8.7.8 Menu als ActiveX object invoegen in Progress

Deze bouweenheid is deels gelijktijdig uitgevoerd met de implementatie van bouweenheid 7. Het invoegen van het menu als ActiveX object mislukte echter. In het pilotontwikkelrapport heb ik de negatieve uitkomst van de testgevallen voor deze bouweenheid beschreven.

8.7.9 Ontwikkelen mogelijkheid opstarten van programma's

Omdat de implementatie van de voorgaande bouweenheid is mislukt, is deze bouweenheid niet geïmplementeerd.

8.8 Invoeringsprocedure

De laatste activiteiten die ik heb uitgevoerd in het kader van het project, omvatten het bijwerken van de documentatie bij de pilot. Een van deze activiteiten bestond uit het opstellen van een invoeringsprocedure voor de pilot. Deze procedure heb ik op dezelfde wijze opgesteld als de invoeringsprocedure voor de eerste pilot. Na een analyse van de activiteiten die benodigd zijn om het menu in te kunnen voeren, heb ik het volgende stappenplan opgesteld.

1. Controleren aanwezigheid benodigde hard- en software:
   - Ontwikkelomgeving Progress (OpenEdge 10);
   - Toegang tot de broncode van de applicatie XL-Enz;
   - Toegang tot de database waar XL-Enz gebruik van maakt;
   - Microsoft .NET Framework (versie 1.1 of hoger) is geïnstalleerd op de client;
   - Het menu in de vorm van een ActiveX (DLL) object;
   - Het gegenereerde proxy-object;
   - Progress applicatie waarin het menu is geplaatst.
2. Installeren Progress functie waarmee menu-data wordt opgehaald uit de database;
3. Plaatsen proxy-object en benodigde assembly bestanden in de juiste directory. Voor gedetailleerde informatie hierover wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET";
4. Vervangen huidige menu van XL-Enz met nieuw menu.

Vervolgens heb ik testspecificaties opgesteld die ter controle op een correcte invoering kunnen dienen.

8.9 Integreren bouweenheden

Omdat nog niet alle bouweenheden waren geïmplementeerd, was het integreren van de bouweenheden tot een samenwerkend geheel niet zinvol. Om deze reden heb ik deze activiteit uitgesteld tot het moment dat alle bouweenheden zijn geïmplementeerd. Deze activiteit is dan ook niet meer uitgevoerd binnen de doorlooptijd van het afstudeerproject.

8.10 Beoordelen en testen

In de activiteit “Beoordelen en testen” heb ik in samenwerking met de begeleider van het project de pilot beoordeeld voor zover deze is ontwikkeld. Na de testresultaten van de bouweenheden te hebben beschreven en doorgenomen, is op basis van deze resultaten een beoordeling van de pilot opgesteld.
Daarbij is gecontroleerd of de pilot na de eerste iteratieslag voldoet aan de eisen die in de definitiestudie opgesteld zijn. Hieronder is het resultaat van deze controle opgenomen.

ID	Omschrijving	Voldaan
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma’s van XL-Enz kunnen worden opgestart	Nee, mogelijkheid programma’s opstarten is nog niet geïmplementeerd
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer	Ja
MF3	Een gebruiker kan het menu afsluiten	Ja
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database	Ja
MN2	Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa	Ja
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven	Ja
MN4	Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd	Nee, invoegen mislukt
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress dataobject en vervolgens doorgegeven aan de .NET client, waar het object wordt geconverteerd naar een ADO.NET dataobject	Nee, dataobject wordt lokaal opgebouwd

Figuur 37: *Resultaten eerste iteratieslag pilot 2*

Zoals reeds bekend was, is niet aan alle eisen voldaan. De beoordeling van de pilot zoals die in het pilotontwikkelrapport is opgenomen, is op de volgende pagina te vinden.

Op dit moment is het mogelijk om in een .NET omgeving het menu te initialiseren en te navigeren. De functionaliteit voor het opstarten van programma's kan echter alleen in de Progress omgeving worden geïmplementeerd.
Deze functionaliteit kan echter nog niet worden geïmplementeerd vanwege het feit dat het invoegen van het menu in een Progress applicatie mislukt. Als gevolg hiervan kan de pilot niet als succesvol afgerond worden beschouwd.

Het is nog onbekend wat de reden is waardoor de Progress applicatie vastloopt of wordt afgesloten bij het invoegen van het menu. Op dit moment wordt hiernaar onderzoek verricht. Wanneer de reden bekend is en een oplossing beschikbaar is, kan de ontwikkeling van bouweenheid 8 worden voortgezet door de oplossing te implementeren. Wanneer de geboden oplossing dit vereist, kan de fase pilotontwikkeling in zijn geheel opnieuw worden doorlopen, waarbij de eisen en het ontwerp worden aangepast aan de nieuwe inzichten.
Wanneer er geen oplossing mogelijk blijkt te zijn voor het genoemde probleem, dient met de begeleider en/of opdrachtgever van dit project overleg te worden gepleegd over de verdere ontwikkeling van het menu.

8.11 Afronding

Na het schrijven van een samenvatting van de inhoud van het pilotontwikkelrapport en het controleren van de rapportopmaak, heb ik het rapport opgeleverd. De begeleider heeft het rapport vervolgens doorgenomen en is akkoord gegaan met de inhoud ervan. Het pilotontwikkelrapport is als externe bijlage opgenomen bij dit verslag.

Wegens het uitlopen van de bouw van enkele bouweenheden, heb ik de fase pilotontwikkeling voor deze pilot niet kunnen afronden in de geplande drie weken. Na ruim vijf weken is besloten om de fase pilotontwikkeling voor deze pilot af te ronden, als gevolg van de genoemde problemen met de bouw van bouweenheid 7 en 8.

Ter afsluiting van het project heb ik het rapport over .NET interfacing uitgebreid met de kennis die ik tijdens de ontwikkeling van pilot 2 heb opgedaan. Deze kennis omvatte voornamelijk oplossingen voor problemen waar ik tegenaan was gelopen tijdens de bouw van het menu en het beschrijven van stap 4 "Invoeren .NET client applicatie " uit het stappenplan (zie paragraaf 5.4.5).
Na oplevering van het rapport over .NET interfacing, is de afstudeerperiode afgerond.

9. Evaluatie

9.1 Inleiding

In dit hoofdstuk wordt het uitgevoerde project geëvalueerd. Na in paragraaf 9.2 het verloop van elke fase uit het project te hebben beoordeeld, worden in paragraaf 9.3 de opgeleverde producten geëvalueerd.

9.2 Procesevaluatie

9.2.1 Opstellen plan van aanpak

Tijdens de opleiding Informatica & Informatiekunde heb ik meerdere malen een plan van aanpak voor het uitvoeren van projecten op moeten stellen. Deze ervaring heb ik kunnen gebruiken voor het opstellen van het plan van aanpak voor dit project. Tijdens de module voorbereidend op het afstuderen is zeer veel aandacht besteed aan het opstellen van een zo compleet mogelijk plan van aanpak. Het plan van aanpak dat daaruit voortkwam, heb ik als uitgangspunt gebruikt bij het opstellen van het plan van aanpak voor dit project. Dit bleek een waardevol houvast te bieden om binnen de geplande tijd een eerste versie van het document te kunnen opleveren. Het opleveren van een definitieve versie bleek na bespreking met de begeleider een kwestie van het aanpassen of uitbreiden van enkele kleine onderdelen uit het document.
Het opstellen van het document heeft geen problemen opgeleverd en is binnen de geplande periode uitgevoerd. In een volgend project zou ik het plan van aanpak op dezelfde wijze opstellen als dat ik dat tijdens dit project heb gedaan.

9.2.2 Onderzoek verrichten naar interfacing

In de fase “Onderzoek verrichten naar interfacing” heb ik zeer veel nieuwe kennis opgedaan. Bij aanvang van het project was ik onbekend met Progress en Crystal Reports en had ik minimale kennis van .NET.
Het uitvoeren van het stappenplan dat de begeleider had opgesteld om kennis op te doen van Progress heeft een groot deel van de kennis opgeleverd die ik benodigd had tijdens het bouwen van de interfacing. Vooral het leren werken met TempTables en ProDataSets heeft zijn vruchten afgeworpen tijdens de fase pilotontwikkeling.

In de fase “Onderzoek verrichten naar interfacing” was ook een activiteit gepland waarin kennis diende te worden opgedaan van Crystal Reports. In overleg is echter besloten deze activiteit niet in deze fase uit te voeren, omdat de kennis van Crystal Reports die benodigd was om de opdracht uit te kunnen voeren, minimaal was. Voor de opdracht was alleen kennis benodigd op het gebied van het inlezen van rapporten. Deze kennis heb ik in de fase pilotontwikkeling opgedaan. Het uitstellen van deze activiteit heeft geen nadelige gevolgen gehad voor het project.

Het onderzoek verrichten naar .NET heeft mij veel inzicht opgeleverd in de werking en het doel van .NET. Bij aanvang van het project kon ik me nog weinig voorstellen bij .NET. Het onderzoek naar de definitie, technische werking en toekomst van .NET heeft mij inzicht gegeven in het doel en de reikwijdte van .NET.

Het verwoorden van de opgedane kennis van .NET in het rapport over interfacing was moeilijk. De overvloed aan informatie, de verschillende meningen en de complexiteit van .NET maakte het niet eenvoudig om een compact verhaal te schrijven waarin de hoofdgedachte van .NET en de overheersende meningen daarover werden verwoord. Toch ben ik wel tevreden over het resultaat. Nadat enkele medewerkers het eerste hoofdstuk van het rapport hadden gelezen, bleek dat .NET voor hen ook niet langer een vage term was.

Het is een goed besluit geweest om het laatste hoofdstuk van het rapport over interfacing open te laten gedurende het project om later opgedane kennis alsnog toe te kunnen voegen. Vooral tijdens de bouw van de pilots deed ik veel kennis op van de technische implementatie van interfacing tussen Progress en .NET. Hierbij kwamen veel details aan het licht, die waardevol bleken om te vermelden in het rapport.
Een nadeel van het open laten van het rapport was dat het rapport veel later werd opgeleverd dan aanvankelijk gepland. De medewerkers van Reflecta werden daardoor pas op een later tijdstip in staat gesteld kennis op te doen van (interfacing naar) .NET.

Ik ben tevreden over de wijze waarop ik het onderzoek naar interfacing heb verricht. Door van tevoren overeenstemming te bereiken over de te onderzoeken onderdelen, kon ik het onderzoek gericht uitvoeren. Hierdoor is geen tijd verloren gegaan met het verrichten van onderzoek naar irrelevante onderwerpen.

9.2.3 Fase definitiestudie

Voordat ik het IAD ontwikkeltraject startte, heb ik onderzoek verricht naar de activiteiten die kunnen worden verricht tijdens fase definitiestudie. Uit dit onderzoek bleek dat enkele activiteiten die ik aanvankelijk relevant achtte, dat niet bleken te zijn (bijvoorbeeld “Beschouwen organisatorische inrichting”). Hierdoor kon ik de aanvankelijk geplande doorlooptijd van de fase definitiestudie met een week verkorten.
Daarnaast heeft het feit dat de systeemeisen reeds vanaf het begin van het project vrijwel geheel bekend waren, bijgedragen aan een snelle doorloop van de fase definitiestudie.
Het onderzoeken hoe de technieken van UML in de fase definitiestudie konden worden toegepast, kostte echter veel tijd. Uit het onderzoek bleek dat het klassedigram, de use-cases en het use-case diagram in de definitiefase konden worden toegepast. Deze diagrammen bleken echter niet toepasbaar op het systeemontwerp voor het dataobject voor Crystal Reports. De UML diagrammen modelleren namelijk de acties die gebruikers met het systeem kunnen uitvoeren. Het dataobject voor Crystal Reports is echter geen gebruikersgericht systeem. Om deze reden heb ik ervoor gekozen een tekstueel systeemontwerp te maken.
Het menu van XL-Enz kon ik wel met de genoemde UML diagrammen modelleren, omdat dit systeem wel gebruikersgericht is.

Tijdens de fase definitiestudie heb ik geleerd hoe een definitiestudie wordt uitgevoerd in een IAD ontwikkeltraject. Ik was reeds bekend met het uitvoeren van een definitiestudie in een SDM ontwikkeltraject. De definitiestudie in IAD bleek hetzelfde doel te hebben. Enkele activiteiten die binnen IAD worden uitgevoerd, verschillen echter met SDM. Zo wordt in de IAD methode een pilotontwikkelplan opgesteld, wat bij de SDM methode niet plaatsvindt.

Ik ben tevreden over de wijze waarop ik de fase definitiestudie heb uitgevoerd. In een volgend project waar de ontwikkelmethode IAD wordt gebruikt, zou ik deze fase op dezelfde wijze uitvoeren als dat ik dat voor dit project heb gedaan.

9.2.4 Pilotontwikkeling pilot 1

Voor aanvang van de fase pilotontwikkeling heb ik wederom een onderzoek verricht naar de activiteiten die volgens IAD kunnen worden verricht in deze fase. Hierdoor kon ik ook weer enkele activiteiten die ik aanvankelijk had ingepland, achterwege laten omdat deze niet relevant bleken voor dit project.
Na bestudering van de activiteiten kon ik de planning voor de fase pilotontwikkeling beter inschatten doordat ik meer inzicht had in de duur van de activiteiten.
Het onderzoek dat ik heb verricht naar hoe de technieken van UML konden worden toegepast bij het ontwerpen van de functionele en technische structuur van de eerste pilot, kostte veel tijd. Uiteindelijk heb ik de technieken van UML slechts beperkt toegepast bij het ontwerp. Het grotendeels tekstuele ontwerp van de bouweenheden voldeed echter wel om de pilot te kunnen bouwen.

In deze fase heb ik geleerd hoe de pilotontwikkeling binnen IAD wordt doorlopen. De iteratieve wijze van ontwikkeling bleek goed toepasbaar op dit project. De verdeling in iteraties zoals die gemaakt is voor de eerste pilot (zie paragraaf 7.5) heeft de doorlooptijd van de pilot versneld.
Naast de leereffecten op het gebied van het gebruik van IAD en UML, heb ik in deze fase zeer veel geleerd op het gebied van de Progress 4GL, (ADO.NET) XML en Crystal Reports. Met de programmeerervaring die ik heb opgedaan tijdens de opleiding Informatica & Informatiekunde bleek het aanleren van een onbekende taal als Progress geen problemen op te leveren.
Ik ben tevreden over de wijze waarop ik de fase pilotontwikkeling voor de eerste pilot heb uitgevoerd. In een volgend project waar de ontwikkelmethode IAD wordt gebruikt, zou ik deze fase op dezelfde wijze uitvoeren als dat ik dat voor dit project heb gedaan.

9.2.5 Pilotontwikkeling pilot 2

Met de ervaring die ik had opgedaan tijdens de fase pilotontwikkeling voor de eerste pilot, kon ik de fase pilotontwikkeling voor de tweede pilot in enkele opzichten sneller uitvoeren. Zo had ik bijvoorbeeld meer inzicht in de uit te voeren activiteiten en de duur daarvan. Hierdoor kon ik de activiteiten voor deze fase beter plannen.
De technieken van UML heb ik ook bij het ontwerp van deze pilot slechts beperkt gebruikt. De functionaliteiten van het menu waren naar mijn mening na de modellering van de sequencediagrammen reeds voldoende gemodelleerd. Mede op basis van dit ontwerp kon ik de bouweenheden implementeren. Achteraf heeft het achterwege laten van het ontwerpen van bijvoorbeeld toestandsdiagrammen en componentdiagrammen geen problemen opgeleverd.

Naast de leereffecten op het gebied van het gebruik van IAD en UML, heb ik in deze fase zeer veel geleerd op het gebied van het ontwikkelen van .NET applicaties in Visual Studio .NET 2003 en de wijze waarop Progress interfacing met .NET realiseert. Visual Studio .NET 2003 bleek een zeer complexe, maar ook erg krachtige ontwikkelomgeving te zijn. De complexiteit van deze omgeving maakte het echter niet eenvoudig om snel met deze omgeving overweg te kunnen.

Bij het aanleren van de taal C# heb ik gebruik kunnen maken van de kennis van Java en C die ik heb opgedaan tijdens de opleiding Informatica & Informatiekunde. C# bleek veel overeenkomsten te hebben met deze talen, waardoor het aanleren van de taal relatief snel verliep. Wel was het moeilijk om uit de vele mogelijkheden van de taal de juiste functionaliteiten te vinden die benodigd waren om het menu te bouwen. De documentatie van Visual Studio .NET 2003 bleek hierbij een waardevolle bron van informatie.

De problemen die ontstonden bij de implementatie van bouweenheid 7 en 8 van de pilot, waren van tevoren niet te voorzien. Binnen Reflecta zijn er vaker ActiveX objecten die waren ontwikkeld in een andere omgeving dan Progress succesvol ingevoegd in een Progress omgeving. Er was dus geen reden om te verwachten dat het invoegen van het menu als ActiveX object problemen zou opleveren. Daarnaast bleek het probleem bij de Technical Support van Progress ook nog onbekend te zijn toen het werd gemeld.

Ik ben niet geheel tevreden over de wijze waarop ik de fase pilotontwikkeling voor de tweede pilot heb uitgevoerd. In een volgend project waarbij nieuwe technieken worden gebruikt, zou ik eerst een klein testprogramma willen ontwikkelen, waarmee kan worden getest of de te ontwikkelen oplossing technisch haalbaar is. Daarmee zou het probleem dat bij de implementatie van bouweenheid 7 en 8 van deze pilot is opgetreden, in een eerder stadium kunnen worden ontdekt.

Eén van de leereffecten van dit project is dat het gebruik van nieuwe technieken zeer risicovol kan zijn en onvoorziene problemen kan opleveren. Deze risico's zouden kunnen worden verkleind door bij aanvang van het project een verkenningstraject te doorlopen of een kleine testimplementatie te verrichten.

9.3 Productevaluatie

9.3.1 Plan van aanpak

Het doel van het plan van aanpak, overeenstemming bereiken over de opdracht en over de uitvoering daarvan, is bereikt. Na oplevering van de definitieve versie van het document is de begeleider namelijk akkoord gegaan met de inhoud ervan.
Het document heeft gedurende het project zijn nut op meerdere punten bewezen. Vooral de planning van het project bleek een waardevol houvast om het project te kunnen beheersen. De schattingen van de benodigde tijd per fase bleken in de praktijk vrij dichtbij de daadwerkelijk benodigde tijd te liggen. Het gebruik van het genoemde planningsalgoritme zal daar aan bijgedragen hebben. Alleen de fase pilotontwikkeling is uitgelopen op de oorspronkelijke planning wegens technische problemen die van tevoren niet waren te voorzien.
Daarnaast heeft een vrij uitgebreide beschrijving van de standaarden en richtlijnen met betrekking tot de rapportopmaak bijgedragen aan een consistente opmaak van de opgeleverde documentatie.
De opgestelde risicofactoren zijn allen niet opgetreden. Het feit dat het project echter niet is afgerond binnen de geplande periode, werpt de vraag op of de opgestelde risicofactoren wel volledig zijn geweest. In een volgend project zou ik dan ook een risicofactor opnemen die het risico van het gebruik van nieuwe technieken benoemt.
Omdat het plan van aanpak tijdens dit project zijn waarde heeft bewezen, zou ik dit plan

van aanpak als uitgangspunt kunnen gebruiken bij het opstellen van een dergelijk document voor volgende projecten.

9.3.2 Rapport .NET interfacing

Het doel van het schrijven van een rapport over interfacing tussen Progress en .NET was de medewerkers van Reflecta kennis op te laten doen van .NET en van de wijze waarop interfacing tussen Progress en .NET kon worden geïmplementeerd. Daarnaast wilde men kennis opdoen van de mogelijkheden van .NET voor Reflecta.
Deze doelen zijn bereikt doordat het rapport onder meer de volgende onderdelen bevat:

- Een definitie en een beschrijving van de technische werking van .NET;
- Een overzicht van de mogelijkheden van .NET voor Reflecta;
- Een beschrijving van de technische implementatie van interfacing tussen Progress en .NET aan de hand van voorbeeldimplementaties.

De opmaak van het rapport voldoet aan de eisen die daarvoor zijn gesteld in het plan van aanpak.

9.3.3 Rapport definitiestudie

Het rapport definitiestudie bevat de resultaten van de activiteiten die zijn uitgevoerd tijdens de fase definitiestudie. Het doel van het rapport was om de eisen aan de te ontwikkelen software vast te stellen en een basis te vormen voor de ontwikkeling van deze software.
Deze doelen zijn bereikt doordat het rapport onder meer de volgende onderdelen bevat:

- De eisen aan de te ontwikkelen software. Bij de beoordeling van deze pilot is gecontroleerd of de pilot aan deze eisen voldoet;
- Een systeemconcept dat als uitgangspunt heeft kunnen dienen voor het ontwerp dat tijdens de fase pilotontwikkeling is gemaakt;
- Een verdeling van de fase pilotontwikkeling in meerdere pilots. Deze verdeling is aangehouden tijdens het project en heeft als uitgangspunt gefungeerd voor het verdelen van de pilotdelen in bouweenheden.

De opmaak van het rapport voldoet aan de eisen die daarvoor zijn gesteld in het plan van aanpak.

9.3.4 Pilotontwikkelrapport 1

Het eerste pilotontwikkelrapport had als doel om ondersteuning te bieden bij de bouw van het dataobject voor Crystal Reports.
Dit doel is bereikt doordat het rapport onder meer de volgende onderdelen bevat:

- Een ontwerp van de bouweenheden op basis waarvan de bouweenheden zijn geïmplementeerd;
- Een pilotontwikkelplan waarmee de bouw van de pilot is gestructureerd;
- Testspecificaties waarmee de geïmplementeerde bouweenheden zijn getest en beoordeeld.

De opmaak van het rapport voldoet aan de eisen die daarvoor zijn gesteld in het plan van aanpak.

9.3.5 Pilotontwikkelrapport 2

Het tweede pilotontwikkelrapport had als doel om ondersteuning te bieden bij de bouw van het menu voor XL-Enz. De opbouw van dit rapport is gelijk aan de opbouw van het eerste pilotontwikkelrapport en heeft daarmee ook zijn doel bereikt.
De opmaak van het rapport voldoet ook aan de eisen die daarvoor zijn gesteld in het plan van aanpak.

9.3.6 Dataobject Crystal Reports

Zoals vermeld in paragraaf 7.10 voldoet het opgeleverde dataobject voor Crystal Reports niet aan alle eisen die zijn opgesteld tijdens de definitiestudie. Technisch is het dataobject in de vorm van een ADO.NET XML document bruikbaar als datasource voor Crystal Reports. De slechte performance van het opbouwen van een rapport op basis van deze datasource maakt het dataobject in de praktijk echter onbruikbaar.
De slechte performance wordt waarschijnlijk veroorzaakt doordat de data in het ADO.NET XML document niet geïndexeerd is opgeslagen. Doordat ADO.NET XML geen indexering ondersteunt, lijkt het performanceprobleem momenteel niet oplosbaar.

9.3.7 Menu XL-Enz

Ook het ontwikkelde menu van XL-Enz voldoet niet aan alle eisen die zijn opgesteld tijdens de definitiestudie (zie paragraaf 8.10). Doordat het technisch niet mogelijk blijkt te zijn om het menu als ActiveX object in te voegen in Progress, kan het menu niet als succesvol opgeleverd worden beschouwd.

Geraadpleegde literatuur

Hieronder is in alfabetische volgorde een overzicht opgenomen van de geraadpleegde literatuur. Voor een overzicht van de bronnen die zijn geraadpleegd tijdens de fase “Onderzoek verrichten naar interfacing” wordt verwezen naar het deel “Geraadpleegde literatuur” achterin het rapport “Interfacing tussen Progress en Microsoft .NET” dat als externe bijlage bij dit verslag is opgenomen.

- Ghezzi, C., e.a., Fundamentals of Software Engineering, Prentice-Hall, januari 1991;
- Online documentatie over Progress, Crystal Reports en Microsoft .NET;
  Progress : http://www.progress.com
  Crystal Reports : http://www.businessobjects.com
  Microsoft .NET : http://www.msdn.microsoft.com
- Reader AV-03 Haagse Hogeschool, nr. 908: Rapportagetechniek;
- Tolido, R.J.H., IAD - Het evolutionair ontwikkelen van informatiesystemen, Academic Service, 2e dr., 1998;
- Warmer, J., en Kleppe A., Praktisch UML, Addison Wesley, 2e editie, 2001.

Bijlagen

Vanwege de omvang zijn de bijlagen in een aparte bundel opgenomen.
Deze bundel bevat de volgende bijlagen:

1. Plan van aanpak;
2. Rapport “Interfacing tussen Progress en Microsoft .NET”;
3. Rapport definitiestudie;
4. Pilotontwikkelrapport pilot 1: Dataobject Crystal Reports;
5. Pilotontwikkelrapport pilot 2: “Must-Have” functionaliteiten menu XL-Enz.

Bijlagen
Project .NET interfacing

Student
A. Schlingmann, 99004010
Opleiding
Haagse Hogeschool
Informatica & Informatiekunde
Opleidingsvariant
Ontwikkeling van Software en Technische Infrastructuren (OSTI)
Afstudeerperiode
17 november 2003 t/m 26 maart 2004
Organisatie
Reflecta Automation B.V.
Dorpsweg 26
2811 KH Reeuwijk-Dorp
Tel: 0182-398130
Fax: 0182-398163
Email: info@reflecta.nl
Bedrijfsmentoren
Dhr. T. Dekker
Dhr. G.A. Lenselink
Examinatoren
Dhr. E.M. van Doorn
Dhr. J.L.A. Schramp

Voorwoord

Deze bundel bevat de bijlagen bij het eindverslag dat ik heb geschreven naar aanleiding van mijn afstudeeropdracht die ik heb uitgevoerd ter afsluiting van de opleiding Informatica & Informatiekunde aan de Haagse Hogeschool. De opdracht is uitgevoerd bij softwarehuis Reflecta Automation B.V. te Reeuwijk-Dorp.

De bijlagen bestaan uit de documentatie die ik heb opgeleverd gedurende de uitvoering van de afstudeeropdracht.

Reeuwijk-Dorp, 26 maart 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

Bijlage 1

Plan van aanpak

Voorwoord

Voor u ligt het plan van aanpak voor het project .NET interfacing. Dit project wordt uitgevoerd in opdracht van Reflecta Automation B.V. te Reeuwijk-Dorp en omvat het verrichten van onderzoek naar en de bouw van interfacing tussen enerzijds Progress en Microsoft .NET en anderzijds tussen Progress, Microsoft .NET en Crystal Reports.

In dit rapport wordt het project omschreven en de aanpak vastgesteld.
Het document is tot stand gekomen in overleg met dhr. T. Dekker, programmeur bij Reflecta Automation.

Reeuwijk-Dorp, 9 december 2003

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

1. Inleiding

Reflecta heeft een project gestart waarin onderzoek wordt verricht naar de werking van interfacing tussen de 4GL Progress en Microsoft .NET. Op basis van de resultaten van dit onderzoek dient vervolgens in een .NET omgeving een menu te worden ontwikkeld voor de applicatie XL-Enz. Daarnaast omvat dit project het implementeren van een .NET dataobject dat als datasource kan dienen voor Crystal Reports.

De aanleiding voor de uitvoering van het project is de behoefte van Reflecta om .NET technologieën te integreren in de applicatie XL-Enz, een handelsapplicatie dat Reflecta in eigen beheer ontwikkelt.

Het doel van dit rapport is om de inhoud en de aanpak van het project te definiëren.

De opbouw van dit rapport is als volgt. Als eerste wordt in hoofdstuk 2 de opdracht omschreven. Deze wordt verder uitgewerkt in de volgende hoofdstukken door de projectorganisatie vast te stellen en projectrisico's te onderkennen. Als laatste wordt in de hoofdstukken 5 tot en met 7 de aanpak van het project beschreven.

2. Opdrachtomschrijving

2.1 Inleiding

Dit hoofdstuk beschrijft de inhoud van het project ".NET interfacing". Hierbij wordt ingegaan op het ontstaan van de opdracht en wordt de probleemstelling beschreven. Vervolgens wordt het doel van de opdracht gedefinieerd en de aanpak vastgesteld.

2.2 Opdrachtgever

De opdrachtgever van het project is dhr. G.A. Lenselink, directeur van Reflecta Automation B.V. te Reeuwijk-Dorp. Reflecta is een softwarehuis dat sinds 1990 werkzaam is binnen de automatisering van groothandelaren, importeurs en detailhandelsbedrijven in de sport-, mode-, schoenen- en tweewielerbranche.

2.3 Aanleiding opdracht

Reflecta heeft in eigen beheer het softwarepakket RA-Trade ontwikkeld. RA-Trade is een handelsapplicatie waarin specifieke toepassingen zijn opgenomen voor organisaties die actief zijn in bovengenoemde branches. Hieronder vallen functionaliteiten die benodigd zijn voor het registreren van de beweging van goederen. In grote lijnen betreft dit relatiebeheer, inkoop, verkoop, voorraadbeheer en de financiële administratie. RA-Trade is geschreven in de 4GL Progress en heeft een character based gebruikersinterface. Op dit moment is Reflecta bezig met het ontwikkelen van een opvolger van RA-Trade, die de naam XL-Enz heeft gekregen. Deze nieuwe applicatie heeft een grafische gebruikersinterface.

De nieuwste versie van Progress (OpenEdge 10) zal volledige ondersteuning bieden met betrekking tot een koppeling met Microsoft .NET technologieën. Reflecta verwacht dat deze nieuwe technologieën in de toekomst een steeds belangrijkere rol gaan spelen.

2.4 Probleemstelling

Op dit moment bezit Reflecta geen kennis van de technologieën van .NET. De opdrachtgever wil deze technologieën echter wel gaan integreren in XL-Enz, om zodoende een voorsprong op de concurrentie te behalen.

In XL-Enz wordt momenteel het menu-dataobject dynamisch opgebouwd door Progress. Op basis van dit dataobject wordt de gebruikersinterface van het menu door Progress opgebouwd. Deze gebruikersinterface is in de ogen van de opdrachtgever niet erg aantrekkelijk en hij zou deze dan ook anders willen.

Reflecta maakt gebruik van de reporting tool Crystal Reports, waarmee gegevens uit databases worden gelezen en vervolgens in een rapport worden gepresenteerd. Momenteel wordt het dataobject dat als datasource dient voor Crystal Reports op een inefficiënte wijze opgebouwd. Reflecta wil de opbouw van dit dataobject verbeteren door gebruik te maken van .NET technologieën.

2.5 Omschrijving opdracht

De opdracht omvat enerzijds het verrichten van een onderzoek naar de werking van interfacing tussen Progress en .NET. Deze kennis dient zodanig te worden gedocumenteerd dat deze bruikbaar is voor de ontwikkelaars van Reflecta om .NET te kunnen integreren in XL-Enz.
De volgende punten dienen te worden onderzocht:

- Algemene werking van .NET (definitie, doel, (on)mogelijkheden, voorbeelden);
- Mogelijke betekenis van .NET voor Reflecta;
- Werking van interfacing tussen Progress en .NET;
- Werking van interfacing tussen Crystal Reports en .NET.

Anderzijds omvat de opdracht het doorlopen van een systeemontwikkeltraject, met als doelen:

- In een .NET-omgeving een data- en gebruikersinterface maken voor het menu van XL-Enz. De data-interface bestaat uit een menu-dataobject, dat een dynamisch opgebouwde menustructuur kan inlezen. De gebruikersinterface wordt opgebouwd op basis van het menu-dataobject en dient een vergelijkbaar uiterlijk te hebben als de Windows Explorer: aan de linkerkant staan de hoofditems van het menu in een boomstructuur en aan de rechterkant staat de inhoud van de hoofditems weergegeven. Wanneer een item uit het menu aangeklikt wordt, dient de achterliggende functionaliteit te worden uitgevoerd. Het menu dient ook functionaliteiten als knippen, slepen en plakken van menu-items te bevatten, waardoor de gebruiker in staat wordt gesteld de opbouw van het menu naar eigen inzicht samen te stellen.
  Wijzigingen in de opbouw van het menu dienen te worden opgeslagen in de Progress database.

- Crystal Reports laten werken op basis van een .NET dataobject als datasource. Met behulp van de laatste Progress techniek, de ProDataSet, dient vanuit Progress een ADO.NET dataobject te worden gecreëerd, dat vervolgens als datasource kan worden gebruikt om rapporten op te bouwen, te tonen en te printen.

2.6 Doelstellingen

Het doel van de opdracht is drieledig:

- Kennis opdoen voor Reflecta over de mogelijkheden van interfacing tussen enerzijds Progress en .NET en anderzijds Crystal Reports en .NET. Deze kennis dient zodanig te worden gedocumenteerd dat deze bruikbaar is voor de ontwikkelaars van Reflecta om .NET te kunnen integreren in XL-Enz;
- Een data- en gebruikersinterface voor XL-Enz opleveren die aan bovenstaande eisen voldoen;
- Crystal Reports laten werken op basis van een .NET dataobject als datasource.

2.7 Aanpak van de opdracht

Het te verrichten onderzoek wordt uitgevoerd door het opstellen van een rapport waarin de eerder genoemde onderwerpen worden beschreven.
Het ontwerpen, implementeren en invoeren van het menu van XL-Enz en de datasource van Crystal Reports wordt aangepakt volgens de systeemontwikkelmethode Iterative Application Design (IAD). Hierbij zal gebruik worden gemaakt van de technieken van Unified Modeling Language (UML).

2.8 Afbakeningen

De volgende afbakeningen zijn vastgesteld:

- De doorlooptijd van het project kan niet worden aangepast. Het project wordt uitgevoerd in een periode van 19 weken;
- Bij de op te leveren software worden geen handleidingen opgeleverd;
- Het samenstellen van opleidingsmateriaal is geen onderdeel van de opdracht.

2.9 Op te leveren producten

De volgende producten worden opgeleverd:

- Document plan van aanpak
  Hierin wordt het project omschreven en gepland;
- Rapport .NET interfacing
  Dit rapport bevat een beschrijving van de werking van interfacing tussen Progress en .NET;
- Document definitiestudie
  Hierin worden de systeemeisen geanalyseerd en wordt een systeemconcept opgesteld. In een pilotplan wordt het systeem opgedeeld in pilots;
- Document pilotontwikkelrapport (per pilot)
  Hierin wordt de functionele en technische structuur van de pilot ontworpen;
- Menu-dataobject van XL-Enz met bijbehorende gebruikersinterface, dynamisch opgebouwd door middel van .NET-technologieën;
- Datasource van Crystal Reports gekoppeld aan .NET;
- Beheerdocumentatie.

3. Projectorganisatie

3.1 Inleiding

Dit hoofdstuk beschrijft de samenstelling van de projectgroep en de faciliteiten en hulpmiddelen die benodigd zijn om het project te kunnen uitvoeren. Als laatste wordt beschreven wat tijdens het project wordt verwacht van de opdrachtgever en de begeleider van het project.

3.2 Samenstelling

Het project wordt uitgevoerd als afstudeeropdracht door dhr. A. Schlingmann, vierdejaars student Informatica & Informatiekunde aan de Haagse Hogeschool.
Sturing van het project wordt verzorgd door dhr. T. Dekker, programmeur bij Reflecta Automation.
De opdrachtgever van het project, dhr. G.A. Lenselink, volgt het verloop van het project.

3.3 Benodigde faciliteiten en hulpmiddelen

Om het project te kunnen uitvoeren, zijn de volgende faciliteiten en hulpmiddelen benodigd:

- Een werkplek bij Reflecta met hardware dat voldoet aan de eisen van Progress, Crystal Reports en Visual Studio .NET 2003;
- Software:
  - Progress ontwikkelomgeving (versie 10 (beta));
  - Crystal Reports;
  - OS: Microsoft Windows XP;
  - Visual Studio .NET 2003;
  - Microsoft .NET Framework;
  - Microsoft Office XP;
  - Microsoft Project;
  - Microsoft Visio 2003.
- Voor het project is geen budget vastgesteld. Het is niet waarschijnlijk dat er kosten gemaakt moeten worden qua aanschaf van software en hardware. Alle benodigde middelen zijn aanwezig. Kosten van begeleiding en sturing door de begeleider zijn ingecalculeerd, maar hiervoor is geen budget vastgesteld;
- Benodigde literatuur:
  - Ghezzi, C., e.a., Fundamentals of Software Engineering, Prentice-Hall, januari 1991;
  - Online documentatie over Progress, Crystal Reports en Microsoft .NET;
  - Reader AV-03 Haagse Hogeschool, nr. 908: Rapportagetechniek;
  - Tolido, R.J.H., IAD Het evolutionair ontwikkelen van informatiesystemen, Academic Service, 2^{e} dr., 1998;
  - Warmer, J., en Kleppe A., Praktisch UML, Addison Wesley, 2^{e} editie, 2001.

3.4 Verwachtingen opdrachtgever en begeleider

Van de opdrachtgever wordt verwacht dat deze betrokken is bij het verloop van het project. Er wordt geen directe sturing van de opdrachtgever verwacht; dit wordt uitgevoerd door de begeleider.
Daarnaast wordt van de opdrachtgever verwacht dat deze bij aanvang van het project de benodigde faciliteiten beschikbaar stelt.

Van de begeleider van het project wordt verwacht dat deze het project stuurt.
Sturing vindt plaats door:

- Het doornemen en becommentariseren van opgeleverde producten;
- Ondersteuning te bieden bij het oplossen van knelpunten;
- Het tijdig melden wanneer in een verkeerde richting wordt gewerkt en sturing in de juiste richting;
- Het juist verwoorden van de functionaliteiten van de te ontwikkelen software.

4. Projectrisico's

4.1 Inleiding

In dit hoofdstuk worden risicofactoren genoemd die het gewenste verloop van het project negatief kunnen beïnvloeden. Voor elke factor wordt een maatregel beschreven die kan worden genomen om kans op het optreden van het risico te verkleinen of de impact van het opgetreden risico te minimaliseren.

4.2 Risicofactoren

- Tijdsvertraging door ziekte of afwezigheid van de uitvoerende van het project.
  Maatregel: Het project inkorten. Deze maatregel is alleen van toepassing bij een afwezigheid gedurende één week of langer. In de planning van het project is een overhead van één week opgenomen.

- Tijdsvertraging door ziekte of afwezigheid van de begeleider van het project.
  Maatregel: Wanneer er niet verder kan worden gewerkt zonder feedback van de afwezige begeleider, wordt er een persoon aangesteld als (tijdelijk) begeleider, die in staat is om de benodigde feedback te geven.

- Het project is te groot in omvang om te kunnen uitvoeren in de beschikbare tijd.
  Maatregel: In overleg met de begeleider en de opdrachtgever de opdracht inkorten.

- Problemen met hardware en/of software.
  Maatregel: Dagelijks back-ups maken van de applicatie en de documentatie.

- De technieken van .NET zijn te complex om binnen de gestelde periode te beheersen.
  Maatregel: Een verkenningstraject naar .NET doorlopen tijdens het onderzoek naar interfacing. Op basis van de bevindingen uit dit onderzoek kan eventueel de opdracht worden aangepast.

5. Standaarden en richtlijnen

5.1 Inleiding

Tijdens het project wordt gebruik gemaakt van een aantal standaarden en richtlijnen. Om het project te beheersen wordt een systeemontwikkelmethode gebruikt in combinatie met enkele technieken. In dit hoofdstuk worden de gekozen methode en technieken beschreven, alsmede de richtlijnen voor de opmaak van de op te leveren rapporten.

5.2 Methoden en technieken

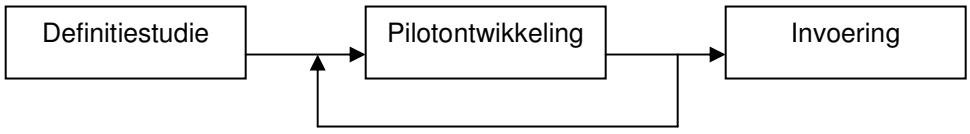
Het rapport over interfacing wordt opgebouwd volgens de rapportagetechnieken die als standaard worden gebruikt door de Sector Informatica van de Haagse Hogeschool.

Het ontwerpen en bouwen van het menu en de koppeling van Crystal Reports met .NET wordt uitgevoerd volgens de systeemontwikkelmethode Iterative Application Design (IAD). Voor deze methode is gekozen vanwege de iteratieve ontwikkelstrategie. Bij aanvang van het project is geen kennis beschikbaar van de technieken van .NET. Tijdens het project zal hierover gaandeweg meer kennis worden opgedaan. Door iteratief te ontwikkelen kan deze opgedane kennis alsnog worden verwerkt in eerder ontwikkelde pilots.

De ontwikkelmethode IAD kent drie fasen:

- Definitiestudie;
- Pilotontwikkeling;
- Invoering.

Als iteratiestrategie is de variant “incrementeel ontwikkelen” gekozen. Hierbij wordt de fase definitiestudie eenmalig doorlopen. De pilotontwikkeling wordt in iteraties uitgevoerd. De invoering van de pilots vindt in één keer plaats.



Voor deze iteratiestrategie is gekozen omdat de eisen aan het op te leveren product reeds voor een groot deel vast staan. Als gevolg hiervan kan worden volstaan met het eenmalig doorlopen van de definitiestudie, waarin de eisen definitief vastgesteld worden.

De keuze om de pilotontwikkeling iteratief te doorlopen, komt voort uit het feit dat alle betrokkenen bij het project onbekend zijn met de technieken van .NET. Door pilots iteratief te ontwikkelen kan later opgedane kennis alsnog worden verwerkt in de pilots.

Binnen de systeemontwikkelmethode wordt gebruik gemaakt van technieken om het systeem te ontwerpen door middel van modellen. De techniek die hierbij gebruikt wordt is de Unified Modeling Language (UML).
De volgende diagrammen van UML worden (waarschijnlijk) gebruikt:

- Use-case-diagram
  Toont hoe het systeem gebruikt kan worden door externe entiteiten, zoals menselijke gebruikers;
- Klassediagram
  Toont de statische structuur van het softwaresysteem, weergegeven als klassen en hun relaties;
- Sequencediagram
  Toont de volgorde in tijd van de boodschappen die in het systeem verstuurd en ontvangen worden;
- Toestandsdiagram
  Toont de toestanden waarin een object zich kan bevinden gedurende zijn levensloop;
- Componentdiagram
  Toont de verdeling van het gehele systeem in componenten en de relaties tussen die componenten.

Het menu en de koppeling tussen Crystal Reports en .NET worden geschreven in de ontwikkelomgevingen Progress OpenEdge 10 en Visual Studio .NET 2003, door middel van de programmeertaal C#.

5.3 Rapportopmaak

De op te leveren rapporten worden opgebouwd volgens de rapportagetechnieken die als standaard worden gebruikt door de Sector Informatica van de Haagse Hogeschool. De checklist uit Bijlage A van dit rapport geldt daarbij als richtlijn.

Daarnaast worden voor de lay-out van de rapporten de volgende standaarden aangehouden:

Item	**Waarde**
Standaard lettertype	Arial 11 pt
Kop 1	Arial 16 pt Bold
Kop 2	Arial 14 pt Bold, Italic
Kop 3	Arial 13 pt Bold
Inhoudsopgave	Arial 10 pt
Koptekst	Linkerkant: projectnaam, Arial 10pt, Italic Rechterkant: documentnaam, Arial 10pt, Italic
Voettekst	Midden: - Paginanummer -, Arial 11pt

6. Planning

6.1 Inleiding

In dit hoofdstuk wordt een planning opgesteld voor het verloop van het project. Als eerste wordt een globale schatting gemaakt van de indeling van het project door een procentuele verdeling op te stellen van de fasen in het project.

6.2 Faseplanning

Hieronder is per fase van het project een schatting gemaakt van het percentage van het totale project dat de fase in beslag zal nemen.

Fase	**% van totale project**
Plan van aanpak	5 %
Verslag interfacing	25 %
Definitiestudie	10 %
Pilotontwikkeling	35 %
Invoering	5 %
Totaal:	**80 %**

Om eventuele uitloop op te kunnen vangen als gevolg van ziekte of uitloop van werkzaamheden is een overhead van 5 % van het totale project gepland.

Naast de activiteiten die benodigd zijn voor het uitvoeren van de opdracht, is het noodzakelijk om tijd in te plannen voor activiteiten die vereist zijn in het kader van het afstudeertraject van de projectuitvoerende. Deze activiteiten bestaan hoofdzakelijk uit het schrijven van een eindverslag en uit activiteiten met betrekking tot voortgangsrapportage naar de examinatoren van de Haagse Hogeschool.
Het percentage van het totale project dat geschat wordt hiervoor nodig te hebben, is 15 %.

6.3 Dagenplanning

Het project wordt uitgevoerd door één persoon, gedurende een doorlooptijd van 19 weken (95 dagen), van 17 november 2003 tot en met 26 maart 2004.
Hiervan zijn drie dagen feestdagen (25 en 26 december 2003, 1 januari 2004), waarop niet aan het project gewerkt wordt. Daarnaast is 2 januari 2004 een verplichte vrije dag. Door omstandigheden start het project niet op 17 november 2003, maar op 19 november 2003.
Het totaal aantal dagen dat aan het project gewerkt wordt, komt hiermee op 89 dagen.

Aan de hand van het beschikbare aantal dagen en de hierboven genoemde procentuele verdeling van de fasen is hieronder een planning gemaakt met daarin het aantal geplande dagen per fase.
Het geplande aantal dagen is berekend met behulp van het min/max-algoritme. De volgende formule is hierbij gebruikt:

gepland # dagen = (minimaal + maximaal + 2 * gemiddeld) / 4.

Fase	# dagen			
	minimaal	**maximaal**	**gemiddeld**	**gepland**
Plan van aanpak	2	4	3	3
Verslag interfacing	15	25	20	21
Definitiestudie	7	12	9	10
Pilotontwikkeling	25	33	30	30
Invoering	3	5	4	4
Overhead	4	5	4	5
Activiteiten afstudeertraject	13	20	17	16
Totaal # dagen:	69	104	87	89

6.4 Detailplanning

In Bijlage B is een gedetailleerd overzicht opgenomen van de planning van het project. In deze planning is tevens het schrijven van het afstudeerverslag opgenomen.

Naast de activiteiten die gepland zijn om het project uit te voeren, zijn de onderstaande activiteiten gepland in het kader van het afstudeertraject van de projectuitvoerende.

Activiteit	**Betrokkenen**	**Gepland**	**# uur**
Bedrijfsbezoek examinatoren (Formaliteiten afstuderen, vaststellen definitieve opdrachtomschrijving)	Projectuitvoerende, begeleider	Tussen 15 – 19 december 2003 of tussen 5 – 9 januari 2004	2 uur
Opleveren definitieve opdrachtomschrijving	Projectuitvoerende	Uiterlijk 16 januari 2004	2 uur
Opsturen voortgangsverslag	Projectuitvoerende	Tussen 19 en 23 januari 2004	2 uur
Opleveren bouwplan afstudeerverslag	Projectuitvoerende	Tussen 23 en 27 februari 2004	2 uur
Gesprek examinatoren (voortgang, afstudeerverslag, inhoud voordracht)	Projectuitvoerende, evt. begeleider	Tussen 1 – 12 maart 2004	2 uur
Opleveren afstudeerverslag	Projectuitvoerende	26 maart 2004, tussen 10:30 en 13:00 uur	2 uur
Afstudeerbespreking	Projectuitvoerende, evt. begeleider	Tussen 13 – 23 april 2004	2 uur

7. Testaanpak en Kwaliteitscontrole

7.1 Inleiding

De ontwikkelde pilots dienen te worden getest op een correcte werking, op het bevatten van de juiste functionaliteiten en op performance. Deze tests worden uitgevoerd voordat de pilots worden onderworpen aan de acceptatietest door de opdrachtgever en/of de begeleider.
De kwaliteit van de op te leveren producten dient te worden gewaarborgd en behoren daarom te worden gecontroleerd. In dit hoofdstuk wordt beschreven hoe het testen en de kwaliteitscontrole plaatsvinden.

7.2 Systeemtest

De ontwikkelde pilotdelen worden getest door middel van de teststrategie “black box testing”. Deze strategie test de werking van het systeem zonder te kijken naar de innerlijke werking van de proceslogica.
Tijdens de fase pilotontwikkeling worden per pilotdeel testspecificaties opgesteld. Deze specificaties dienen alle mogelijke interacties met het systeem te beschrijven door voor elke mogelijke invoer de verwachte uitvoer te specificeren.
De tests worden uitgevoerd in de activiteit “Beoordelen en testen pilot” tijdens de fase pilotontwikkeling.

De pilots worden op onderstaande punten getest:

- Correcte werking
  De werking van de pilot wordt gecontroleerd d.m.v. zogenaamde testinvoer. De applicatie dient hierop te reageren zoals wordt verwacht.

- Functionaliteit
  De pilot wordt getest op de aanwezigheid van de vastgestelde vereiste functionaliteiten.

- Performance
  De wachttijden (response) van de applicatie dienen te voldoen aan de eisen die hiervoor worden opgesteld tijdens de definitiestudie.

7.3 Acceptatietest

Zodra de pilots de systeemtest succesvol hebben doorlopen, kan (na goedkeuring door de begeleider of opdrachtgever) worden overgegaan tot invoering van de pilots.
Na invoering dient de begeleider en/of opdrachtgever op basis van de opdrachtomschrijving een oordeel uit te brengen over de opgeleverde pilots. Dit oordeel wordt gevormd door middel van het uitvoeren van een acceptatietest. Hierbij worden de pilots op de hierboven beschreven punten getest na invoering.
Tijdens de fase pilotontwikkeling worden testspecificaties opgesteld die dienen om de invoering van de pilot te testen. Deze testspecificaties worden uitgevoerd ten tijde van de acceptatietest.

7.4 Kwaliteitscontrole rapportopmaak

De kwaliteit van de opmaak van de rapporten wordt gewaarborgd door middel van naleving van de gestelde standaarden en richtlijnen uit paragraaf 5.3. Elk rapport wordt voor oplevering gecontroleerd aan de hand van de checklist uit Bijlage A van dit rapport. Daarnaast wordt gecontroleerd of het rapport voldoet aan de eisen van de rapportagetechnieken die als standaard worden gebruikt door de Sector Informatica aan de Haagse Hogeschool.

7.5 Kwaliteitscontrole software

Om de kwaliteit van het softwareontwerp te waarborgen, wordt gebruik gemaakt van een systeemontwikkelmethode en verschillende technieken.

Om de sourcecode duidelijk en leesbaar te laten zijn voor derden, worden de richtlijnen voor opbouw van sourcecode gebruikt, die als standaard opbouw gelden binnen de Haagse Hogeschool.

Zoals beschreven in voorgaande paragrafen wordt de kwaliteit van de geschreven software gewaarborgd door deze te testen door middel van testspecificaties.

Bijlage A: Checklist rapportagetechniek

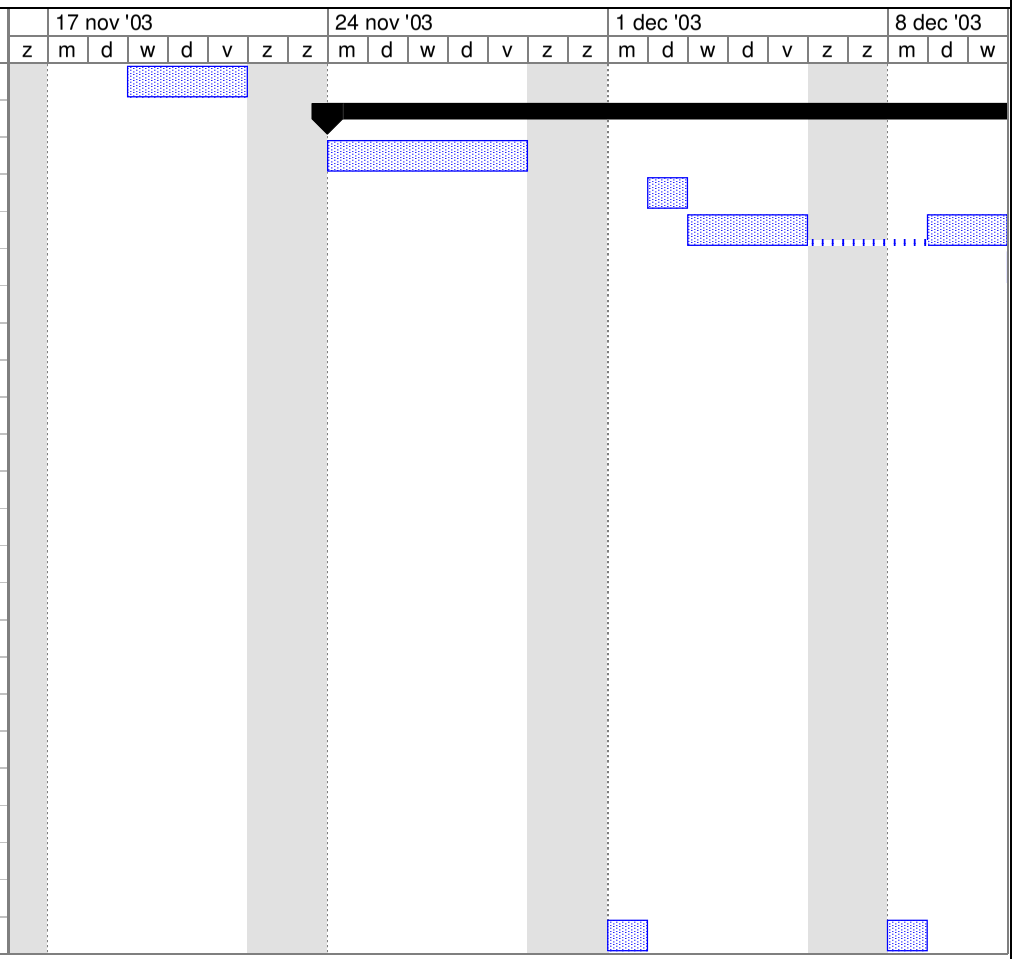
De onderstaande checklist wordt gebruikt om de opmaak van de op te leveren rapporten te controleren.

Onderdeel	Element	Aanwezig/correct
Titelpagina	Titel (en eventueel ondertitel)	ja/nee
	Naam student	ja/nee
	Naam opdrachtgever	ja/nee
	Plaats en datum	ja/nee
	Naam bedrijf	ja/nee
Voorwoord	Tussen titelpagina en inhoudsopgave	ja/nee
	Persoonlijk	ja/nee
	Plaats `	ja/nee
	Datum	ja/nee
	Naam/namen	ja/nee
Inhoudsopgave	Inhoudsopgave en voorwoord afwezig	ja/nee
	Inleiding (= hoofdstuk 1)	ja/nee
	Niveau's in hoofdstukken (max. 3) aangebracht	ja/nee
	Verband tussen hoofdstukken en paragrafen	ja/nee
	Inspringen per niveau	ja/nee
	Typografisch onderscheid per niveau	ja/nee
	Titels: informatief	ja/nee
	consequent/gelijkvormig	ja/nee
	kernachtig geformuleerd	ja/nee
	Literatuurlijst (niet als hoofdstuk)	ja/nee
	Bijlagen (met nummer/letter en titel)	ja/nee
	Paginanummering	ja/nee
Inleiding	Aanleiding/achtergrond	ja/nee
	Doelstelling rapport	ja/nee
	Structuurbeschrijving/samenvatting hoofdstukken	ja/nee
	Alinea-indeling	ja/nee
	Goede volgorde alinea's	ja/nee
Kern	Hoofdstukken ingeleid	ja/nee
	Rapportindeling overeenkomstig inhoudsopgave	ja/nee
	Verwijzingen naar de bijlagen	ja/nee
	Paginanummering loopt door in bijlagen	ja/nee
	Opsommingen correct	ja/nee
Literatuurlijst	Volgens richtlijnen voor titelbeschrijvingen	ja/nee
Bijlagen	Nummer/letter en titel	ja/nee
	Zelfstandig leesbaar	ja/nee
Publieksgerichtheid	Leesbaar/begrijpelijk voor opdrachtgever/ belanghebbenden	ja/nee
Verzorgdheid	Spelling correct/ typefouten ontbreken	ja/nee
Deadline	Tijdig opgeleverd	ja/nee

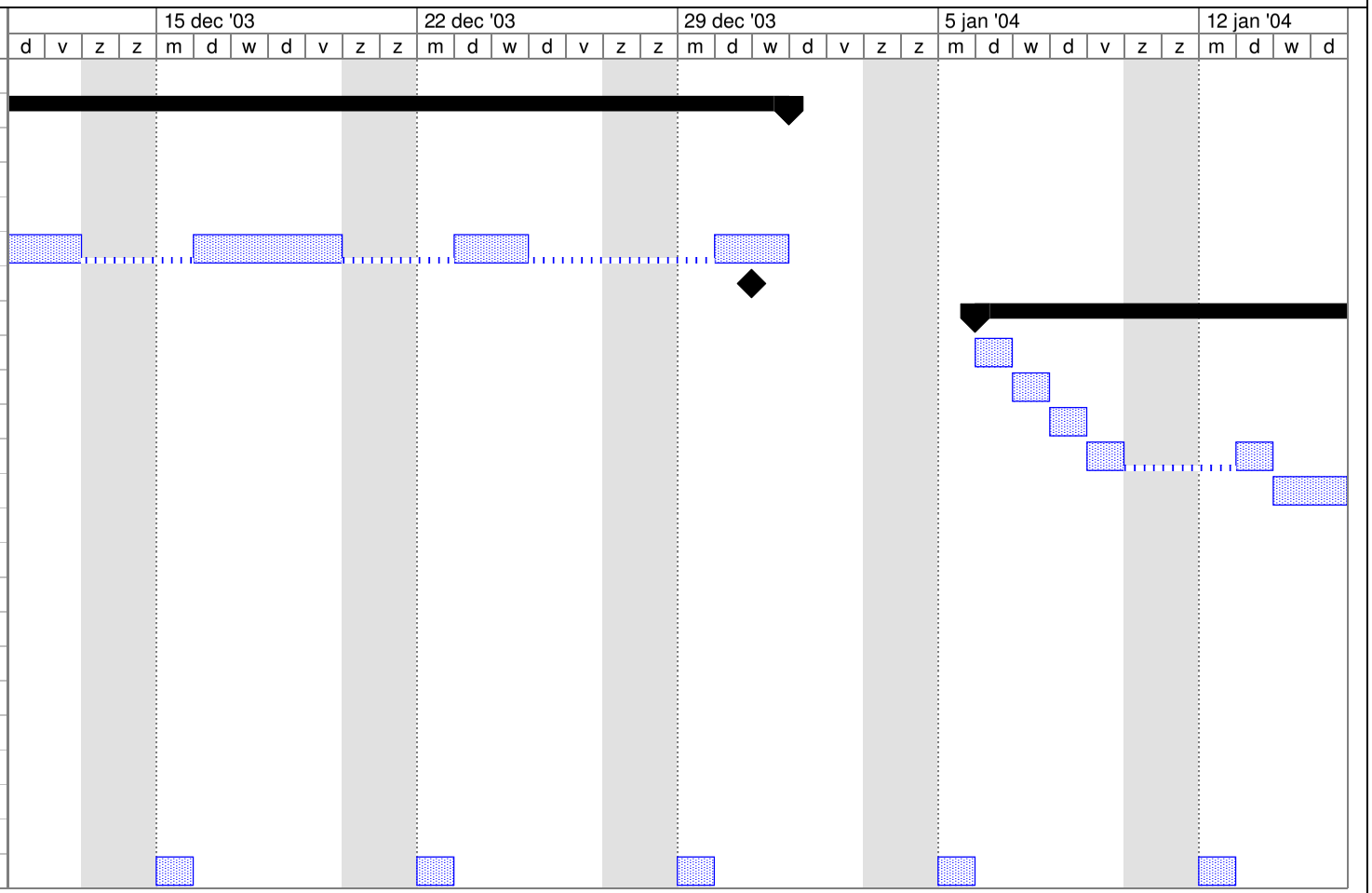
Bijlage B: Detailplanning

Op de volgende pagina's is een detailplanning opgenomen van het project.

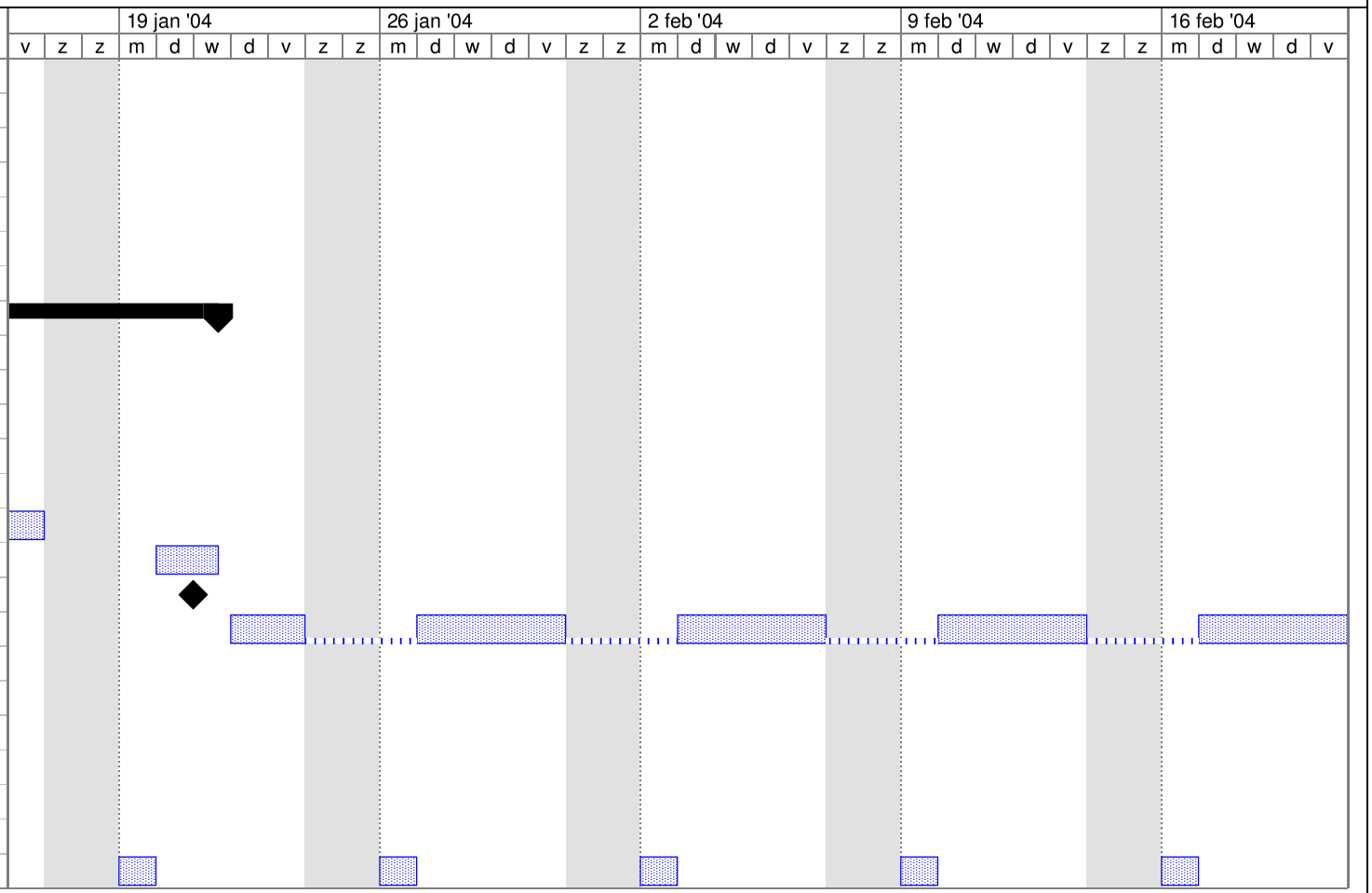
Id	Taaknaam	Duur	Begindatum	Einddatum
1	**Opstellen plan van aanpak**	**24 uur**	**wo 19-11-03**	**vr 21-11-03**
2	**Onderzoek verrichten naar interfacing**	**224 uur**	**ma 24-11-03**	**wo 31-12-03**
3	Kennis vergaren van Progress	40 uur	ma 24-11-03	vr 28-11-03
4	Kennis vergaren van Crystal Reports	8 uur	di 2-12-03	di 2-12-03
5	Kennis vergaren van .NET	40 uur	wo 3-12-03	wo 10-12-03
6	Rapport schrijven over interfacing	80 uur	do 11-12-03	wo 31-12-03
7	Opleveren rapport over interfacing	1 uur	wo 31-12-03	wo 31-12-03
8	**Uitvoeren definitiestudie**	**95 uur**	**di 6-1-04**	**wo 21-1-04**
9	Opstellen plan van aanpak definitiestudie	8 uur	di 6-1-04	di 6-1-04
10	Definiëren ontwikkelscenario	8 uur	wo 7-1-04	wo 7-1-04
11	Definiëren systeemeisen	8 uur	do 8-1-04	do 8-1-04
12	Bepalen systeemconcept	16 uur	vr 9-1-04	di 13-1-04
13	Beschouwen technische structuur	16 uur	wo 14-1-04	do 15-1-04
14	Beschouwen organisatorische inrichting	8 uur	vr 16-1-04	vr 16-1-04
15	Opstellen pilotplan	15 uur	di 20-1-04	wo 21-1-04
16	Opleveren definitiestudie	1 uur	wo 21-1-04	wo 21-1-04
17	**Pilotontwikkeling**	**232 uur**	**do 22-1-04**	**do 11-3-04**
18	**Invoering**	**48 uur**	**vr 12-3-04**	**vr 19-3-04**
19	Opstellen Plan van aanpak	4 uur	vr 12-3-04	vr 12-3-04
20	Invoeren pilots	12 uur	vr 12-3-04	di 16-3-04
21	Uitvoeren pilotacceptatie	4 uur	di 16-3-04	di 16-3-04
22	Opstellen beheerdocumentatie	24 uur	wo 17-3-04	vr 19-3-04
23	**Overhead**	32 uur	di 23-3-04	vr 26-3-04
24	**Eindverslag schrijven**	**673 uur**	**ma 1-12-03**	**vr 26-3-04**



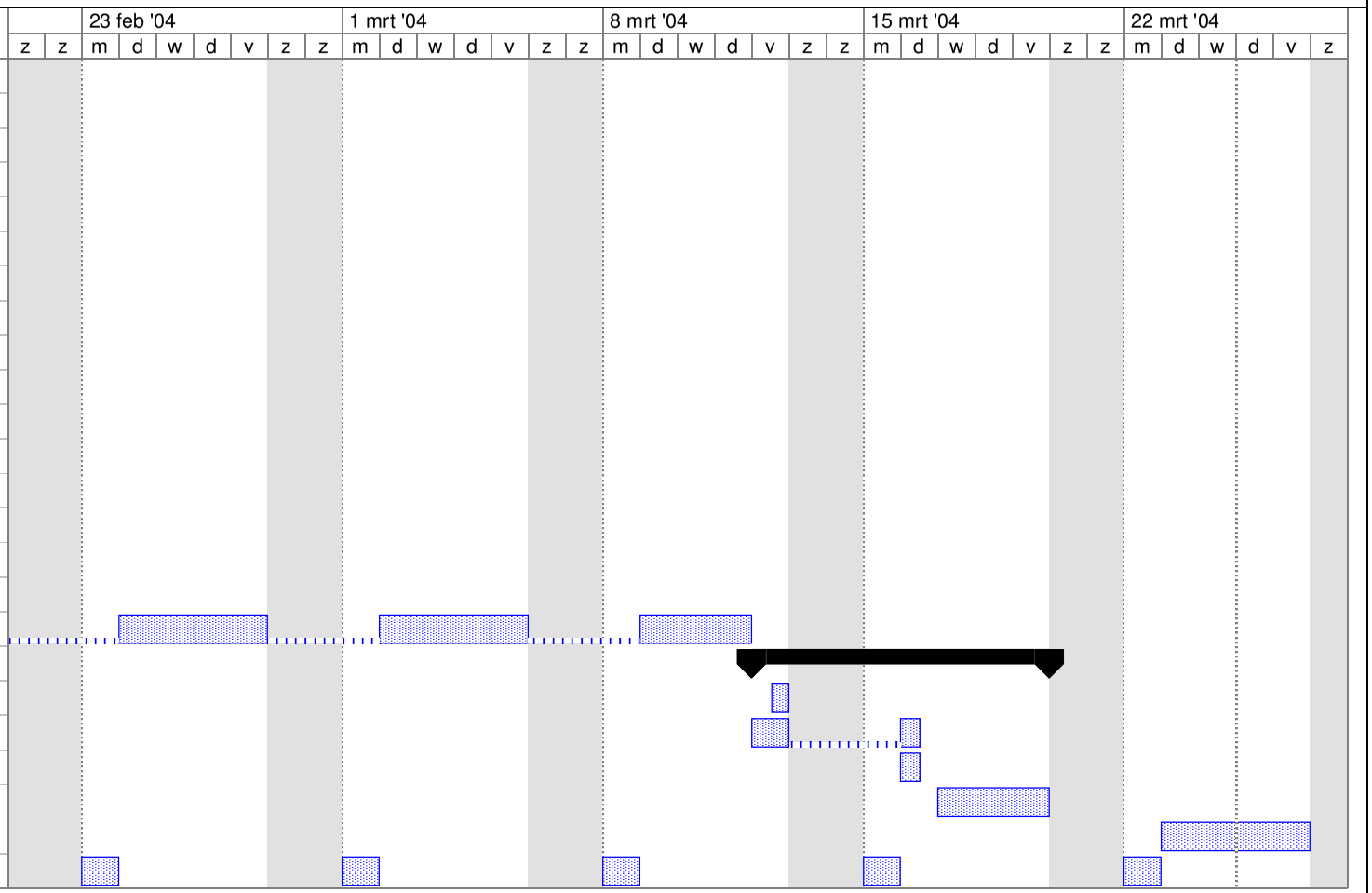
Id	Taaknaam
1	**Opstellen plan van aanpak**
2	**Onderzoek verrichten naar interfacing**
3	Kennis vergaren van Progress
4	Kennis vergaren van Crystal Reports
5	Kennis vergaren van .NET
6	Rapport schrijven over interfacing
7	Opleveren rapport over interfacing
8	**Uitvoeren definitiestudie**
9	Opstellen plan van aanpak definitiestudie
10	Definiëren ontwikkelscenario
11	Definiëren systeemeisen
12	Bepalen systeemconcept
13	Beschouwen technische structuur
14	Beschouwen organisatorische inrichting
15	Opstellen pilotplan
16	Opleveren definitiestudie
17	**Pilotontwikkeling**
18	**Invoering**
19	Opstellen Plan van aanpak
20	Invoeren pilots
21	Uitvoeren pilotacceptatie
22	Opstellen beheerdocumentatie
23	**Overhead**
24	**Eindverslag schrijven**



Id	Taaknaam
1	**Opstellen plan van aanpak**
2	**Onderzoek verrichten naar interfacing**
3	Kennis vergaren van Progress
4	Kennis vergaren van Crystal Reports
5	Kennis vergaren van .NET
6	Rapport schrijven over interfacing
7	Opleveren rapport over interfacing
8	**Uitvoeren definitiestudie**
9	Opstellen plan van aanpak definitiestudie
10	Definiëren ontwikkelscenario
11	Definiëren systeemeisen
12	Bepalen systeemconcept
13	Beschouwen technische structuur
14	Beschouwen organisatorische inrichting
15	Opstellen pilotplan
16	Opleveren definitiestudie
17	**Pilotontwikkeling**
18	**Invoering**
19	Opstellen Plan van aanpak
20	Invoeren pilots
21	Uitvoeren pilotacceptatie
22	Opstellen beheerdocumentatie
23	**Overhead**
24	**Eindverslag schrijven**



Id	Taaknaam
1	**Opstellen plan van aanpak**
2	**Onderzoek verrichten naar interfacing**
3	Kennis vergaren van Progress
4	Kennis vergaren van Crystal Reports
5	Kennis vergaren van .NET
6	Rapport schrijven over interfacing
7	Opleveren rapport over interfacing
8	**Uitvoeren definitiestudie**
9	Opstellen plan van aanpak definitiestudie
10	Definiëren ontwikkelscenario
11	Definiëren systeemeisen
12	Bepalen systeemconcept
13	Beschouwen technische structuur
14	Beschouwen organisatorische inrichting
15	Opstellen pilotplan
16	Opleveren definitiestudie
17	**Pilotontwikkeling**
18	**Invoering**
19	Opstellen Plan van aanpak
20	Invoeren pilots
21	Uitvoeren pilotacceptatie
22	Opstellen beheerdocumentatie
23	**Overhead**
24	**Eindverslag schrijven**



Bijlage 2

Rapport
“Interfacing tussen Progress en Microsoft .NET”

Voorwoord

Voor u ligt het rapport “Interfacing tussen Progress en Microsoft .NET”. Hierin wordt enerzijds ingegaan op de mogelijkheden van .NET in het algemeen en voor Reflecta in het bijzonder. Daarnaast biedt dit rapport de software ontwikkelaars van Reflecta een beschrijving van de wijze van technische implementatie van interfacing tussen Progress en .NET.

Dit rapport is het resultaat van een onderzoek dat is uitgevoerd naar aanleiding van de behoefte die binnen Reflecta bestaat om kennis op te doen van de mogelijkheden van .NET.

Lezers die vooral geïnteresseerd zijn in een beschrijving van de mogelijkheden van .NET worden verwezen naar de hoofdstukken 2 en 4. Voor een beschrijving van de technische werking van .NET wordt verwezen naar hoofdstuk 3. Lezers die geïnteresseerd zijn in de technische implementatie van interfacing tussen Progress en .NET, worden verwezen naar hoofdstuk 5.

Reeuwijk-Dorp, 8 maart 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

1. Inleiding

De laatste jaren is ".NET" een veelgehoorde term. Volgens velen is .NET "de toekomst". Wanneer echter naar een definitie van deze term wordt gevraagd, weten velen het niet te omschrijven. Toch wordt de invloed van .NET steeds groter: bestaande ontwikkelomgevingen leveren voorzieningen om te kunnen integreren met .NET en Microsoft integreert .NET in haar nieuwe besturingssystemen.
Ook Progress biedt met de release van ontwikkelomgeving OpenEdge 10 mogelijkheden om de Progress-omgeving te integreren met .NET. Mede hierdoor is vanuit Reflecta de behoefte ontstaan om duidelijkheid te krijgen in de mogelijkheden van .NET.

Het doel van dit rapport is om in deze behoefte te voorzien. Daartoe is enerzijds een onderzoek verricht naar de mogelijkheden en technische werking van .NET. Anderzijds is een onderzoek verricht naar de wijze van technische implementatie van interfacing tussen de Progress-omgeving en .NET.

De opbouw van dit rapport is als volgt. Als eerste wordt in hoofdstuk 2 ingegaan op de definitie en reikwijdte van .NET. Vervolgens bevat hoofdstuk 3 een beschrijving van de technische werking van de belangrijkste delen van .NET. In hoofdstuk 4 worden de mogelijkheden van .NET voor Reflecta onderzocht en beschreven. Als laatste wordt in hoofdstuk 5 de technische implementatie van interfacing tussen enerzijds Progress en .NET en anderzijds Progress, .NET en Crystal Reports beschreven.

2. Microsoft .NET

2.1 Inleiding

Momenteel vindt er een verschuiving plaats van het gebruik van stand-alone pc's naar het gebruik van pc's in combinatie met een verscheidenheid aan apparaten. Dit zijn apparaten als laptops, werkstations, mobiele telefoons, handcomputers en tablet-pc's. Deze apparaten zijn in staat gebruik te maken van applicaties via het Internet, al dan niet draadloos.
De behoefte groeit om de informatie die deze applicaties leveren te allen tijde beschikbaar te hebben, op elk apparaat dat wordt gebruikt, in een vorm die geschikt is voor dat apparaat. Daarnaast bestaat de behoefte, sinds informatie beschikbaar is via Internet, om onbewerkte data uit verschillende bronnen te combineren tot zinvolle informatie. Bovendien verwachten gebruikers software te kunnen gebruiken met zo min mogelijk oponthoud door installaties en upgrades.

De groei van Internet heeft veroorzaakt dat server-based applicaties meer gebruikt worden dan ooit tevoren. Communicatie tussen servers onderling is echter moeilijk, vooral tussen servers die op verschillende platformen draaien. Ook het afhandelen van bijvoorbeeld de verschillen tussen Internet browsers en verschillende versies daarvan is complex, net als het omgaan met verschillen tussen diverse typen draadloze apparaten.

Om in de bovenstaande behoeften te voorzien en de genoemde problemen op te lossen heeft Microsoft een platform ontwikkeld dat het mogelijk moet maken om op een consistente, gestandaardiseerde manier informatie, mensen en apparaten met elkaar te verbinden. Dit platform heeft de naam .NET gekregen.

In dit hoofdstuk wordt het doel en de werking van .NET beschreven. Als eerste wordt ingegaan op hoe Microsoft zelf .NET ziet en definieert. Vervolgens wordt deze subjectieve beschrijving tegen het licht gehouden door de ontvangst van .NET door het bedrijfsleven te analyseren. Als laatste worden enkele praktijkvoorbeelden van .NET gegeven.

Lezers die geïnteresseerd zijn in een specifiek onderdeel van .NET en hierover meer willen lezen, worden verwezen naar het deel "Geraadpleegde literatuur", achter in dit document.

2.2 .NET volgens Microsoft

2.2.1 Definitie

Microsoft hanteert enkele verschillende definities van .NET, die allen meer uitleg vereisen. Hieronder zijn enkele definities opgenomen zoals Microsoft die vermeldt op haar website.

> "Microsoft .NET is een reeks softwaretechnologieën van Microsoft om verbinding te maken in uw wereld van informatie, mensen, systemen en apparaten. Met .NET kunt u software op een ongekende manier integreren via het gebruik van XML-webservices: kleine, onopvallende toepassingen die als bouwstenen op elkaar - en op andere, grotere toepassingen - aansluiten via Internet." (Bron: www.microsoft.com)

> "Microsoft .NET is het Microsoft-platform voor XML-webservices. Toepassingen kunnen op basis van XML met elkaar communiceren en gegevens uitwisselen via het Internet, ongeacht het besturingssysteem, apparaat of de programmeertaal. Het Microsoft .NET-platform biedt wat ontwikkelaars nodig hebben om XML-webservices te maken en samen te voegen. Het voordeel voor gebruikers is een naadloze, fascinerende ervaring.
>
> Microsoft .NET creëert nieuwe mogelijkheden om overal, altijd en op elk apparaat gegevens te verwerken en te communiceren. .NET maakt gebruik van een gedistribueerd computermodel en bouwt op open standaarden zoals XML om pc's en andere intelligente apparaten te verbinden." (Bron: www.microsoft.com)

> ".NET is het platform voor XML-webservices van Microsoft, de volgende generatie software die ervoor zorgt dat onze wereld van informatie, apparaten en mensen op een consistente, gepersonaliseerde manier met elkaar wordt verbonden.
>
> Het .NET-platform maakt het mogelijk op XML gebaseerde toepassingen, processen en websites te maken en te gebruiken als services die samen informatie en functionaliteit delen en combineren op elk willekeurig platform of intelligent apparaat, met als doel maatwerkoplossingen te bieden aan organisaties en individuele gebruikers.
>
> .NET is een uitgebreide familie van producten die is opgebouwd rond industrie- en Internet-standaarden, die voorzien in alle aspecten op het gebied van het ontwikkelen (tools), beheren (servers), gebruiken (bouwsteenservices en smart clients) en ervaren (rijke gebruikersoplossingen) van XML-webservices. De Microsoft-toepassingen, -tools en -servers die u vandaag al gebruikt, zullen deel gaan uitmaken van .NET, evenals nieuwe producten waarmee de functionaliteit van XML-webservices beschikbaar wordt gesteld, zodat deze kan voorzien in al uw zakelijke behoeften." (Bron: www.microsoft.com)

> "Met .NET lanceren we eigenlijk Microsoft 3.0. Microsoft 1.0 was het bedrijf dat DOS en Windows lanceerde als besturingssystemen voor stand-alone pc's. Het was het bedrijf van de 'personal empowerment'.
> In de jaren '90 gingen we over naar Microsoft 2.0. Dat was namelijk de periode van 'information at your fingertips'. De doorbraak van het Internet creëert nu weer nieuwe mogelijkheden en uitdagingen.
> Voortaan zullen alle pc's en andere devices altijd met elkaar verbonden zijn. Op straat, in de kantoren, thuis en tot in de taxi toe zullen wij op ieder moment met iedereen digitale informatie en diensten kunnen uitwisselen. Microsoft 3.0 is het bedrijf dat resoluut inspeelt op deze evolutie. En .NET is het platform dat dit moet mogelijk maken."
>
> Citaat van dhr. G. de Bruyker, productmanager Developer Tools van Microsoft Belux, april 2002. (Bron: http://www.lysias.be)

Uit deze definities blijkt dat .NET moeilijk in één zin te definiëren is. Het omvat een groot scala aan producten, diensten en mogelijkheden. Samengevat kan worden gezegd dat .NET tot doel heeft om een standaard communicatiemodel te bieden en dat de twee belangrijkste componenten daarin XML en XML-webservices zijn.
Om.NET beter te kunnen begrijpen, wordt hieronder de werking en het doel van deze twee componenten behandeld. Voor meer informatie over de technische werking van de componenten wordt verwezen naar hoofdstuk 3 "Standaarden en technieken van .NET".

2.2.2 XML

Om een standaard te kunnen bieden voor data-uitwisseling maakt .NET gebruik van eXtensible Markup Language (XML).
XML wordt op dit moment al veelvuldig gebruikt bij data-uitwisseling op Internet. De informatie die een WAP-telefoon bijvoorbeeld inleest, is opgesteld in XML. En de kans is groot dat het nieuws van een online krant is samengesteld in XML en naar HTML is geconverteerd.
In een XML-document zijn alleen de gegevens zelf en de structuur van de gegevens opgeslagen. Het document hoeft slechts één keer aangemaakt te worden en de gegevens kunnen vervolgens via allerlei media getoond worden: op een computermonitor, een mobiele telefoon, een televisie, enzovoort. XML is overal te gebruiken waar informatie opgeslagen en gepresenteerd wordt. Het kan dus niet alleen gebruikt worden om informatie te presenteren, maar ook voor de uitwisseling van informatie.

XML is een krachtig middel in vooral de volgende situaties:

- Beheren van grote hoeveelheden informatie in documentvorm;
- Uitwisselen van documenten binnen en buiten een organisatie;
- Uitwisselen van berichten tussen samenwerkende systemen;
- Produceren van documenten in verschillende standaardformaten en naar verschillende media, waaronder het internet;
- Produceren van meerdere documenten vanuit één informatiebron;
- Produceren van documenten met een lange levensduur;
- Publiceren op maat: verschillende elementen voor verschillende doelgroepen.

Microsoft zegt het volgende over XML met betrekking tot .NET:

> "Hoewel XML bedrieglijk eenvoudig is, leidt deze tot een ingrijpende wijziging in de manier waarop wij software bouwen en gebruiken.
> Het web heeft voor een revolutie gezorgd in de wijze waarop gebruikers communiceren met toepassingen. XML zorgt voor een revolutie in de wijze waarop toepassingen praten met andere toepassingen—of breder gezien, waarop computers praten met andere computers—door een universele gegevensindeling te bieden die het mogelijk maakt gegevens op eenvoudige wijze aan te passen of om te zetten in andere vormen.
> Op XML gebaseerde standaarden vormen samen de open methodologie voor communicatie tussen toepassingen die bekend staat als XML-webservices."
> (Bron: http://www.microsoft.com)

2.2.3 XML-webservices

De tweede belangrijke component van .NET zijn XML-webservices. Een XML-webservice is een op zichzelf staande service die direct aanroepbaar is via het Internet.

Een voorbeeld: wanneer op een website waar telefoonnummers opgezocht kunnen worden een naam wordt ingegeven, wordt een opgemaakte HTML-pagina als output gegeven met daarin ergens het telefoonnummer. Een webservice echter krijgt als input de naam, zoekt het telefoonnummer op en geeft vervolgens alleen dat nummer als output.

Websites zijn bedoeld voor het presenteren van informatie aan een gebruiker. XML-webservices echter bieden toepassingen een direct middel om met andere toepassingen te communiceren. Toepassingen die intern worden gehost, dan wel op externe systemen, kunnen via het Internet met elkaar communiceren met behulp van XML.

Microsoft zegt het volgende over XML-webservices:

> "XML-webservices stellen toepassingen in staat met elkaar te communiceren via het Internet, ongeacht het besturingssysteem of de programmeertaal.
> Zij kunnen op elk platform worden geïmplementeerd en zijn gedefinieerd door openbare standaardisatie-instellingen zoals het W3C.
> En met behulp van XML-webservices kunnen toepassingen niet alleen gegevens uitwisselen, maar ook functies vanuit andere toepassingen aanroepen, ongeacht hoe deze andere toepassingen zijn gebouwd.
> Doordat zij gegevens kunnen uitwisselen via XML, zijn zij in staat onafhankelijk van elkaar te opereren en tegelijkertijd samen te werken in een groep van toepassingen die een bepaalde taak uitvoert." (Bron: http://www.microsoft.com)

Microsoft beschrijft een situatie waarin een XML-webservice succesvol zou kunnen worden geïmplementeerd:

> "Laten we zeggen dat u over een voorraadsysteem beschikt. Als u dit niet aansluit op andere toepassingen, is de waarde ervan beperkt. U kunt voorraden bijhouden, maar dat is een hele hoop werk.
> Bovendien zijn de mogelijkheden van dat ene systeem beperkt. Elk artikel dat u verkoopt moet afzonderlijk worden ingevoerd, niet alleen in het voorraadsysteem maar ook in uw boekhoudsysteem en in uw klantensysteem.
> Vervolgens moet u extra exemplaren van dat artikel bestellen bij uw leverancier wanneer u uw volgende order plaatst. De kosten-/batenanalyse valt negatief uit, omdat de opbrengst van het systeem niet of nauwelijks hoger is dan de overhead-kosten voor het gebruik ervan.
>
> Als u echter uw voorraadsysteem via XML koppelt aan uw boekhoudsysteem, wordt het allemaal een stuk interessanter. Nu kunnen in één stap de gevolgen van een inkoop- of verkooptransactie voor uw voorraad en kaspositie worden bijgehouden.
> Als u nog verder gaat, en uw systeem voor magazijnbeheer, uw bestelsystemen voor klanten en leveranciers en uw expediteur koppelt via XML, wordt dat voorraadbeheersysteem plotseling heel waardevol; u bent nu in staat uw bedrijfsprocessen van begin tot einde te beheren, terwijl u elke transactie slechts éénmaal hoeft in te voeren, in plaats van in elk systeem waarin deze transactie een rol speelt. Zo hebt u een stuk minder werk en wordt de kans op fouten een stuk geringer.
>
> Deze koppelingen kunnen met behulp van XML-webservices op eenvoudige wijze tot stand worden gebracht. XML-webservices stellen de toepassingen in staat informatie uit te wisselen via het Internet, ongeacht het besturingssysteem of de back-endsoftware waarmee de toepassing werkt." (Bron: http://www.microsoft.com)

2.2.4 .NET in vijf delen

Om zicht te krijgen op de reikwijdte van .NET heeft Microsoft het .NET platform onderverdeeld in vijf delen:

1. Hulpmiddelen voor de ontwikkelaar;
2. Servers;
3. Basisset XML-webservices;
4. Clients;
5. Gebruikerservaringen.

Hieronder wordt elk onderdeel toegelicht.

1. Hulpmiddelen voor de ontwikkelaar

Om het schrijven van XML-webservices zo eenvoudig mogelijk te maken, is een nieuwe ontwikkelomgeving ontworpen in vorm van het .NET Framework en de toolset Visual Studio .NET.
Visual Studio .NET vormt de volgende generatie van Visual Studio, de ontwikkeltool voor meerdere talen van Microsoft, die speciaal is ontwikkeld voor .NET. Hierop wordt dieper ingegaan in hoofdstuk 3.6 "Applicaties ontwikkelen in .NET".

2. Servers

De Microsoft .NET Enterprise Servers, met inbegrip van de familie van Microsoft Windows 2000-servers, vormen de .NET-serverinfrastructuur voor het implementeren, beheren en aansturen van XML-webservices.

3. .Basisset XML-webservices

Microsoft kondigde bij de lancering van .NET aan dat een basisset van XML-webservices zou worden ontwikkeld, die routinetaken uitvoert en als fundering zou fungeren waarop ontwikkelaars verder kunnen bouwen. Deze basisset kreeg de codenaam "HailStorm" en werd later omgedoopt tot “.NET MyServices”.
Hailstorm zou bestaan uit services die op de gebruiker zijn gericht, en niet op specifieke apparaten, netwerken of toepassingen. HailStorm was gebaseerd op Passport, het systeem voor gebruikerverificatie van Microsoft.
“Met HailStorm ontvangen gebruikers relevante informatie, op de wijze zoals zij deze nodig hebben, op de apparaten die zij gebruiken en op basis van de voorkeuren die zij hebben ingesteld”, aldus Microsoft.

In april 2002 is de strategie van Hailstorm echter sterk gewijzigd als gevolg van de sceptische ontvangst van Hailstorm door bedrijven en consumenten. Het aantal services dat wordt ontwikkeld is verlaagd en de overkoepelende naam Hailstorm (of MyServices) is langzamerhand verdwenen.

4. Clients

Clients zijn pc's, laptops, werkstations, telefoons, handheld computers, tablet pc's, spelconsoles en andere apparaten. Deze apparaten zijn in staat toegang te krijgen tot XML-webservices. Clients maken gebruik van software dat XML-webservices ondersteunt en in staat stelt toegang te krijgen tot gegevens, ongeacht het type en het aantal clients waarmee gewerkt wordt en de locatie hiervan.
Tot de .NET-clientsoftware behoren Windows CE, Windows Embedded, Windows 2000 en Windows XP. Met deze software zullen pc's, laptops, werkstations, intelligente telefoons, handheld computers en tablet pc's worden aangestuurd.

5. Gebruikerservaringen

Gebruikerservaringen zijn XML-webservices die gebruikers in staat stellen toegang te krijgen tot informatie overal op het Internet en op stand-alone toepassingen. Microsoft zal zowel individuele gebruikers als bedrijven van gebruikerservaringen gaan voorzien. Tot de producten die door Microsoft worden omgebouwd tot .NET-ervaringen behoren MSN, bCentral, .NET Passport en Microsoft Visual Studio .NET.

2.2.5 Verwachte toekomst

Volgens Microsoft zal .NET de manier waarop gedacht wordt over en gebruik gemaakt wordt van computers drastisch veranderen. Momenteel wordt de computerwereld overheerst door twee concepten: de server en de client. .NET breidt dit model uit tot een gedistribueerd computermodel van flexibel gekoppelde services.
In plaats van het gebruikelijke onderscheid tussen pc en server, vindt de verwerking plaats waar dit het beste uitkomt, ongeacht of het hierbij gaat om een server, pc, handheld apparaat of ander apparaat. “Dit is intelligent computergebruik voor een nieuwe generatie intelligente apparaten”, aldus Microsoft.

Hoewel.NET veranderingen in het computergebruik veroorzaakt, blijven er volgens Microsoft ook veel dingen hetzelfde:

- Individuele gebruikers werken nog altijd met vertrouwde interfaces binnen de .NET-omgeving, zoals Microsoft Office. “Hierdoor blijven de omscholingskosten beperkt en kunnen individuele gebruikers direct aan de slag met de .NET-software”, aldus Microsoft;
- Besturingssystemen zoals Microsoft Windows, UNIX, Windows CE en PalmOS worden nog altijd op dezelfde hardware uitgevoerd;
- Ontwikkelaars kunnen hun favoriete programmeertaal blijven gebruiken. Het .NET-platform zorgt ervoor dat XML-webservices samenwerken, ongeacht de brontaal;
- Oude systemen hoeven niet te worden vervangen. Enkele Microsoft .NET-producten zijn specifiek ontworpen om bestaande toepassingen eenvoudig in nieuwe .NET XML-webservices en .NET-programma's te kunnen integreren.

Volgens Microsoft merken bedrijven, individuele gebruikers en ontwikkelaars allemaal op een andere manier de invloed van het .NET-computermodel. Voor individuele gebruikers zullen de veranderingen zorgen voor een “opvallend persoonlijkere, meer geïntegreerde computerervaring”. Voor bedrijven en ontwikkelaars zal de manier waarop zij software ontwikkelen en producten verkopen veranderen.

Om .NET tot een succes te maken in de toekomst, heeft Microsoft drie doelstellingen gedefinieerd die als hendels worden gezien die moeten worden overgehaald.

> “Naar onze mening zijn er drie hendels die moeten worden overgehaald om deze nieuwe generatie van gedistribueerd computergebruik zo snel mogelijk te laten opgroeien:
>
> - *Webservices*: alles moet een webservice zijn. Dit geldt zowel voor software als voor bronnen in het netwerk, zoals opslagcapaciteit;
> - *Samenvoeging en integratie*: wanneer deze webservices er eenmaal zijn, deze op eenvoudige wijze samenvoegen en integreren;
> - *Gebruikerservaringen*: eenvoudige en aanspreekbare toepassingen voor eindgebruikers ontwikkelen.”
>
> (Bron: http://www.microsoft.com)

2.3 .NET volgens critici

Om een goed beeld van de werkelijke mogelijkheden en impact van .NET te krijgen, is het noodzakelijk om naast de visie van Microsoft de meningen van bedrijven over .NET te peilen.
Om achter deze meningen te komen, is een onderzoek verricht door op Internet te zoeken naar reacties op .NET van bedrijven. Daarnaast is gebruik gemaakt van enkele artikelen uit tijdschriften.

2.3.1 Reactie bedrijfsleven

Een veelgehoorde reactie uit het bedrijfsleven is dat de mogelijkheden van .NET erg aanspreken en dat men ook verwacht dat .NET in de toekomst een grote rol zal gaan spelen. Het probleem is echter dat bedrijven vaak niet weten wat .NET voor hén kan betekenen.
Een van de oorzaken van dit probleem is dat men vaak niet goed weet wat .NET precies inhoudt. Tot op heden kampt Microsoft met het probleem dat klanten blijven vragen wat .NET eigenlijk is. ".NET is lijkt voor ons een interne wedstrijd bij Microsoft om zoveel mogelijk jargon in een concept te duwen". Volgens het onafhankelijke onderzoeksinstituut Forrester Research is het ontstaan van dit probleem te wijten aan een verkeerde marketing van .NET.

Het concept van .NET bleek vaag vanaf het begin, maar werd steeds verwarrender door het gebruik van dezelfde naam ".NET" voor veel verschillende producten en diensten. Zo is er de ontwikkeltool Visual Studio .NET en werd een nieuwe server operating system "Microsoft .NET Server 2003" gedoopt. Daarnaast kondigde Microsoft aan dat een basisset van XML-webservices in ontwikkeling was, met als codenaam ".NET MyServices".
Bedrijven en consumenten zien het verband tussen de verschillende producten en diensten niet en krijgen hierdoor geen zicht op de essentie van .NET. Het gevolg daarvan is een afwachtende houding.

Een andere veelgehoorde reden waarom bedrijven niet gelijk overgaan tot integratie met .NET is het feit dat de voorbeelden die worden genoemd waarbij .NET wordt ingezet, vaak niet in de praktijk van bedrijven toepasbaar lijken. Microsoft heeft met .NET tot doel systemen met elkaar te integreren. De voorbeelden die worden genoemd, gaan uit van een wereld waarbij die integratie reeds een feit is. De voorbeelden (zie bijvoorbeeld het voorbeeld uit paragraaf 2.2.3) klinken dan ook meer ambitieus dan werkelijk toepasbaar in de praktijk.
Bedrijven staan vaak ook sceptisch tegenover de doelstelling van Microsoft om systemen zover met elkaar te integreren. Men vraagt zich af of dat wel wenselijk is en men twijfelt aan het vermogen van Microsoft om in een dergelijke geïntegreerde wereld de beveiliging van gegevens te garanderen.

Het gevolg van deze houding is dat er na ruim drie jaar nog weinig van .NET in de praktijk te zien is. Het aantal webservices dat momenteel wereldwijd actief is, loopt slechts in de tientallen.
Toch zijn bedrijven over het algemeen wel geïnteresseerd in de mogelijkheden van .NET. De mogelijkheden om platformonafhankelijkheid te creëren en het idee om gegevens tussen verschillende applicaties en apparaten uit te kunnen wisselen, spreken aan.

Volgens het bedrijfsleven zijn er meer redenen waarom .NET nog geen groot succes is. Het feit dat .NET een platform is, wordt gezien als een zwakheid. Om .NET op een computer te draaien moet de .NET runtime worden geïnstalleerd, een bestand van 18Mb. Eigenlijk is deze runtime een uitbreiding van het operating system, zodat een objectgeoriënteerde omgeving ontstaat. Verwacht wordt dat het nog enkele jaren zal duren voordat een stabiele versie van deze runtime op een voldoende percentage computers in de wereld is geïnstalleerd. Tot dat tijdstip zal de markt terughoudend zijn met het leveren van .NET applicaties: het aantal computers waarop het draait is immers beperkt.

Tenslotte is het nog niet stabiel zijn van .NET een groot obstakel. Niemand had verwacht dat .NET meteen na introductie stabiel zou zijn. Dat blijkt in de praktijk ook realiteit te zijn. Alleen eenvoudige web applicaties, waarbij slechts een deel van het .NET platform wordt gebruikt, zijn stabiel. Zodra .NET in een complexere situatie wordt ingezet, blijken er soms problemen te ontstaan. Op Internet klaagt een bedrijf: "Wij hebben een .NET applicatie gemaakt, maar helaas verdwenen op de meest onverwachte momenten spontaan de knoppen van het invoerscherm. Gelukkig was dit een eenvoudig systeem, dat wij snel in Delphi konden herschrijven, maar het geeft aan hoe moeilijk het in de praktijk vaak is om systemen met nieuwe technologie te realiseren."

2.3.2 Verwachte toekomst

Om .NET tot een succes te maken, heeft Microsoft haar marketingstrategie aangepast. De naamgeving van .NET-gerelateerde producten wordt momenteel herzien. Zo is bijvoorbeeld de term ".NET" uit "Windows Server .NET 2003" gehaald. Daarnaast heeft Microsoft een ".NET Connected" logo ontworpen, dat producten en diensten die gebruik maken van .NET, zullen dragen. Deze aanpassingen in de marketingstrategie hebben tot doel om de essentie van .NET eindelijk duidelijk te maken bij bedrijven en consumenten.

Ondanks het tot op heden uitblijven van een grote doorbraak, verwachten bedrijven en consumenten dat .NET een grote toekomst heeft. De onderstaande redenen voor deze verwachting zijn vaak gehoord:

- Er is behoefte aan integratie van systemen en aan standaarden. Microsoft heeft nu eenmaal de beste marktpositie om in deze behoefte te kunnen voorzien, dus .NET zal een succes worden;
- De .NET ontwikkelomgeving is veel krachtiger dan de bestaande omgevingen. Het verleden heeft geleerd dat degene met de krachtigste ontwikkelomgeving een grote voorsprong heeft. De omgeving zal dé omgeving van de toekomst worden;
- Een nieuwe techniek wordt vaak met argwaan ontvangen en nooit gelijk ingezet. Dit geldt vooral voor .NET, dat een grote impact kan hebben voor een organisatie. Wanneer .NET stabieler wordt en zijn waarde in de praktijk kan bewijzen, zal de markt de techniek breder gaan accepteren.

2.4 Voorbeelden

Om het principe van .NET meer concreet te maken, zijn hieronder enkele voorbeelden beschreven, waarbij gebruik gemaakt wordt van .NET technieken.

2.4.1 .NET Passport

.NET Passport is een online service waarbij een consument zich met één e-mailadres en één wachtwoord kan aanmelden bij alle aan de .NET Passport-service deelnemende websites en services.
Microsoft is zelf initiatiefnemer van deze dienst, dat als doel heeft een eerste stap te zetten in de richting van integratie van systemen. Eenmaal ingelogd, is de gebruiker bekend bij alle deelnemende websites en services. Vervolgens kan informatie op maat worden aangeboden, afhankelijk van de voorkeur van de gebruiker.
Microsoft heeft het .NET Passport geïntegreerd in haar e-mail dienst Hotmail, waardoor elke gebruiker van Hotmail een .NET Passport heeft. Gezien de populariteit van Hotmail is het aantal potentiële gebruikers groot. Het aantal deelnemende websites en diensten is echter klein: momenteel bevat de lijst met deelnemers enkele tientallen namen.

2.4.2 .NET Alerts

.NET Alerts is een service van Microsoft dat gebruikers voorziet in berichtgeving op maat. Gebruikers kunnen Alerts ontvangen op hun computer door middel van MSN of Windows Messenger, per e-mail of op een mobiel apparaat als een telefoon of PDA. Bedrijven kunnen gebruik maken van deze service om klanten persoonlijke informatie te bieden. Zo is het voor een bank bijvoorbeeld mogelijk om door middel van deze service een klant erop te attenderen dat het saldo van zijn bankrekening onder een bepaald minimum is geraakt. Online shops kunnen .NET Alerts gebruiken om klanten op de hoogte te houden van de status van hun orders.
Bij de lancering van .NET Alerts eind 2001 maakte het deel uit van het Microsoft MyServices pakket. Toen dat pakket in 2002 geen succes bleek te zijn, is .NET Alerts grotendeels zelfstandig verder ontwikkeld door Microsoft. .NET Alerts wordt momenteel gebruikt door enkele grote organisaties als e-Bay, NASDAQ en McAfee.

2.4.3 XML-webservices in de praktijk

Het aantal XML-webservices dat momenteel is geïmplementeerd, loopt slechts in de tientallen. Hieronder zijn enkele voorbeelden van organisaties in Nederland opgenomen die XML-webservices hebben ingezet.

- *Ministerie van Financiën*
  Maakt voor haar website gebruik van een XML-webservice die door een nieuwsleverancier wordt aangeboden aan partijen die nieuwsberichten willen integreren in hun eigen infrastructuur (internet, intranet).
  De inhoud van de site wordt via een netwerk van leveranciers van nieuwsberichten aangeleverd. Door gebruik te maken van XML-webservices wordt de site automatisch gevuld met informatie van de nieuwsleveranciers. De gestandaardiseerde opbouw van de berichten d.m.v. XML zorgt ervoor dat verschillende afnemers gebruik kunnen maken van de services.

- *Traumacentrum Utrecht*
  Door middel van .NET wordt snel toegankelijke en actuele informatie voor de hulpdiensten en de rampencoördinatoren gerealiseerd.
  Wanneer een ramp plaatsvindt, wordt op dat moment het Ziekenhuis Informatie Systeem geactiveerd. Deze applicatie haalt eerst de IP-adressen van alle relevante zorginstellingen en hulpdiensten op via een XML-webservice. Op deze manier blijft de applicatie automatisch up-to-date, ongeacht eventuele reorganisaties die zich in de toekomst in de sector kunnen plaatsvinden.
  De applicatie haalt daarna volgens een vast interval de gegevens binnen van alle patiënten die bij de spoedeisende hulp zijn binnengekomen en aan de selectiecriteria voldoen. Hulpdiensten kunnen de gegevens via mobiele apparaten raadplegen en aanvullen.

- *ProRail*
  ProRail had de behoefte om informatie over onderhoudswerkzaamheden aan het spoor uit te wisselen tussen verschillende betrokken partijen. Daarvoor is een website opgezet die gebruik maakt van .NET technologieën. Door middel van deze website zijn de betrokken partijen in staat om op een efficiënte wijze over de benodigde informatie te beschikken.

2.5 Conclusie

Momenteel is er behoefte aan integratie van systemen en standaardisatie voor uitwisseling van gegevens. Microsoft heeft met het .NET platform tot doel in deze behoefte te voorzien. .NET kan informatie, mensen, systemen en apparaten koppelen op een gestandaardiseerde wijze.

Tot op heden blijft een grote doorbraak van .NET uit. Bedrijven hebben een afwachtende houding ten opzichte van .NET, die veroorzaakt is door onduidelijkheid over de mogelijkheden van .NET. Daarnaast twijfelen bedrijven aan de beveiliging binnen .NET: is Microsoft wel in staat om hun gegevens in sterk geïntegreerde omgevingen te beveiligen?

De problemen waar .NET op dit moment mee te kampen heeft, houden een grote doorbraak nog tegen, maar wanneer het product verder wordt uitontwikkeld en zijn waarde zal tonen in de praktijk, zal een groter draagvlak ontstaan. Bedrijven en consumenten voorzien een grote toekomst voor .NET.

3. Standaarden en technieken van .NET

3.1 Inleiding

Om inzicht te krijgen in de technische werking van .NET beschrijft dit hoofdstuk de architectuur van het .NET Framework en de belangrijkste gebruikte standaarden en technieken waar .NET op is gebaseerd. Verder wordt ingegaan op de mening die het bedrijfsleven heeft over deze gebruikte technieken. Vervolgens wordt de beveiliging binnen .NET behandeld. Als laatste wordt beschreven wat ontwikkelaars benodigd hebben om toepassingen kunnen ontwikkelen in een .NET omgeving.

Dit hoofdstuk is bedoeld voor technici die meer willen weten over de achterliggende werking van .NET. Lezers die echter geïnteresseerd zijn in de mogelijke betekenis van .NET voor Reflecta, worden verwezen naar hoofdstuk 4.

3.2 .NET Framework

Het .NET Framework vormt de infrastructuur voor het .NET platform. Kennis van de architectuur van het .NET Framework is noodzakelijk om de werking van .NET te kunnen begrijpen. Door middel van deze kennis wordt inzicht verkregen in hoe .NET programmeertaal- en platformonafhankelijkheid creëert.

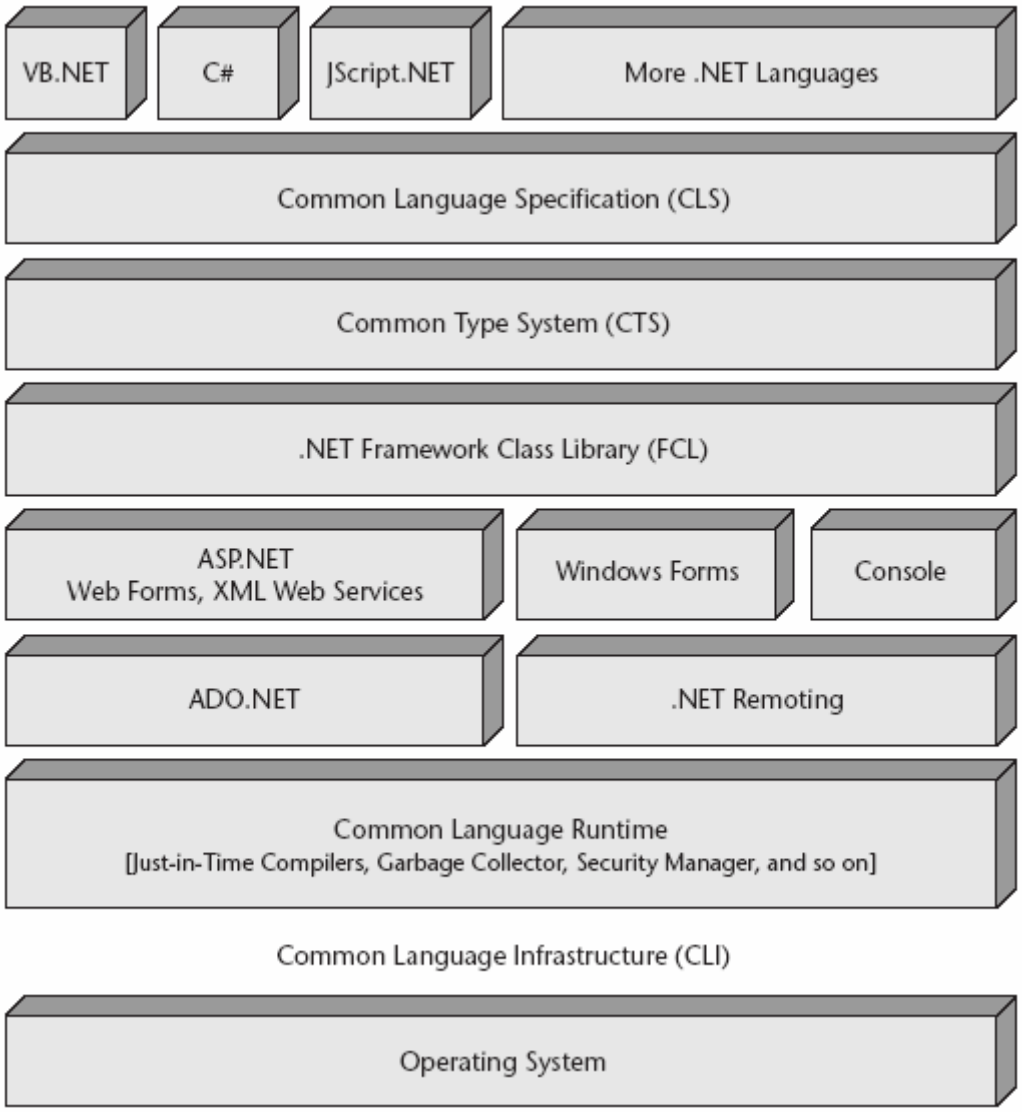
Het .NET Framework is gelaagd, modulair en hiërarchisch opgebouwd.

- Gelaagd:
  Elke laag uit het Framework heeft een niveau van abstractie. .NET programmeertalen vormen de bovenste laag en zijn het meest abstract. De Common Language Runtime is de onderste en minst abstracte laag.

- Modulair:
  Het Framework is verdeeld in modules, die elk hun eigen verantwoordelijkheden hebben.

- Hiërarchisch:
  Een laag heeft alleen services nodig die worden geleverd door lager gelegen lagen.

De architectuur van het Framework is schematisch weergegeven in onderstaand figuur.



Figuur 1: *.Architectuur .NET Framework*

Hieronder wordt elk onderdeel uit het Framework kort toegelicht, te beginnen bij de bovenste laag.

- **Programmeertalen**

Een van de doelen van .NET is om onafhankelijkheid van programmeertalen te creëren. Ontwikkelaars kunnen kiezen in welke taal zij .NET applicaties schrijven. In paragraaf 3.6 “Applicaties ontwikkelen in .NET” wordt dieper ingegaan op de programmeertalen zelf.

.NET maakt onderscheid in zogenaamde *managed* en *unmanaged* programmeertalen. Managed talen zijn Visual Basic .NET en C#. Code die in deze talen is geschreven wordt gecompileerd naar Intermediate Language (IL) en niet naar machine code. De IL wordt opgeslagen in een bestand, de assembly, samen met metadata dat de klassen, methoden en attributen beschrijft die zijn gecreëerd door de ontwikkelaar.

De IL draait in de Common Language Runtime (CLR). Deze laadt en verifieert de assembly om te verzekeren dat de IL correct is. Wanneer methoden worden aangeroepen compileert de CLR deze in machine code die geschikt is voor de machine waar de assembly op draait (Just In Time compilatie). Wanneer de assembly eenmaal draait, verzorgt de CLR services als beveiliging en geheugenbeheer. De applicatie wordt dus ge-*managed* door de CLR.

Code die is geschreven in unmanaged talen wordt direct gecompileerd naar machine code. Alle talen die zijn ontwikkeld vóór het ontstaan van .NET zijn unmanaged. .NET kan deze talen compileren naar managed code, waardoor deze talen alsnog gebruik maken van de CLR.

- **Common Language Specification (CLS)**

De CLS is een verzameling specificaties of richtlijnen waaraan een .NET programmeertaal moet voldoen. Zo zijn er bijvoorbeeld operaties gedefinieerd voor events en properties die voldoen aan een standaard naamgeving.
Het doel van de CLS is de uitwisseling tussen programmeertalen te vereenvoudigen door het gebruik van standaarden.

- **Common Type System (CTS)**

De CTS is een catalogus met .NET types (bijvoorbeeld System.Int32, System.Decimal, System.Boolean). Ontwikkelaars hoeven deze types niet direct aan te spreken. De CTS zorgt ervoor dat types die gedefinieerd zijn in een willekeurige .NET programmeertaal, worden omgezet naar generieke .NET types.

- **.NET Framework Class Library (FCL)**

De FCL bevat een verzameling diensten zoals input/output van bestanden, sockets en toegang tot databases. In andere omgevingen dan .NET worden deze diensten door het operating system aangeboden. Alle applicaties die werken op het .NET Framework maken gebruik van de functies van de FCL. Dit heeft als voordeel dat de applicaties in principe onafhankelijk zijn van het operating system, en dat ze kunnen werken op alle operating systems die het .NET Framework ondersteunen.

- **ASP.NET**

ASP.NET is een geheel nieuwe implementatie van het ASP Webserver Framework, waarmee naast webapplicaties ook webservices gemaakt kunnen worden.
De belangrijkste verandering ten opzichte van ASP is dat web-pagina's nu worden gecompileerd, waarmee snelheidswinst wordt behaald. Daarnaast is in ASP.NET caching, sessiebeheer en beveiliging verbeterd.
Veel ontwikkelaars waren niet geïnteresseerd in het leren van de scripttaal ASP, omdat het wéér een nieuwe taal was die moest worden aangeleerd of omdat men een aversie had tegen scripttalen. ASP.NET kan echter worden geschreven in de *managed* taal naar keuze, waardoor bovenstaande bezwaren verdwijnen.

- **Windows Forms en Console Applications**

Windows Forms is een verzameling van klassen voor het maken van grafische gebruikersinterfaces en is vergelijkbaar met de Forms Engine van Visual Basic 6.
Windows Forms is vooral een code generator die klassen genereert voor formulieren, knoppen, text boxes, menu's en andere elementen van de grafische gebruikersinterface.

Console Applications zijn reeds bekend uit de ontwikkelomgeving C. In .NET echter zijn Console Applications te gebruiken in alle *managed* programmeertalen.

- **ADO.NET**

ADO.NET biedt klassen voor toegang tot databases. Zoals de naam al aangeeft vormt ADO.NET een evolutie van ActiveX Data Objects (ADO). ADO.NET lijkt nog wel op ADO, maar is nu gebaseerd op XML. Naast directe databasetoegang is het mogelijk een gegevensverzameling uit een database op te halen en lokaal als database te benaderen (d.m.v. ADO.NET DataSets).
De gegevensverzameling wordt daarvoor in XML overgezonden en kan op verzoek ook in XML vertaald worden (en vice versa).
Er is lokaal toegang tot de gegevens mogelijk, waarbij door de relaties tussen tabellen kan worden genavigeerd, tabellen kunnen worden gesorteerd en gefilterd, en events kunnen worden gegenereerd bij het wijzigen van de gegevens. Nadat lokaal wijzigingen in de gegevensverzameling zijn doorgevoerd, kan weer worden gesynchroniseerd met de centrale database.
In paragraaf 5.4 wordt dieper ingegaan op ADO.NET.

- **.NET Remoting**

.NET Remoting wordt gebruikt voor het ontwikkelen van gedistribueerde applicaties. Hiermee is het mogelijk om te communiceren met andere objecten in het netwerk. Objecten kunnen gebruik maken van verschillende transportkanalen en berichtformaten.

- **Common Language Runtime (CLR)**

De CLR vormt de omgeving waarin .NET applicaties worden uitgevoerd. De CLR werkt bovenop het operating system, zodat applicaties niet direct met het operating system communiceren. Op deze manier worden applicaties onafhankelijk van het operating system waarop deze draaien.
De CLR bevat diensten voor bijvoorbeeld het beheren van geheugen, processen en threads, excepties, beveiliging en metadata.
In het bijzonder biedt de CLR “garbage collection”, waarbij objecten die niet meer worden gebruikt automatisch worden opgeruimd. .NET applicaties kunnen worden uitgevoerd in elke omgeving waarvoor een CLR implementatie beschikbaar is, onafhankelijk van de onderliggende hardware en software.
De CLR wordt beschreven door de Common Language Infrastructure (CLI).

3.3 Standaarden

Microsoft is betrokken bij het standaardisatieproces voor de belangrijkste technologieën achter .NET. Door standaardisatie van technologieën die software in staat stellen met elkaar te communiceren en gegevens uit te wisselen, worden oplossingen mogelijk die meerdere platformen omvatten.
Microsoft zegt het volgende over standaardisatie:

> “Eén van de belangrijkste lessen die we hebben geleerd van het web is dat standaarden voor het beschrijven van gegevens en interacties (zoals HTML) oplossingen mogelijk maken met een veel groter bereik dan mogelijk zou zijn met behulp van bedrijfsspecifieke technologieën. Het .NET-platform is gebouwd op een technologiefundament van industriestandaarden. “
> (Bron: http://www.microsoft.com)

De standaarden en technieken die worden gebruikt voor .NET zijn niet van Microsoft zelf. Ze worden door een consortium beheerd en zijn daardoor wereldwijd geaccepteerde standaarden (door o.a. Microsoft, IBM, HP, SUN en Oracle).

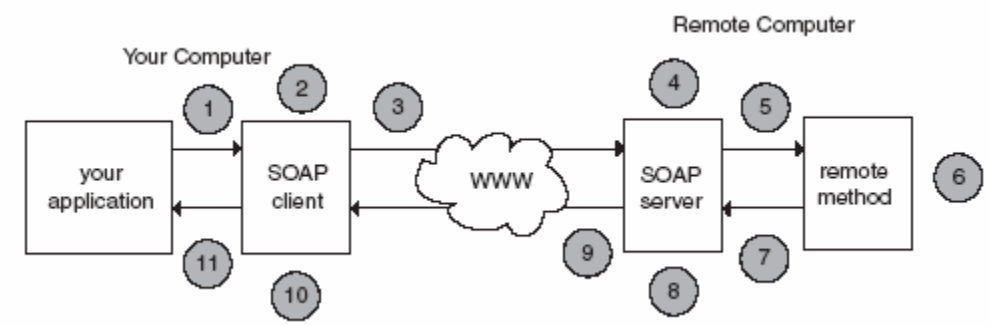
De belangrijkste standaarden voor .NET vormen XML, SOAP, UDDI en WSDL. XML is reeds in het voorgaande hoofdstuk beschreven. De overige standaarden worden hieronder toegelicht.

3.3.1 SOAP

SOAP (Simple Object Access Protocol) is een op XML gebaseerd protocol dat toepassingen in staat stelt elkaar aan te roepen op een gestandaardiseerde wijze. Hierdoor wordt het mogelijk om gedistribueerde toepassingen te bouwen die werken via het Internet.
SOAP voorziet in onafhankelijkheid van operating system, programmeertaal en objectmodel. In eerste instantie bundelde SOAP het transport (HTTP) en de boodschap (XML). In de huidige SOAP-specificatie (versie 1.2) kunnen ook andere protocollen het transport van de boodschap vervullen, waaronder SMTP, FTP en MQ.

SOAP definieert een standaard XML-berichtstructuur. Een SOAP-bericht kan worden gezien als een envelop. De envelop bevat het adres (URL) waarnaar het bericht moet worden verstuurd: de header van het SOAP-bericht. De inhoud van de envelop bevat het werkelijke bericht in XML formaat. Dit bericht bestaat onder meer uit de naam van de functionaliteit die wordt aangeroepen, alsmede de parameters die vereist zijn om deze functionaliteit uit te kunnen voeren.

Om een indruk te krijgen van de wijze waarop SOAP in de praktijk functioneert, is hieronder een voorbeeld opgenomen, waarin een applicatie informatie aanvraagt die zich op een andere computer bevindt.



Figuur 2: *Voorbeeld SOAP*

Het verloop van de communicatie wordt hieronder stapsgewijs uitgelegd.

1. De applicatie zendt een aanvraag voor data uit. Deze wordt opgevangen door de SOAP client;
2. De SOAP client vertaalt deze aanvraag naar een SOAP-bericht;
3. De client verzendt dit SOAP-bericht naar de computer waar de aangevraagde data zich bevindt. Op deze computer draait een SOAP server, die luistert naar binnenkomende SOAP-berichten;
4. De SOAP server vertaalt het binnengekomen SOAP-bericht naar een formaat dat de betreffende applicatie op deze computer kan lezen;

5. Het vertaalde bericht vormt de input voor de applicatie;
6. De applicatie voert de betreffende functie uit en genereert output data;
7. Deze data wordt teruggegeven aan de SOAP server;
8. De SOAP server fungeert nu als een SOAP client: de data wordt omgezet naar een SOAP-bericht;
9. Dit SOAP-bericht wordt verzonden naar de lokale SOAP client;
10. Deze client vertaalt het SOAP-bericht naar een formaat dat de betreffende applicatie kan lezen;
11. De applicatie beschikt nu over de benodigde data, in een formaat dat geschikt is voor deze applicatie.

Uit dit voorbeeld blijkt dat wanneer applicaties onderling willen communiceren, zij slechts hoeven te beschikken over de mogelijkheid om met SOAP-berichten om te gaan.

3.3.2 UDDI

UDDI (Universal Description, Discovery and Integration) voorziet in een catalogus van XML-webservices, waarin beschikbare webservices zijn opgenomen. Deze catalogus bevat alle informatie om een specifieke webservice te kunnen vinden. De catalogus fungeert als een soort Gouden Gids. Deze kan worden bezocht door applicaties en vervolgens dynamisch worden doorzocht naar webservices die beschikbaar en relevant zijn.

Een UDDI specificatie bestaat uit drie delen:

- Informatie over de eigenaar van de webservice;
- Een toewijzing aan categorieën;
- Een technische beschrijving van de webservice.

Op dit moment kunnen webservices worden aangemeld bij Microsoft, IBM, SAP en HP.

Microsoft zegt het volgende over UDDI:

> "Vele ondernemingen kunnen slechts via het Internet zaken doen met de handelspartners elders in de wereld die ze al kennen, en vaak dan nog alleen met hen die dezelfde applicaties en Web Services gebruiken.
> Succesvolle E-commerce vereist dat ondernemingen in staat zijn elkaar te ontdekken, hun behoeften en mogelijkheden bekend te maken, en hun services te integreren door gebruik te maken van elkaars technologie, Web Services en zakelijke processen."
>
> "UDDI vereenvoudigt het vinden van handelspartners, en zorgt er voor dat zij ook uw bedrijf kunnen vinden, zodat de samenwerking via het Internet gemakkelijker wordt."
>
> "Potentiële handelspartners zullen elkaar hierdoor snel ontdekken en dynamisch met elkaar kunnen samenwerken via het Internet met gebruikmaking van hun eigen applicaties."
> (Bron: http://www.microsoft.com)

3.3.3 WSDL

Web Services Description Language (WSDL) voorziet in een standaard om te beschrijven welke functies een XML-webservice biedt en welke argumenten moeten worden gebruikt om deze functies aan te roepen. UDDI maakt gebruik van WSDL.

WSDL beschrijft de volgende vier eigenschappen van een webservice:

- De publiek toegankelijke functies;
- De input en output parameters van de publiek toegankelijke functies;
- Informatie over welk transport protocol wordt gebruikt;
- Informatie over de locatie van de webservice.

Een WSDL document wordt opgebouwd door middel van XML specificaties. Om het concept concreet te maken, is hieronder een voorbeeld van een WSDL document opgenomen. Hierin wordt een XML-webservice beschreven, die op basis van een ingevoerde naam als uitvoer “Hello <naam>” teruggeeft. Om het voorbeeld overzichtelijk en kort te houden, zijn de XML specificaties uit het document in conceptvorm opgenomen.

<definitions>:
The HelloService
<message>:
1) SayHelloRequest:FirstName parameter
2) SayHelloResponse:greeting return value
<portType>:
SayHello operation that consists of a request/response service
<binding>:
Direction to use the SOAP HTTP transport protocol
<service>:
Service available at: http://localhost:8080/soap/servlet/rpcrouter

Figuur 3: *Voorbeeld WSDL document*

3.4 Beveiliging

Na lancering van .NET bleek dat dit platform te maken had met enkele beveiligingsproblemen.
Zo is bijvoorbeeld de data-uitwisseling van en naar webservices niet voldoende beveiligd. Ook komen er nog steeds geregeld beveiligingslekken in Microsoft Passport aan het licht. Gegevens van gebruikers blijken daarbij vrij toegankelijk.

De beveiligingsproblemen zijn een oorzaak van het uitblijven van het succes van .NET. Organisaties en consumenten hebben nog te weinig vertrouwen in de beveiliging van .NET.
Om de beveiliging te verbeteren, heeft Microsoft in samenwerking met IBM en VeriSign de specificatie Web Service Security (WS-Security) ontwikkeld. WS-Security is een specificatie die een aantal SOAP extensies beschrijft en aangeeft hoe beveiligde en gesigneerde berichten in een webserviceomgeving kunnen worden uitgewisseld.

Een initiatief dat onafhankelijk van WS-Security ontwikkeld is, is de Security Assertion Markup Language (SAML). SAML is een taal die ontwikkeld is om op een eenduidige

manier autorisatie-informatie tussen verschillende beveiligingssystemen, waaronder webservices, te delen. SAML bevat een aantal standaardprotocollen en messaging frameworks als XML Signature, XML Encryption en SOAP.
SAML is als standaard aangenomen door het E-business consortium Organization for the Advancement of Structured Information Standards (OASIS).

3.5 Risico's

De vele nieuwe standaarden en technieken worden met argwaan ontvangen door het bedrijfsleven. Deze paragraaf gaat in op de risico's die deze standaarden en technieken met zich meebrengen.

Het nieuwe concept van XML-webservices brengt enkele serieuze risico's met zich mee. Het bedrijfsleven vraagt zich af of Microsoft rekening heeft gehouden met deze risico's en hoe daar mee om wordt gegaan.
De volgende risico's worden genoemd:

- Webservice spaghetti
  Wanneer webservices op grote schaal worden geïmplementeerd, kan de onderlinge afhankelijkheid voor grote problemen zorgen. Als gevolg van verschillende standaarden zijn webservices net iets verschillend van elkaar geïmplementeerd, met verschillende beveiligingsconfiguraties of toegangsbeheer.
  Het gebrek aan administratieve consistentie tussen providers en de algemene complexiteit kan het risico van mislukken vergroten.

- Afhankelijkheid van aanbieder van de webservice
  De gebruiker van een webservice is afhankelijk van de aanbieder van de webservice. Wanneer de aanbieder de webservice wijzigt of stopt, is de gebruiker daarvan de dupe. Op dit moment zijn tegen dit risico nog geen maatregelen genomen. Waarschijnlijk worden deze pas genomen wanneer dergelijke situaties zich in de praktijk voordoen.

- Slechte performance
  Wanneer webservices van elkaar afhankelijk zijn, wordt het risico van een slechte performance en het optreden van crashes groter.
  Een webservice kan bijvoorbeeld een functie aanroepen die door een andere webservice wordt geleverd, maar die tijdelijk offline is. De functie kan niet worden uitgevoerd en de gebruiker krijgt een time-out melding.
  Wanneer de gebruiker vervolgens contact opneemt met de leverancier van zijn software, rijst de vraag naar boven wie voor dit probleem verantwoordelijk is: de softwareleverancier of de eigenaar van de webservice.
  Hieruit blijkt de noodzaak van het maken van goede afspraken over verantwoordelijkheden.

- Beveiligingsrisico's
  Zoals in de vorige paragraaf reeds is beschreven, is de problematiek rondom beveiliging een groot risico. Traditionele applicaties zijn connectiegericht, maar webservices zijn "berichtgericht" en missen de garantie van een koppeling tussen de partijen. Daardoor werken de traditionele connectiegerichte beveiligingsmethoden niet of onvoldoende. Alle aspecten van beveiliging dienen te worden herzien.

- Gebrek aan flexibiliteit
  Een kleine verandering in een webservice kan grote gevolgen hebben voor applicaties die gebruik maken van deze webservice. Wanneer bijvoorbeeld het type van een invoerparameter wordt gewijzigd, levert dat problemen op voor de applicatie die de service aanroept.

3.6 Applicaties ontwikkelen in .NET

Om applicaties te kunnen ontwikkelen in .NET is de .NET Framework SDK (Software Development Kit) en een ontwikkelomgeving benodigd.
De .NET Framework SDK is gratis te downloaden vanaf de website van MSDN. De SDK bevat compilers, de .NET class libraries en command-line tools. Daarnaast voorziet de SDK in voorbeelden, documentatie en hulpmiddelen om .NET applicaties te ontwikkelen.

Microsoft heeft gelijktijdig met de lancering van .NET de nieuwe ontwikkelomgeving Visual Studio .NET uitgebracht. Het doel van Visual Studio .NET is om ontwikkelaars snel en eenvoudig Windows-applicaties, Internetapplicaties en XML-webservices te kunnen laten ontwikkelen.
Deze omgeving is de opvolger van Visual Studio, de omgeving waarin applicaties in Visual Basic kunnen worden ontwikkeld. Begin 2003 bracht Microsoft het zeer uitgebreide Visual Studio .NET 2003 uit.
Om .NET programmeertaalonafhankelijk te maken, ondersteunt deze omgeving ongeveer 20 programmeertalen. Op dit moment worden veel talen geschikt gemaakt voor gebruik binnen .NET, zoals bijvoorbeeld de talen COBOL, Eiffel, Perl, Smalltalk en Python. De meest gebruikte en best ondersteunde .NET talen zijn C++, Visual Basic, ASP en de nieuwe taal C# ("C-Sharp").

De objectgeoriënteerde taal C# is momenteel de meest gebruikte taal waarin .NET applicaties worden ontwikkeld. De oorzaak hiervoor is dat C# het beste functioneert in een .NET omgeving, omdat de taal speciaal voor de CLR is ontwikkeld. Andere talen hebben uitbreidingen of aanpassingen nodig om goed te kunnen werken in een .NET omgeving. Dit levert in de praktijk vaak problemen op.
De syntax van C# is gebaseerd op die van C++, maar bevat een aantal functionaliteiten die het gebruik eenvoudiger en productiever maakt dan C++, zoals bijvoorbeeld garbage collection.

Momenteel zijn er meerdere ontwikkelomgevingen voor .NET beschikbaar. Hieronder is een overzicht opgenomen van de belangrijkste omgevingen die momenteel beschikbaar zijn.

Productnaam	**Kenmerken**	**Ontwikke-laar**
Visual Studio .NET 2003 Enterprise Architect	- Lichtste versie van Visual Studio .NET 2003; - Bedoeld voor ontwikkelaars die zelfstandig software ontwikkelen.	Microsoft
Visual Studio .NET 2003 Enterprise Developer	- Geavanceerde versie van Visual Studio .NET 2003; - Bedoeld voor grootschalige ontwikkeling in teamverband binnen ondernemingen.	Microsoft
Visual Studio .NET 2003 Professional	- Meest uitgebreide versie van Visual Studio .NET 2003; - Biedt ook ondersteuning bij ontwikkeling van mobiele webtoepassingen.	Microsoft
Visual Basic of C# of C++ .NET	Separaat te kopen delen van Visual Studio .NET 2003.	Microsoft
C# Builder	Gericht op toepassingen voor .NET in C#	Borland
SharpDevelop	- Open Source product (gratis te downloaden); - Vooral voor Windows Forms applicaties.	IC#Code
WebMatrix	- Gratis te downloaden; - Gericht op ASP.NET webapplicaties.	ASP.NET team

De meest volledige omgeving is momenteel Visual Studio .NET 2003. De andere omgevingen richten zich allen op een deel van .NET.

4. Mogelijke betekenis van .NET voor Reflecta

4.1 Inleiding

Eind 2003 heeft Progress de nieuwe versie van haar ontwikkelomgeving, OpenEdge 10, uitgebracht. OpenEdge 10 biedt ondersteuning van .NET door middel van enkele nieuwe technieken. Dit maakt het mogelijk om (delen van) .NET te integreren in Progress applicaties. Dit hoofdstuk gaat in op deze en andere mogelijkheden van .NET.

4.2 XML-webservices

Volgens Microsoft zullen veel toekomstige applicaties worden geïmplementeerd als webservices. In hoofdstuk 2 is al gezegd dat daar tot op heden nog weinig van in de praktijk te zien is: een afwachtende houding overheerst.
Ook voor Reflecta lijkt dit momenteel de juiste houding. De problemen waar webservices momenteel mee kampen, dienen eerst opgelost te worden. Wanneer webservices zich echter in de praktijk gaan bewijzen, lijken zij voor Reflecta interessante mogelijkheden te bieden.
Webservices maken namelijk communicatie met mobiele apparaten (bijvoorbeeld handhelds of pocket pc's) mogelijk. Bij een activiteit als bijvoorbeeld orders lopen zou dit toepasbaar zijn. De gebruiker kan met behulp van een pocket pc orders opvragen uit het systeem en de ordergegevens tijdens het lopen opvragen en bewerken.

4.3 Gebruikersinterface Progress applicaties

Tot Progress OpenEdge 10 was de gebruikersinterface van een Progress applicatie afhankelijk van de functionaliteiten die Progress bood op dit gebied. De Progress gebruikersinterfaces staan bekend als functioneel, maar weinig aantrekkelijk.
Eén van de doelen die Progress wil bereiken met OpenEdge 10 is "User Interface Independence": Progress wil de ontwikkelaar zijn eigen gebruikerinterface kunnen laten kiezen. Zij zegt hierover:

> "One of the strengths of OpenEdge is the availability of many interfaces. These include Web Speed browsers, 4GL GUI clients, WebClient, Character clients, Web services, and OpenClients such as Java. The new addition here, of course is the Open Client for .NET."
> (Bron:PowerPoint presentatie "Accessing the OpenEdge AppServer From .NET" door Ken Wilner, medewerker Progress)

Een van de mogelijkheden is dus een .NET gebruikersinterface. Progress richt zich met OpenEdge 10 sterk op deze interface, omdat het momenteel de meest gebruikte en meest vertrouwde interface is:

> "At this point in time it is clear that Microsoft, through their operating systems and products, define the most used look and feel."
> Bron: OpenEdge 10 Beta Training, PowerPoint Presentatie ".NET Open Client", slide 14

Voor Reflecta biedt dit de mogelijkheid om de gebruikersinterface van Progress applicaties te verbeteren. Door middel van de integratie van OpenEdge 10 met .NET kan een gebruikersinterface geïmplementeerd worden die is gebaseerd op .NET, waardoor een Windows look-and-feel wordt verkregen. De business logica van de applicatie kan (onafhankelijk van de gebruikersinterface) in een Progress-omgeving worden uitgevoerd, op de wijze zoals dit reeds plaatsvindt. (Voor meer technische details hierover wordt verwezen naar hoofdstuk 5 van dit document).

Het implementeren van een .NET gebruikersinterface kent enkele voor- en nadelen ten opzichte van Progress gebruikersinterfaces.
Voordelen zijn:

- Gebruikers zijn reeds vertrouwd met de .NET (Windows) interface, wat de gebruikersvriendelijkheid van een Progress applicatie verhoogt;
- De .NET interface biedt meer mogelijkheden dan een Progress gebruikersinterface;
- Met relatief weinig extra inspanning kan een applicatie een sterk gemoderniseerd uiterlijk krijgen.

Een nadeel is dat onbekendheid met .NET tot gevolg heeft dat eerst onderzoek naar de wijze van technische implementatie moet worden gedaan voordat de .NET gebruikersinterface kan worden geïmplementeerd.

4.4 Datasource Crystal Reports

De reporting tool Crystal Reports haalt haar gegevens om een rapport te creëren op uit een (Progress) database. Deze gegevens worden in een XML bestand geplaatst, dat vervolgens wordt ingelezen door Crystal Reports.
Dit XML bestand kan (in haar huidige vorm) veel redundante gegevens (dezelfde gegevens die meerdere keren voorkomen) bevatten. Het bestand is namelijk opgebouwd uit één tabel, waarin elk record alle benodigde gegevens bevat.
Als voorbeeld is hieronder een tabel opgenomen waarin verkoopgegevens staan.

ID	Datum	Naam	Adres	Postcode	Woonplaats	Computer	Geheugen	Prijs
1	26-04	Jan	Straat 1	1111 AB	Utrecht	Pentium 4 4 ghz.	3 Gb	€ 1000
2	12-05	Jan	Straat 1	1111 AB	Utrecht	Pentium 4 5 ghz.	4 Gb	€ 2000
3	13-06	Jan	Straat 1	1111 AB	Utrecht	Pentium 4 2 ghz.	1 Gb	€ 900
4	13-06	Piet	Laan 3	1234 ER	Amsterdam	Pentium 4 2 ghz.	1 Gb	€ 900

Figuur 4: *Voorbeeld redundantie*

In het grijze vlak zijn de redundante gegevens aangegeven. Deze gegevens maken het bestand onnodig groot. Dit is niet efficiënt en het kost meer tijd dan strikt noodzakelijk om dit bestand over een netwerk te versturen.

De combinatie van Progress OpenEdge 10 en .NET biedt een oplossing voor dit probleem. OpenEdge 10 biedt een nieuwe techniek om gegevens uit meerdere tabellen uit de database met hun onderlinge relaties op te slaan in één object (de ProDataSet,

zie hoofdstuk 5). Dit object is een afspiegeling van de fysieke database: het kan dezelfde structuur (tabellen en relaties) hebben als deze database.
In het voorbeeld hierboven zou dat betekenen dat het object de tabellen Klanten, Computers en Verkopen bevat. Door middel van relaties tussen de tabellen kan de juiste combinatie van deze gegevens gemaakt worden.
Op deze manier wordt het voorkomen van redundante gegevens voorkomen.

Crystal Reports kan dit object (een ProDataSet) echter niet rechtstreeks inlezen. Om dit object te kunnen gebruiken als datasource dient het te worden omgezet naar een .NET data-object. In hoofdstuk 5 wordt beschreven hoe dit technisch te realiseren is.

Door de datasource van Crystal Reports te laten werken op een .NET data-object als datasource, wordt verwacht dat de tijd die benodigd is om een rapport te creëren, zal verminderen. Omdat de datasource efficiënter is opgebouwd, is het bestand dat over het netwerk moet worden verzonden namelijk kleiner.

4.5 Conclusie

.NET kan momenteel op twee gebieden waardevol zijn voor Reflecta.

- De gebruikersinterface van Progress applicaties kan worden verbeterd door het implementeren van een .NET gebruikersinterface;
- De datasource van Crystal Reports kan efficiënter worden opgebouwd door gebruik te maken van de combinatie van een nieuwe techniek in Progress OpenEdge 10 (de ProDataSet) en .NET.
  Een efficiëntere opbouw van de datasource kan de tijd die benodigd is om een rapport te creëren, doen verminderen.

Een verdere integratie van .NET door middel van XML-webservices kan in de toekomst interessant zijn. Het biedt mogelijkheden op het gebied van communicatie met mobiele apparaten. Voordat hier aan begonnen wordt, verdient het aanbeveling om te wachten tot de .NET-technologieën verder zjin uitontwikkeld en hun nut hebben bewezen in de praktijk.

Op dit moment worden de twee genoemde mogelijkheden van .NET nader onderzocht, ontworpen en geïmplementeerd.

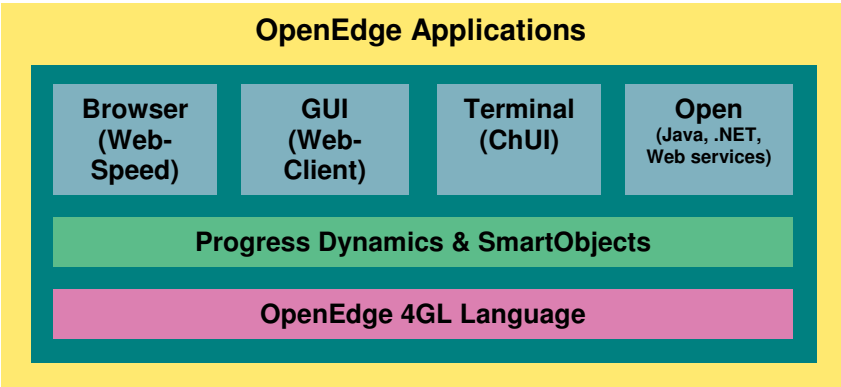
5. Interfacing tussen Progress en .NET

5.1 Inleiding

Dit hoofdstuk gaat in op de technische realisatie van interfacing tussen Progress en .NET. Het hoofdstuk is bedoeld voor de ontwikkelaars van Reflecta Automation en heeft tot doel hen op de hoogte te brengen van de wijze waarop Progress samenwerkt met .NET.
Als eerste wordt ingegaan op de mogelijkheden van Progress OpenEdge 10 op het gebied van .NET. Vervolgens wordt een nieuwe techniek in OpenEdge 10, de ProDataSet, beschreven. Deze techniek is ontworpen volgens de .NET specificaties en is mede bedoeld voor interfacing naar .NET.
In paragraaf 5.4 wordt de ADO.NET DataSet beschreven. ADO.NET is het gedeelte van het .NET Framework dat gegevenstoegang realiseert door middel van een set van classes.
Vervolgens wordt in paragraaf 5.5 de technische realisatie van interfacing tussen Progress en .NET beschreven. Als laatste wordt in paragraaf 5.6 ingegaan op de wijze waarop Crystal Reports omgaat met .NET gerelateerde datasources.

5.2 Progress OpenEdge 10 en .NET

Eén van de nieuwe mogelijkheden in OpenEdge 10 is “User Interface Independence”: naast de Progress gebruikersinterfaces is het mogelijk om zogenaamde Open Clients te gebruiken, zoals Java, Web services en .NET gebruikersinterfaces.



Figuur 5: *Lagen in OpenEdge applicaties*

De ondersteuning voor .NET gebruikersinterfaces komt voort uit het feit dat Microsoft gebruikersinterfaces de meest bekende zijn:

> “We are focused on the Microsoft .NET client integration at this time because it is the leading user interface in the industry. At this point in time it is clear that Microsoft, through their operating systems and products, define the most used look and feel.”
> Bron: OpenEdge 10 Beta Training, PowerPoint Presentatie “.NET Open Client”, slide 14

De beslissing welke gebruikersinterface te implementeren dient af te hangen van de eisen aan de te ontwikkelen applicatie. Wanneer bijvoorbeeld de applicatie beide in een grafische en characted based omgeving dient te worden geïmplementeerd, is het noodzakelijk om de Progress gebruikersinterfaces te gebruiken. Wanneer de applicatie echter een modern uiterlijk dient te hebben, is het gebruik van de Java of .NET Open Client aan te raden.

Het gebruik van een Open Client gebruikersinterface heeft enkele voor- en nadelen. Voordelen zijn:

- Flexibiliteit in het inzetten van een gebruikersinterface naar keuze;
- Applicaties kunnen een moderne look-and-feel krijgen;
- Wanneer de eisen en trends veranderen op het gebied van gebruikersinterfaces, hoeft slechts de interface zelf aangepast te worden, zonder dat dit consequenties heeft voor de business logica.

Nadelen zijn:

- Implementatie van een Open Client gebruikersinterface kost extra tijd, omdat er onderzoek dient te worden verricht naar de werking van Open Clients. Daarnaast kan implementatie in bijvoorbeeld een Java of .NET omgeving kennis vereisen die niet in huis is;
- De applicatie krijgt een extra laag (bijvoorbeeld een .NET of Java laag), wat onwenselijk kan zijn, omdat dit vertragend kan werken.

5.3 Progress ProDataSet

5.3.1 Definitie

In OpenEdge 10 wordt een nieuw 4GL object, de ProDataSet, geïntroduceerd. Dit object is een uitbreiding op de bestaande data objecten (TempTables, queries, buffers, etc.). De ProDataSet kan worden gezien als een “in-memory” relationele dataset, bestaande uit TempTables. Een ProDataSet heeft de volgende kenmerken:

- Het is één object, bestaande uit één of meer TempTables, die onderlinge relaties kunnen hebben;
- De ProDataSet staat los van zijn source. Het bevat een “in-memory” kopie van de data uit de database;
- De data die de ProDataSet bevat, kan worden bewerkt. Deze wijzigingen kunnen vervolgens in de database worden weggeschreven;
- Het is een non-visueel object, het heeft geen associaties met de gebruikersinterface;
- Het object kan zowel statisch als dynamisch worden gedefinieerd;
- Het object kan worden doorgegeven als parameter;
- Het object is één op één te converteren naar een ADO.NET DataSet.

De ProDataSet speelt een grote rol bij communicatie met een .NET omgeving. Het object is ontwikkeld op basis van .NET specificaties en is ook bedoeld om een interface te bieden naar .NET.

5.3.2 Implementatie van een ProDataSet

Hieronder wordt de implementatie van een ProDataSet stapsgewijs beschreven. Slechts de hoofdpunten worden hier behandeld, voor een volledige beschrijving van de syntax wordt verwezen naar de documentatie die door Progress beschikbaar wordt gesteld. Het onderstaande voorbeeld beschrijft het definiëren en het vullen van een ProDataSet die twee TempTables bevat: ttCustomer en ttOrder (gebaseerd op de Sports2000 database).

1. Definieer TempTables
   - Hiervoor kan de bekende syntax worden gebruikt.

```
DEFINE TEMP-TABLE ttCustomer  LIKE Customer.
DEFINE TEMP-TABLE ttOrder     LIKE Order.
```

2. Definieer de ProDataSet
   - Hiervoor wordt het nieuwe object DATASET gebruikt;
   - DATA-RELATION definieert de relatie tussen de parent en child tabellen;
   - RELATION-FIELDS geeft de velden in de parent en child tabellen aan waarop de relatie is gebaseerd;
   - Op basis van de gedefinieerde relatie is Progress in staat om de juiste childs bij de parents te vinden.

```
DEFINE DATASET dSet FOR ttCustomer, ttOrder
    DATA-RELATION dsRelation FOR ttCustomer, ttOrder
     RELATION-FIELDS(custnum,custnum).
```

3. Definieer Data Sources
   - Elke buffer uit de ProDataSet dient een Data Source te hebben;
   - Een Data Source bestaat uit het resultaat van een query of uit een database tabel.

```
DEFINE QUERY qCust FOR customer, order.
QUERY qCust:QUERY-PREPARE("FOR EACH customer WHERE cust-
num > 80, each order of customer WHERE order-num < 40").

DEFINE DATA-SOURCE dsCust FOR QUERY qCust.
DEFINE DATA-SOURCE dsOrder FOR Order.
```

4. Koppel Data Sources aan ProDataSet buffers

```
BUFFER ttCustomer:ATTACH-DATA-SOURCE(DATA-SOURCE
dsCust:HANDLE).

BUFFER ttOrder:ATTACH-DATA-SOURCE(DATA-SOURCE
dsOrder:HANDLE).
```

5. Vul de ProDataSet
   - FILL vult elke buffer uit de ProDataSet, beginnend bij de top-level buffer(s).
   - Door middel van het attribuut FILL-MODE is het mogelijk de wijze van vullen per buffer te specificeren.

```
DATASET dSet:FILL().
```

De ProDataSet is nu gevuld en gereed voor gebruik.

Hieronder volgen enkele aandachtspunten bij het gebruik van de ProDataSet:

- Data uit de buffers van de ProDataSet kan worden benaderd op de gebruikelijke wjize waarop data uit TempTables wordt benaderd.

- Elk element van de ProDataSet kan ook dynamisch worden gecreëerd. Onder andere de onderstaande commando's worden hiervoor gebruikt:
  - CREATE TEMP-TABLE;
  - CREATE DATASET;
  - ADD-RELATION();
  - ADD-BUFFER();
  - CREATE DATA-SOURCE.

- Een ProDataSet kan op dezelfde wijze als parameter worden doorgegeven als een TempTable, zowel statisch als dynamisch:
  - Statisch: door middel van "DATASET", zoals "TABLE" voor TempTables;
  - Dynamisch: door middel van "DATASET-HANDLE", zoals "TABLE-HANDLE"voor TempTables;
  - Lokaal: door middel van "HANDLE", om een handle van een ProDataSet door te geven aan een lokale procedure.

  Lokaal worden DATASET en DATASET-HANDLE standaard by-reference doorgegeven. Remote worden deze standaard by-value doorgegeven.

- Het bijhouden van wijzigingen in een ProDataSet wordt ondersteund door middel van een Before-Image TempTable die automatisch bij de ProDataSet wordt opgenomen.
  Deze tabel bevat:
  - Initiële waarden voor elk toegevoegd record;
  - Originele waarden voor elk gewijzigd en voor elk verwijderd record.

- Door middel van het SET-CALLBACK-PROCEDURE mechanisme kan een procedure worden aangeroepen, die net vóór of gelijk na het vullen van de ProDataSet (of van een buffer uit de ProDataSet) wordt uitgevoerd.
  Dit mechanisme kan worden gebruikt om bijvoorbeeld de inhoud van een buffer uit de ProDataSet aan te passen, wanneer de standaard FILL-methode niet voldoet.
  De mogelijke events zijn:
  - BEFORE-FILL;
  - AFTER-FILL;
  - BEFORE-RECORD-FILL;
  - AFTER-RECORD-FILL.

5.3.3 Beperkingen

Eerste implementaties van de ProDataSet hebben enkele beperkingen van deze nieuwe techniek aan het licht gebracht. Daarnaast meldt Progress zelf dat er enkele beperkingen aan de ProDataSet zijn in OpenEdge 10. Hieronder een overzicht van deze beperkingen.

- De standaard vulmethode van de ProDataSet verloopt anders dan verwacht en voldoet niet in alle gevallen.
  Tijdens testimplementaties van de ProDataSet is gebruik gemaakt van de volgende query op de parentbuffer “ttCustomer”:

```
FOR EACH Customer WHERE Custnum > 80, each Order of Customer
WHERE Ordernum < 40.
```

  De buffer “ttCustomer” wordt correct gevuld met gegevens van klanten met klantnummer groter dan 80, die orders hebben met een ordernummer kleiner dan 40.
  Vervolgens wordt echter de buffer “ttOrder” gevuld met álle ordergegevens van de klanten die in “ttCustomer” staan. De bedoeling was echter om alleen die ordergegevens in “ttOrder” te plaatsen, waarvan het ordernummer < 40 is.
  De buffers uit de ProDataSet worden dus *per query* gevuld en niet door middel van één query op de parentbuffer.
  Dit probleem is dus op te lossen door een query op de childbuffer “ttOrder” te plaatsen:

```
FOR EACH Order WHERE Ordernum < 40.
```

  Dit is echter inefficiënt, omdat op deze manier tweemaal vrijwel dezelfde query moet worden gedefinieerd. In complexere situaties is dit niet acceptabel.

  Progress heeft een forum op haar website geopend waar over OpenEdge 10 kan worden gediscussieerd. Toen bovenstaand probleem daar werd voorgelegd, werd het beaamd:

> “The fact that we can do it manually shouldn't take away from the point that there is a problem in the functionality. Especially when in beta, we should strive for giving feedback to improve the final product instead of being content with workarounds.”
> Bron: www.forums.progress.com , User Forum OpenEdge 10A.

  Er werd ook een andere oplossing voor het probleem geboden, in de vorm van een workaround door het gebruik van een SET-CALLBACK-PROCEDURE. Hierin werd echter de vulmethode van de ProDataSet zodanig aangepast, dat deze oplossing niet als alternatief kon dienen.
  Tot op heden is er nog geen afdoende oplossing voor dit probleem gevonden.

- Progress meldt dat OpenEdge 10 geen batching ondersteunt bij het doorgeven van ProDataSets tussen server en client:

> "Batching is not currently available in this release. Be advised that if you select a large ProDataSet it will all be passed at once. If you want to do batching it will have to be handled programmatically at this point. Development is working on a strategy. Look for future white papers on this topic."
> Bron: OpenEdge 10 Beta Training, PowerPoint Presentatie "ProDataSet", slide 19

- Progress houdt op haar website een Bug Bulletin Board bij, waarin bugs vermeld worden die in OpenEdge 10 Beta zijn geconstateerd.
  Hieronder een overzicht van bekende bugs met betrekking tot ProDataSets. Het is niet bekend in hoeverre deze bugs zijn opgelost in OpenEdge 10A.

Progress bug #	**Omschrijving**
20031106-028	Error 3253 on AppServer calling from .NET with ProDataSet and alternate buffer
20031031-051	ProDataSet generates mismatch with rowid field in Temp-Table
20031021-001	"ERROR condition: doCheckSchema got mismatched fields!!!! (7211)" calling a ProDataSet from .NET client
20031002-027	Visual Studio .NET client hangs when calling a ProDataSet
20030924-002	The DataSet buffer AFTER-FILL and the BEFORE-RECORD-FILL and AFTER-RECORD-FILL fail to fire when manually creating records in the DataSet

5.4 ADO.NET DataSet

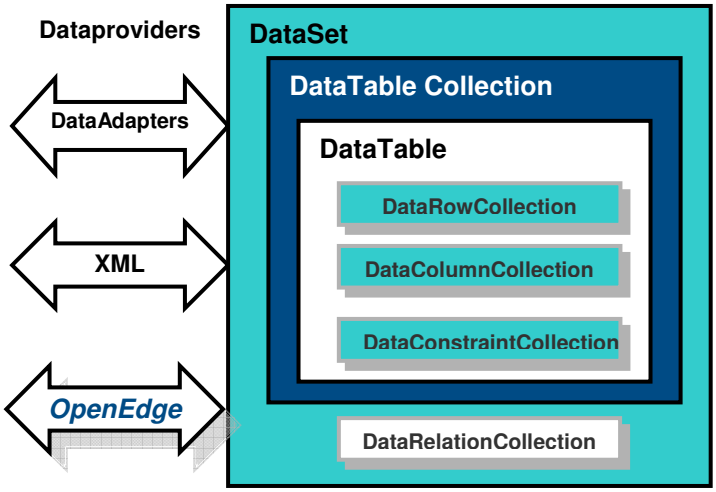
5.4.1 Definitie

De Progress ProDataSet is één op één te converteren naar een ADO.NET DataSet (en vice versa), zodat gegevens uit een Progress database bruikbaar zijn in een .NET omgeving.
ADO.NET is een evolutie van ActiveX Data Objects (ADO) en is het deel van het .NET Framework dat gegevenstoegang mogelijk maakt. Door middel van een set classes kan toegang worden verkregen tot XML-documenten, relationele databases en ODBC en OLEDB datasources.

Het belangrijkste object uit ADO.NET is de DataSet. De DataSet heeft dezelfde structuur als een relationele database. Dit object is vergelijkbaar met de Progress ProDataSet: het bevat een in-memory kopie van data uit een database.

5.4.2 Implementatie van een DataSet

Een DataSet kan door verschillende “dataproviders” worden gevuld, zoals de ADO.NET DataAdapter, een XML bestand of de Progress Appserver (zie het figuur hieronder). In paragraaf 5.5 wordt dieper ingegaan op de interfacing tussen Progress en .NET.



Figuur 6: *ADO.NET DataSet*

Zoals in bovenstaand figuur is te zien, bestaat een DataSet uit de volgende elementen:

- DataTable Collection
  Een verzameling van DataTables, vergelijkbaar met TempTables uit de ProDataSet.
- DataRowCollection
  Hierin staat de data uit de tabel.
- DataColumnCollection
  Dit bevat de definitie van de kolommen van de tabel.
- DataConstraintCollection
  Hierin staan de definities van eventuele constraints op kolommen.
  (De ProDataSet ondersteunt niet alle mogelijke constraints uit de ADO.NET DataSet).
- DataRelationCollection
  Hierin staan de definities van de relaties tussen de tabellen uit de DataSet.

Hieronder volgen enkele aandachtspunten bij het gebruik van de ADO.NET DataSet:

- Data in DataSets wordt altijd offline (disconnected) bewerkt; dit in tegenstelling tot ADO. Nadat een DataSet is gevuld, heeft deze geen connectie meer met haar data source;
  Data in een DataSet kan op gelijke wijze worden bewerkt als een ProDataSet. Gemaakte wijzigingen in de data worden bijgehouden en kunnen eventueel worden teruggeschreven naar de database.
  In .NET (C#) kan hiervoor de method “GetChanges()” worden gebruikt. Deze method levert een DataSet op die alleen de gewijzigde records bevat.
  Ook kan de method “AcceptChanges()” worden gebruikt. Deze method voert een “commit” uit van alle gemaakte wijzigingen in de DataSet.

- Een DataSet is eenvoudig te converteren van en naar XML.
  Hiervoor zijn enkele methods beschikbaar (o.a. “ReadXML()” en “WriteXML()”).
  Door middel van XML is het mogelijk om de gegevens uit de DataSet uit te wisselen tussen verschillende componenten en platformen.
  De structuur van de DataSet (tabelnamen, kolommen, relaties etc.) worden opgeslagen in een XML Schema (XSD bestand). Hiervoor zijn de methods “ReadXmlSchema()” en “WriteXmlSchema()” beschikbaar.

- DataSets kunnen zowel statisch als dynamisch worden gecreëerd.

- De .NET gebruikersinterface is voorbereid op samenwerking met DataSets.
  Bijvoorbeeld: om data uit een DataSet te tonen, is het instellen van een “Datasource” property van een .NET DataGrid voldoende.
  DataSets beschikken over een grote hoeveelheid properties en methods die bedoeld zijn om de data te beheren die de DataSet bevat.

5.4.3 Beperkingen

Implementatie van de ADO.NET DataSet, gebaseerd op een Progress ProDataSet, heeft een beperking van deze techniek aan het licht gebracht, waarvoor tot op heden nog geen afdoende oplossing voor is gevonden.
In Progress werd een ProDataSet gedefinieerd waarbij een tabel werd opgenomen met een samengestelde primaire sleutel. De sleutel bestond uit een combinatie van twee velden uit de tabel. De tabel bevatte meerdere records.
Wanneer deze ProDataSet naar een ADO.NET DataSet werd geconverteerd, bleek de genoemde tabel slechts één record te bevatten.
Dit bleek te worden veroorzaakt doordat de tabel een samengestelde primaire sleutel bevatte.
Wanneer de sleutelgegevens werden opgevraagd van de tabel uit de ADO.NET DataSet, bleek dat de primaire sleutel van de tabel wel een samengestelde sleutel was. Deze definitie was dus correct overgenomen uit de Progress ProDataSet.
Wanneer de samengestelde primaire sleutel werd veranderd naar een enkelvoudige sleutel (in de Progress ProDataSet), trad het probleem niet op.

5.5 Interfacing tussen Progress en .NET

Zoals gezegd wordt de uitwisseling van data tussen Progress en .NET ondersteund door de Progress ProDataSet en de ADO.NET DataSet. Daarnaast dient het mogelijk te zijn om vanuit een .NET omgeving business logica uit te voeren dat zich in een Progress omgeving bevindt.
Om deze communicatie tussen de twee omgevingen te realiseren, wordt gebruik gemaakt van een zogenaamde “proxy”. Deze proxy fungeert als een vertaler tussen de omgevingen. De proxy neemt de data en communicatie vanuit de Progress of .NET omgeving in ontvangst en vertaalt deze naar begrijpelijke data voor de andere omgeving. De proxy bestaat uit een DLL (Dynamic Link Library) bestand, dat kan worden gegenereerd met behulp van de tool ProxyGen. Deze tool wordt meegeleverd met OpenEdge 10.

Nadat de proxy is gegenereerd, kan de .NET client applicatie worden geschreven. Deze applicatie kan de Progress business logica aanroepen via de proxy. De proxy

communiceert vervolgens met de Progress AppServer om de business logica uit te voeren.
Het is niet mogelijk om vanuit een .NET omgeving rechtstreeks te communiceren met de Progress business logica; deze communicatie dient altijd via een proxy én de AppServer te verlopen. Een client – server architectuur is hierbij dus niet mogelijk.

In de volgende paragrafen volgt een overzicht waarbij de communicatie tussen Progress en .NET stap voor stap wordt toegelicht.
Hierbij zijn de volgende aannames gemaakt:

- Er is een AppServer beschikbaar, waarop Progress business logica kan worden uitgevoerd;
- Er is een Progress DBMS beschikbaar, waaruit gegevens kunnen worden opgevraagd via de AppServer. In dit geval wordt ervan uitgegaan dat de Sports2000 database wordt gebruikt;
- Progress OpenEdge 10A is geïnstalleerd;
- Het .NET Framework en de ontwikkelomgeving Visual Studio .NET 2003 zijn geïnstalleerd;
- De .NET applicatie wordt geschreven in de taal C#.

5.5.1 Bouwen Progress procedures

De eerste stap is het schrijven van een (of meerdere) procedure(s) die kan (kunnen) worden aangeroepen vanuit de .NET client applicatie. Vanuit deze procedure(s) kan de Progress DBMS worden aangesproken.
Een procedure GetCustomers.p maakt bijvoorbeeld een ProDataSet “dSetCustomers“ aan die gegevens uit de tabel Customer bevat.
Wanneer het de bedoeling is om deze gegevens beschikbaar te hebben op de .NET client, wordt de ProDataSet als output parameter gedefinieerd:

```
DEFINE OUTPUT PARAMETER DATASET FOR dSetCustomers.
```

Wanneer de procedure input vanuit de .NET client heeft, wordt in de procedure een input parameter gedefinieerd (bijvoorbeeld het aantal records dat moet worden opgevraagd uit de Progress database):

```
DEFINE INPUT PARAMETER NumRecords AS INTEGER.
```

Daarnaast is het mogelijk een parameter als zowel input als output parameter te definiëren. Dit kan bijvoorbeeld nuttig zijn wanneer een .NET client gegevens uit een ProDataSet wijzigt, en deze terug wil sturen naar de Progress omgeving, om deze wijzigingen op te slaan in de database.

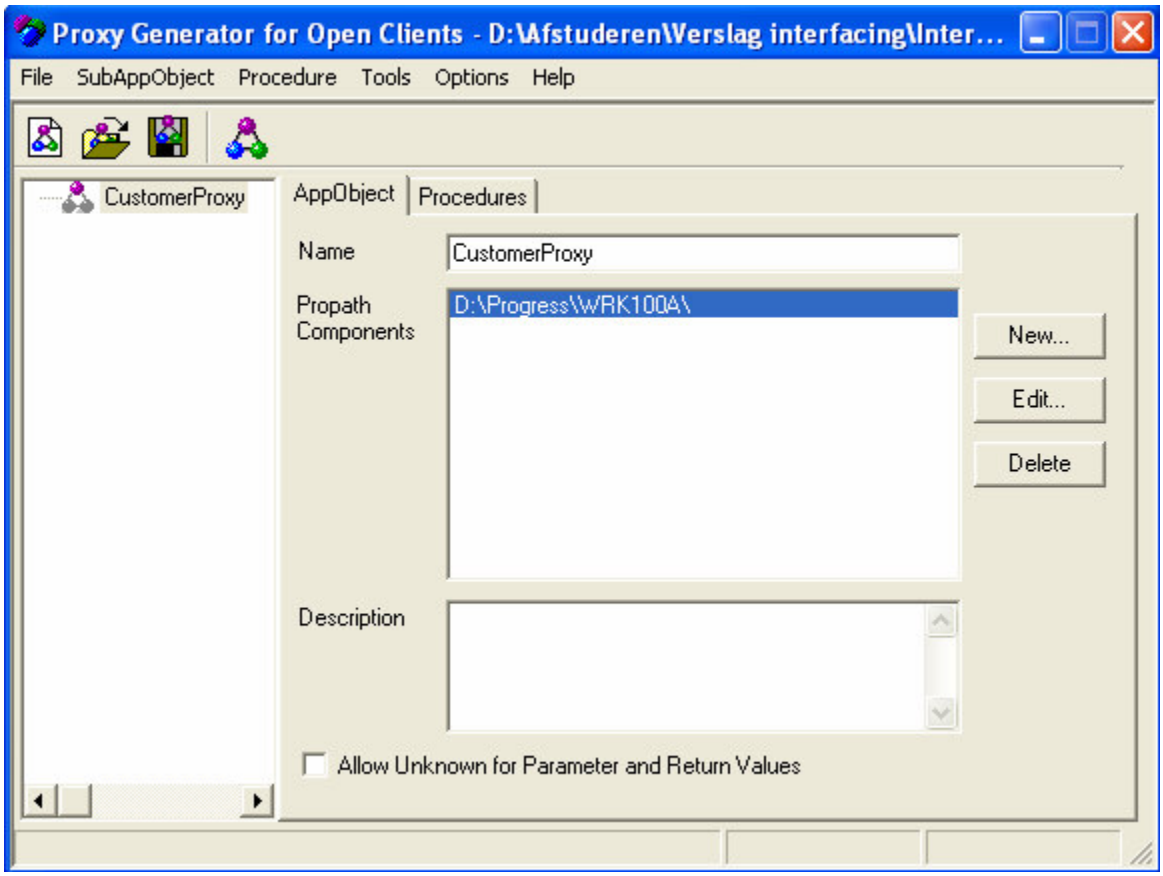
```
DEFINE INPUT-OUTPUT PARAMETER DATASET FOR dSetCustomers.
```

Wanneer de procedure geschreven is en correct functioneert, wordt deze gecompileerd. Het resultaat van het compileren (het .r bestand) dient in de “Working Directory” van de Progress AppServer te worden geplaatst.

5.5.2 Genereren proxy

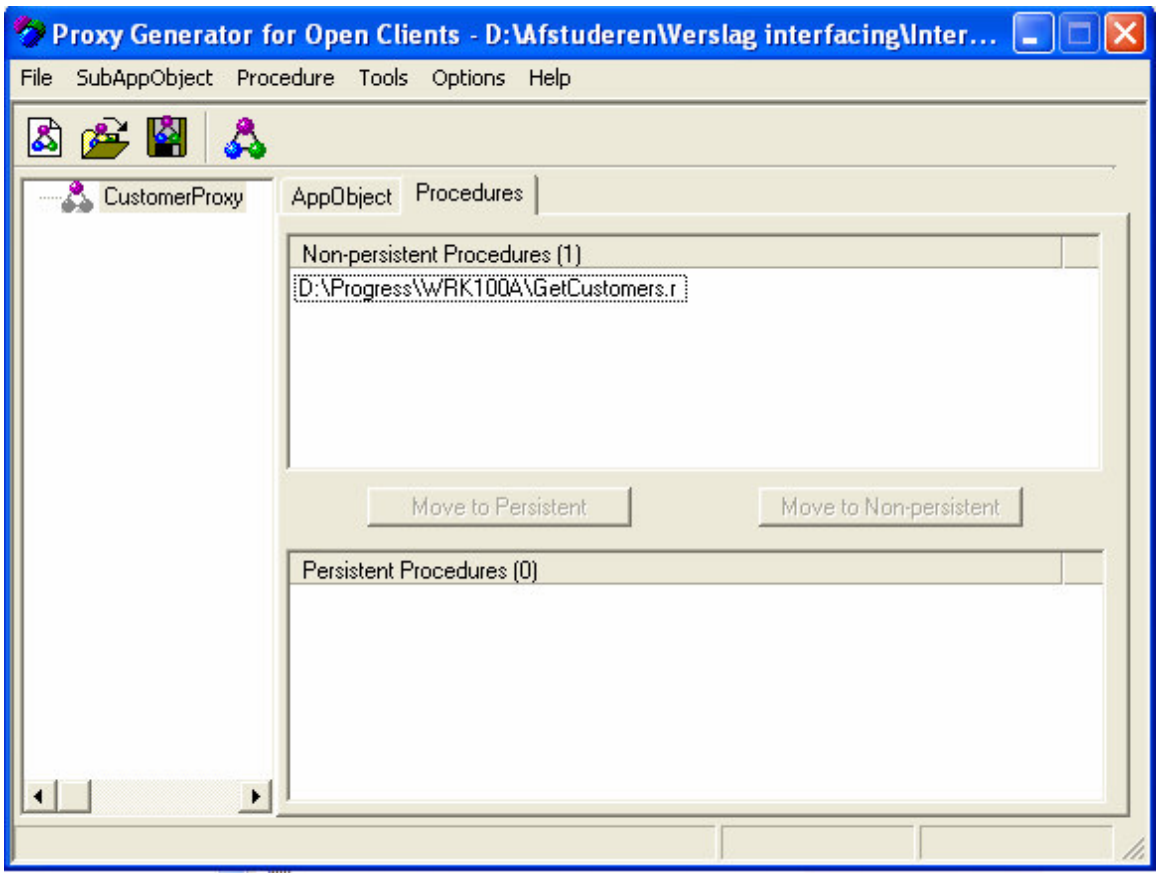
De volgende stap is het genereren van het proxy bestand dat als “vertaler” gaat fungeren. Hiervoor wordt de tool “Proxy Generator” (ProxyGen) gebruikt. Deze tool wordt standaard meegeleverd met OpenEdge 10.

Na opstarten van de tool dient eerst een naam voor de te genereren proxy te worden ingevuld. Deze naam wordt in .NET gebruikt om de proxy aan te roepen.
Daarnaast dient een map aangegeven te worden waarin de procedures staan waarvoor een proxy moet worden gegenereerd (in dit geval de “Working Directory” van de AppServer). Deze map wordt opgegeven als “Propath component”, zie onderstaande afbeelding.



Figuur 7: *ProxyGen, naam proxy en Propath Components*

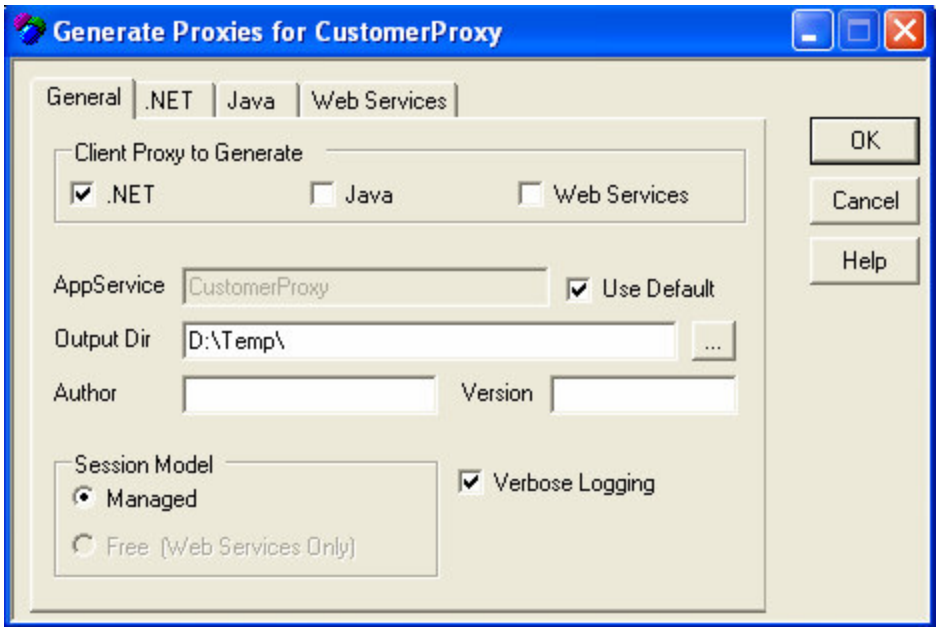
De volgende stap is het toevoegen van de procedures waarvoor een proxy dient te worden gegenereerd. Dit wordt gedaan onder de tab “Procedures”.



Figuur 8: *ProxyGen, procedures toevoegen*

Hier kunnen zowel persistent als non-persistent procedures worden toegevoegd. .r bestanden die slechts één procedure bevatten, kunnen worden toegevoegd als non-persistent procedures. .r bestanden die meerdere procedures bevatten (structured procedures) worden toegevoegd als persistent procedures.
De toegevoegde procedures dienen zich te bevinden in de “Working Directory” van de AppServer.

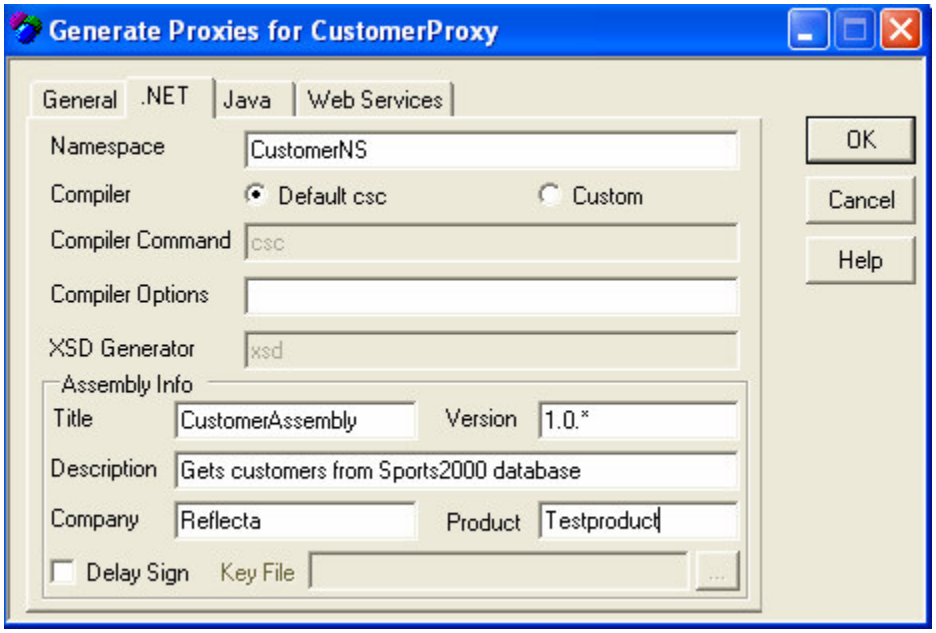
Vervolgens worden de eigenschappen voor de te genereren proxy ingesteld. Dit wordt gedaan door op  “Generate” te klikken. Het onderstaande venster wordt zichtbaar.



Figuur 9: *ProxyGen, algemene instellingen proxy*

Hier kan de omgeving worden gekozen waarvoor de proxy moet worden gegenereerd. In dit geval is dat de .NET omgeving.
De gegenereerde proxy (.dll bestand) wordt geplaatst in de opgegeven “Output Dir”. De velden “Author” en “Version” zijn niet verplicht om in te vullen. De standaard waarden van de overige instellingen hoeven niet te worden aangepast.

De “.NET” tab in dit venster dient om specifieke eigenschappen voor de .NET proxy in te stellen.



Figuur 10: *ProxyGen, instellingen .NET proxy*

In .NET worden alle objecten uit de proxy gedefinieerd als onderdeel van een *namespace* (bijvoorbeeld *namespace.procedure,* in dit geval *CustomerNS.GetCustomers)*. Het is niet verplicht om een namespace te definiëren, maar het wordt aangeraden om dit wel te doen om er zeker van te zijn dat de namen van de objecten uit de proxy uniek zijn.
Wanneer een namespace is gedefinieerd, wordt de proxy (het .dll bestand) geplaatst in de map "*Output Dir\namespace*".

Om de gegeneerde proxy uniek te maken, wordt aanbevolen om "Assembly info" in te stellen. Hier kan een titel, versie, omschrijving, bedrijfsnaam en productnaam worden opgegeven. Door dit in te stellen wordt de proxy (het .dll bestand) "ondertekend" (signed) en is hiermee uniek binnen het systeem.
Wanneer voor "Delay Sign" is gekozen, wordt de proxy niet "ondertekend" bij het compileren, maar dient dit achteraf plaats te vinden. In dit geval is het verplicht een "Key File" te selecteren, waarmee de proxy gaat worden "ondertekend".
Meer informatie over het genereren van "key files" en over het "ondertekenen" van .dll bestanden is te vinden in de documentatie van Visual Studio .NET 2003. Zie hiervoor ook de links die zijn opgenomen in het gedeelte "Geraadpleegde literatuur" van dit document.

De overige instellingen hoeven niet te worden gewijzigd of ingesteld. De proxy kan nu worden gegenereerd door op OK te klikken.
Wanneer de proxy succesvol is gegenereerd, bevindt deze zich in de opgegeven "*Output Dir\namespace*", in dit geval is dat "D:\Temp\CustomerNS". De naam van het proxy bestand is "CustomerProxy.dll".
De proxy kan vervolgens worden gebruikt in de .NET client applicatie om te kunnen communiceren met de Progress omgeving.

Bekend probleem bij het genereren van een proxy:
Wanneer de versie van ProxyGen wordt gebruikt die bij OpenEdge 10 Beta wordt meegeleverd, kan het genereren van een proxy mislukken. De reden die hiervoor wordt genoemd in de Progress Knowlegde Center (ID: P36138) is dat de Microsoft Java SDK niet is geïnstalleerd.
Deze SDK wordt echter niet meer aangeboden door Microsoft, waardoor het verkrijgen van de SDK een probleem vormt. Dit probleem is op te lossen door de versie van ProxyGen te gebruiken die bij OpenEdge 10A wordt meegeleverd. In deze versie komt dit probleem niet voor.

5.5.3 Bouwen .NET client applicatie

Nu kan de .NET client applicatie worden gebouwd die via de gegenereerde proxy communiceert met de Progress omgeving. Hieronder wordt stapsgewijs uitgelegd hoe een dergelijke applicatie kan worden gebouwd met behulp van Visual Studio .NET 2003 en C#. Hierbij wordt basiskennis van de werking van Visual Studio .NET 2003 en C# verwacht.
Wanneer bij de implementatie extra informatie benodigd is, wordt verwezen naar de documentatie die Progress beschikbaar stelt (zie hiervoor ook het deel "Geraadpleegde literatuur", achter in dit document). Daarnaast biedt de MSDN Library van Visual Studio .NET 2003 uitgebreide informatie over alle aspecten van .NET applicaties.

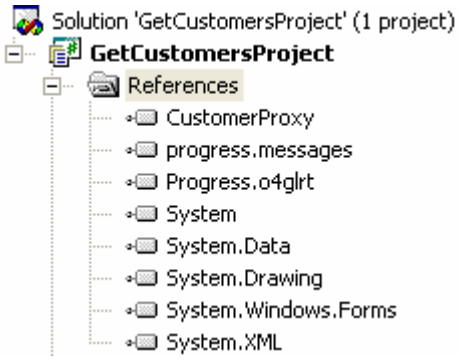
Het onderstaande voorbeeld laat stapsgewijs zien hoe vanuit .NET via de proxy verbinding wordt gemaakt met de AppServer en hoe vervolgens gegevens uit een Progress database worden opgevraagd en getoond in een .NET Windows Form.

1. Maak in Visual Studio .NET 2003 een nieuw project aan voor de .NET applicatie. In dit voorbeeld wordt het project “GetCustomersProject” genoemd en is het gebaseerd op de Windows Application template.

2. Kopieer het eerder gegenereerde proxy bestand (CustomerProxy.dll) naar de map waarin het zojuist gemaakte project zich bevindt.

3. Kopieer de OpenEdge .NET Open Client Runtime assemblies naar de map waarin het zojuist gemaakte project zich bevindt. Het is ook mogelijk deze assemblies in de .NET Global Assembly Cache te plaatsen. De Global Assembly Cache is een verzameling van alle .NET assemblies en bevindt zich in de map *Windows-install-directory*\assembly.
   De Open Client Runtime assemblies zijn benodigd om de communicatie tussen .NET en Progress te kunnen realiseren.
   Het betreft de twee onderstaande assemblies:
   - *OpenEdge-install-directory*\dotnet\Progress.o4glrt.dll
   - *OpenEdge-install-directory*\dotnet\Progress.Messages.dll

4. Voeg referenties naar de gekopieerde assemblies toe in het zojuist gemaakte project.



Figuur 11: *Referenties naar assemblies*

5. Voeg onderstaande namespaces toe aan het project.

```
using Progress.Open4GL.Proxy;
using CustomerNS;
using CustomerNS.StrongTypesNS;
```

6. Definieer globale variabelen met behulp waarvan een connectie kan worden gemaakt met de Progress AppServer.

```
private Connection AppSrvConn;
private CustomerProxy CustomerProxy1;
```

7. Maak een verbinding met de Progress AppServer.

```
AppSrvConn = new Connection
      ("AppServer://localhost:5162/testapp", "", "", "");
CustomerProxy1 = new CustomerProxy(AppSrvConn);
```

   Zie voor de benodigde argumenten bij de methode Connection() de documentatie van Visual Studio .NET 2003.

8. Definieer een DataSet en voer de procedure “GetCustomers” uit op de AppServer.

```
CustDSDataSet CustomerDS = null;
CustomerProxy1.GetCustomers
      (10, out CustomerDS, out errString);
```

   Hierbij is de klasse “CustDSDataSet” de DataSet die wordt gevuld na het uitvoeren van de Progress procedure “GetCustomers”.
   De naam “CustDSDataSet” wordt gevormd door
   “*Naam-Progress-ProDataSet*DataSet”.

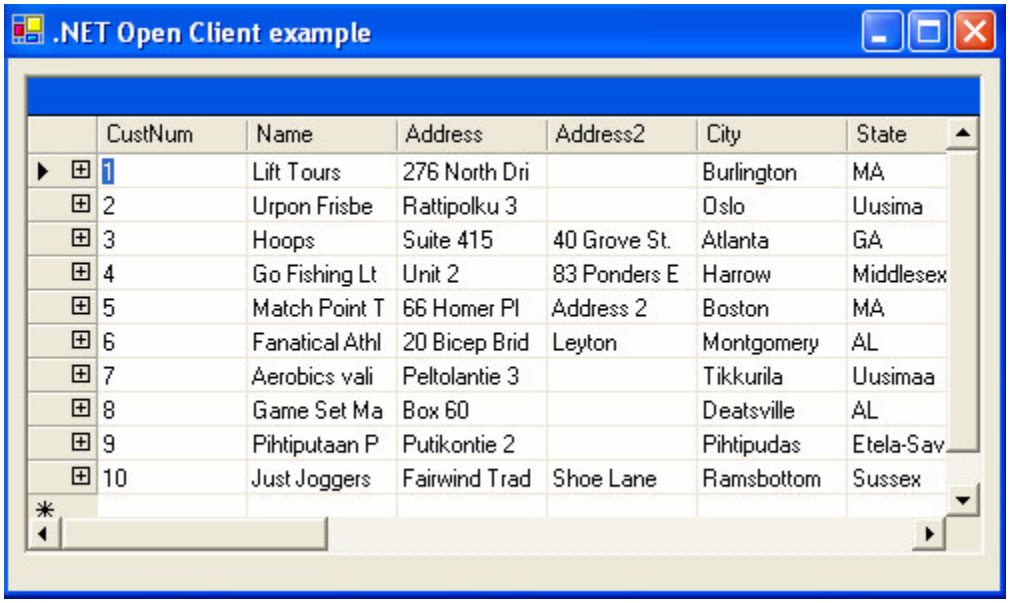
   De procedure “GetCustomers” wordt aangeroepen met de volgende parameters:
   - “10”, het aantal records dat wordt opgevraagd. Deze parameter is als input parameter gedefinieerd in de procedure “GetCustomers”;
   - “out CustomerDS”, de DataSet die de opgevraagde records bevat. Deze parameter is als output parameter gedefinieerd in de procedure “GetCustomers”;
   - “out errString”, wanneer tijdens het uitvoeren van de procedure “GetCustomers” een foutmelding optreedt, wordt deze melding aan de .NET client applicatie doorgegeven. Deze parameter is als output parameter gedefinieerd in de procedure “GetCustomers”;

9. Voeg een DataGrid met de naam “dataGrid1” toe aan het Windows Form.

10. Definieer het resultaat van de uitgevoerde procedure (DataSet CustomerDS) als datasource voor de DataGrid.

```
this.dataGrid1.SetDataBinding(CustomerDS, "ttCust");
```

11. Compileer het project en start de applicatie.
    Het resultaat is het onderstaande Windows Form dat een DataGrid bevat, waarin 10 records uit de tabel “Customers” van de Sports2000 database zijn opgenomen.



Figuur 12: *Resultaat .NET Open Client applicatie*

5.5.4 Invoeren .NET Open Client applicatie

Om een .NET Open Client applicatie te kunnen invoeren op een systeem, dienen enkele stappen uitgevoerd te worden.

1. Installeer de Microsoft .NET Framework Versie 1.1 of hoger op het systeem waar de .NET applicatie wordt geïnstalleerd.
   Dit Framework wordt gratis ter beschikking gesteld op de website van Microsoft. Toekomstige versies van Microsoft besturingsystemen zullen het .NET Framework standaard meeleveren.
   Controleren of het Framework is geïnstalleerd kan worden gedaan door te kijken of het Framework voorkomt in de lijst met geïnstalleerde software (Control Panel – Add or Remove Programs)

2. Kopieer de benodigde bestanden.
   - Kopieer de .NET client applicatie;
   - Kopieer de proxy assembly (.dll bestand) naar de map waar de .NET applicatie deze verwacht.
   - Kopieer de .NET Open Client Runtime assemblies naar de map waar de .NET applicatie deze verwacht, of naar de Global Assembly Cache.
     Het betreft de twee onderstaande assemblies:
     - *OpenEdge-install-directory*\dotnet\Progress.o4glrt.dll
     - *OpenEdge-install-directory*\dotnet\Progress.Messages.dll

3. Start de .NET Open Client applicatie.

5.6 Interfacing tussen Progress, .NET en Crystal Reports

Crystal Reports kan ADO.NET DataSets als datasource gebruiken om rapporten te genereren. Deze datasources bestaan uit XML en/of XSD bestanden, die een opbouw hebben volgens ADO.NET DataSet specificaties.
Voor meer informatie over de voordelen hiervan wordt verwezen naar paragraaf 4.4 van dit document.

Daarnaast bestaat er een versie van Crystal Reports, “Crystal Reports for Visual Studio .NET”, waarmee het mogelijk is om in .NET applicaties rapporten te maken op basis van ADO.NET DataSets. Deze versie van Crystal Reports is geïntegreerd met Visual Studio .NET 2003. In dit document wordt niet verder ingegaan op “Crystal Reports for Visual Studio .NET”. Voor meer informatie hierover wordt verwezen naar het deel “Geraadpleegde literatuur”, achter in dit document.

In de onderstaande paragrafen wordt een voorbeeld gegeven van de technische implementatie van een ADO.NET datasource voor Crystal Reports, gebaseerd op XML. Dit XML bestand wordt opgebouwd in Progress op basis van een ProDataSet en vervolgens ingelezen in Crystal Reports.

Bij dit voorbeeld zijn de volgende aannames gemaakt:

- Progress OpenEdge 10A is geïnstalleerd;
- Het .NET Framework is geïnstalleerd;
- Er is een Progress DBMS beschikbaar, waaruit gegevens kunnen worden opgevraagd (eventueel via de AppServer). In dit geval wordt de Sports2000 database gebruikt;
- Crystal Reports versie 9 of hoger is geïnstalleerd.

5.6.1 Bouwen Progress data-object

De eerste stap is het bouwen van een data-object in Progress dat de benodigde gegevens voor het rapport bevat. Omdat gebruik gaat worden gemaakt van een ADO.NET DataSet, ligt de keuze voor de hand om voor dit data-object de ProDataSet te gebruiken. Dit object is namelijk één op één te converteren tot een ADO.NET DataSet.

Hieronder wordt een voorbeeld gegeven van de definitie van een ProDataSet die gegevens uit de tabel Customer en Order bevat.

```
/* Define TempTables */
DEFINE TEMP-TABLE ttCustomer    LIKE Customer.
DEFINE TEMP-TABLE ttOrder       LIKE Order.

/* Define DataSet */
DEFINE DATASET CustOrderDataSet FOR ttCustomer, ttOrder
    DATA-RELATION CustOrderRelation FOR ttCustomer, ttOrder
        RELATION-FIELDS(custnum,custnum).

/* Define Data-Sources */
DEFINE DATA-SOURCE dsCust       FOR Customer.
DEFINE DATA-SOURCE dsOrder      FOR Order.
```

```
/* Attach DataSources */
BUFFER ttCustomer:ATTACH-DATA-SOURCE(DATA-SOURCE dsCust:HANDLE).
BUFFER ttOrder:ATTACH-DATA-SOURCE(DATA-SOURCE dsOrder:HANDLE).

/* Fill DataSet */
DATASET CustOrderDataSet:FILL().
```

5.6.2 Exporteren Progress data-object naar ADO.NET XML

De volgende stap is het data-object dat de rapportdata bevat, exporteren naar een ADO.NET DataSet. Dit wordt gedaan door de data uit het Progress data-object te exporteren naar een XML bestand dat is opgebouwd volgens de specificaties van een ADO.NET DataSet.
Een ADO.NET DataSet bestaat uit twee delen: de definitie van de DataSet en de data uit de DataSet. Wanneer een DataSet naar XML wordt geëxporteerd, wordt normaal gesproken de definitie van de DataSet (tabellen, kolommen, datatypes) opgenomen in een XML Schema Definition (XSD) bestand. De data uit de DataSet wordt opgenomen in een XML bestand, waarin een verwijzing is opgenomen naar het XSD bestand.

In dit geval wordt het schema (XSD) en de data (XML) echter in één bestand opgenomen. Dit vanwege het feit dat Crystal Reports het schema negeert wanneer een ADO.NET XML bestand wordt ingelezen waarin een verwijzing naar een schema is opgenomen. Wanneer een ADO.NET XML bestand wordt ingelezen waarin beide het schema en de data zijn opgenomen, treedt dit probleem niet op.

De opbouw van de elementen uit het ADO.NET XML bestand dient hierbij aan enkele eisen te voldoen om correct te kunnen functioneren als datasource voor Crystal Reports. Hieronder wordt daarop ingegaan.

Progress biedt functionaliteiten om een XML document te genereren. Voor een volledige beschrijving van deze functionaliteiten wordt verwezen naar de documentatie die Progress hierover ter beschikking stelt.
De belangrijkste benodigde elementen zijn:

- CREATE X-DOCUMENT: maakt een handle voor het XML bestand aan;
- CREATE X-NODEREF: maakt een handle voor een node uit het XML bestand aan;
- CREATE-NODE(): creëert een node die in het XML bestand gaat worden opgenomen;
- APPEND-CHILD(): voegt de eerder gecreëerde node aan het XML bestand toe;
- SET-ATTRIBUTE(): voegt een attribuut toe aan een node;
- SAVE(): bewaart het XML document.

Ter illustratie is hieronder een voorbeeld opgenomen waarin een node aan een XML bestand wordt toegevoegd.

```
/* Define variables */
DEFINE VARIABLE hDocument   AS HANDLE     NO-UNDO.
DEFINE VARIABLE hRoot       AS HANDLE     NO-UNDO.

/* Create XML document elements */
CREATE X-DOCUMENT hDocument.
CREATE X-NODEREF  hRoot.

/* Create root node with attribute */
hDocument:CREATE-NODE(hRoot, "xml":U, 'ELEMENT':U).
hDocument:APPEND-CHILD(hRoot).
hRoot:SET-ATTRIBUTE("xmlns:xs", "http://www.w3.org/2001/XMLSchema").
```

Na opslaan van het gecreëerde document (d.m.v. SAVE()) resulteert deze code in de onderstaande regel in het XML bestand.

```
<xml xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Eisen aan opbouw ADO.NET XML bestand

De opbouw van het ADO.NET XML bestand dient aan eisen te voldoen om correct te kunnen functioneren als datasource voor Crystal Reports.
De belangrijkste eisen zijn:

- De juiste namespaces dienen te zijn gedefinieerd in de header van het XML document. Namespaces voorzien in het definiëren van universeel unieke namen.
- Het XSD schema dient zich in het XML bestand te bevinden vóór de XML data.

Hieronder wordt een template gegeven van een ADO.NET XML bestand dat als uitgangspunt kan worden gebruikt bij implementatie van een dergelijk XML bestand. Schuin gedrukte tekst geeft aan dat dit element een variabele is die dient te worden ingevuld. Commentaar is weergegeven op regels die beginnen met “//”.

```
// XML header with namespaces
<xml xmlns:rs="urn:schemas-microsoft-com:rowset"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
     // XSD Schema definition
     <xs:schema id="schema_name" elementFormDefault="qualified"
     attributeFormDefault="qualified">
       <xs:element name="Progress_ProDataSet_name" msdata:IsDataSet="true"
       msdata:EnforceConstraints="False">
         <xs:complexType>
            <xs:choice maxOccurs="unbounded">
               // Define table table_name
               <xs:element name="table_name">
                  <xs:complexType>
                     <xs:sequence>
                         // Define table fields
                         <xs:element name="field_name" nillable="true">
                           <xs:simpleType>
                             <xs:restriction base="datatype" />
                           </xs:simpleType>
                         </xs:element>
```

```
                        // Define more table fields here (if needed)
                        // ...
                        // ...
                    // End of field list, end of table_name definition
                    </xs:sequence>
                  </xs:complexType>
               </xs:element>
               // Define more tables here (if needed)
               // ...
               // ...
             // End of table definition
          </xs:choice>
        </xs:complexType>
        // Define primary key field(s)
        <xs:key name="key_name" msdata:PrimaryKey="true">
           <xs:selector xpath="./table_name" />
           <xs:field xpath="field_name" />
        </xs:key>
        // Define more primary key fields here (if needed)
        // ...
        // ...
        // Define relations between tables
        <xs:keyref name="relation_name " refer="key_name">
           <xs:selector xpath="./table_name" />
           <xs:field xpath="field_name" />
        </xs:keyref>
        // Define more relations here (if needed)
        // ...
        // ...
      // End of XSD schema
      </xs:element>
   </xs:schema>

   // XML Data
   <rs:data>
     // Instance(s) of table_name
     <table_name>
        <field_name>field_value</field_name>
         // More field data here (if needed)
         // ...
         // ...
     </table_name>
     // More instances of table_name here (if needed )
     // ...
     // ...
   // End of XML data
   </rs:data>
// End of XML file
</xml>
```

Aandachtspunten bij implementatie van een ADO.NET XML bestand.

- Wanneer een record uit een Progress database tabel de waarde “unknown” heeft, mag deze waarde niet in het XML bestand worden overgenomen. Hiervoor dient een lege waarde (“”) te worden opgenomen, of de waarde dient in zijn geheel niét te worden opgenomen in het XML bestand.

- Het element “restriction” in het XSD schema kan worden gebruikt om eigenschappen van een veld te definiëren. Mogelijke eigenschappen zijn o.a.:
  - “base”: wordt gebruikt om het datatype van het veld te definiëren;
  - “maxLength”: wordt gebruikt om de maximale lengte van strings te definiëren;
  - “totalDigits”: wordt gebruikt om het totaal aantal decimalen vast te stellen voor velden met het datatype “decimal” of “float”;
  - “fractionDigits”: specificeert het aantal decimalen achter de komma voor velden met het datatype “decimal” of “float”.

- Vóór de definitie van het datatype van een veld dient de namespace “xs” te worden opgenomen. Bijvoorbeeld:

```
<xs:restriction base="xs:int" />
```

- Gebruik voor velden die waarden hebben die bestaan uit gehele getallen het datatype “int”.

- Gebruik voor velden die waarden hebben die bestaan uit “floating point” getallen het datatype “decimal” of “float”.

- Gebruik voor velden die waarden hebben die bestaan uit datums het datatype “date”.

- Gebruik voor velden die waarden hebben die bestaan uit tekst het datatype “string”.

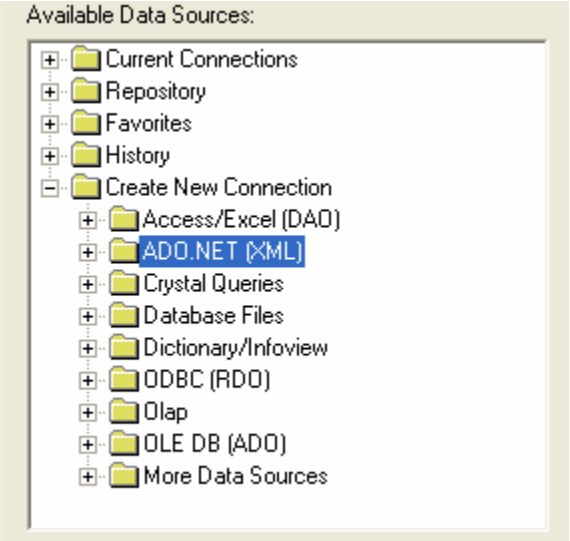
- Gebruik voor velden die waarden hebben die bestaan uit “true” of “false” het datatype “boolean”. Deze velden dienen in het XML bestand de waarde “1” of “0” te hebben. Gebruik hiervoor geen “true” of “false”.

5.6.3 Inlezen ADO.NET XML in Crystal Reports

Wanneer het ADO.NET XML bestand is aangemaakt, kan deze worden ingelezen als datasource voor een rapport in Crystal Reports. Deze paragraaf beschrijft de stappen die dienen te worden gevolgd om dit uit te voeren.

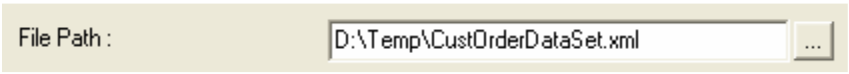
1. Start Crystal Reports versie 9 of hoger en kies voor de optie “Create a new Crystal Reports document as a blank report”.

2. Kies als datasource “ADO.NET (XML)”.



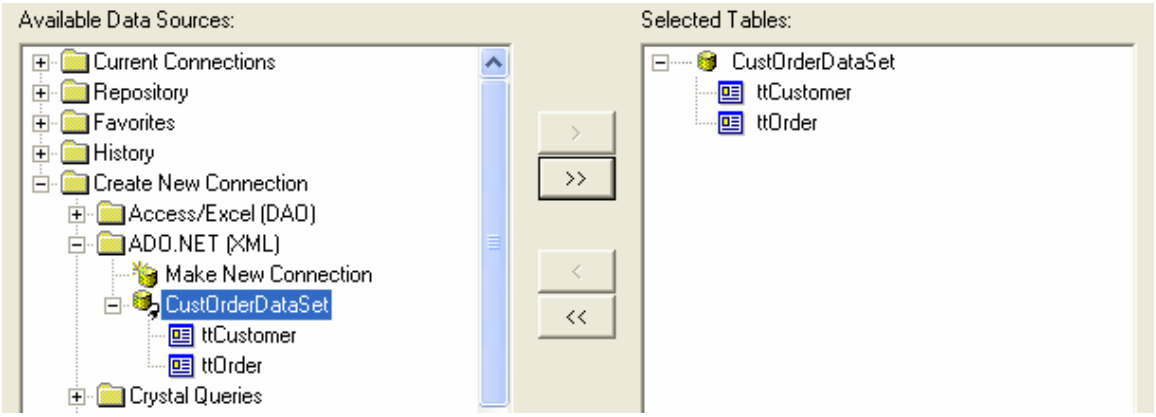
Figuur 13: *Selecteren ADO.NET (XML) datasource*

3. Selecteer het zojuist gecreëerde XML bestand.



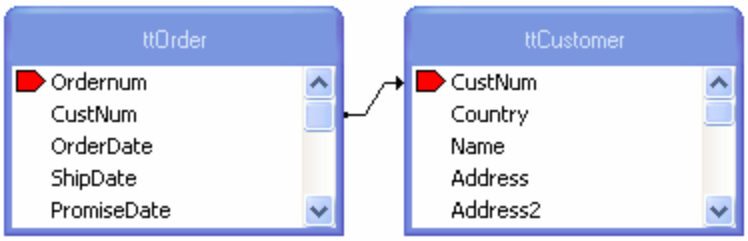
Figuur 14: *Selecteren XML bestand*

4. Selecteer de tabellen die in het rapport dienen te worden opgenomen.



Figuur 15: *Selecteren benodigde tabellen*

5. Geeft de relaties aan tussen de tabellen. Omdat deze reeds zijn overgenomen uit het XSD schema is dit (vrijwel altijd) overbodig. De juiste relaties worden reeds weergegeven.



Figuur 16: *Relaties tussen tabellen*

Wanneer de relaties niet correct zijn weergegeven, kan dit worden gecorrigeerd door de aangegeven relaties te verwijderen (“Clear links”) en deze vervolgens opnieuw aan te maken door voor “Auto-Link Tables By Key” te kiezen.

6. Nu kan het rapport worden samengesteld. De velden uit de tabellen kunnen op het rapport worden geplaatst door deze vanuit de “Field Explorer” naar het “Design” frame te slepen.

Name	Creditcard
Lift Tours	American Express
Hoopelin Pesapallo	American Express
Keilailu ja Biljardi	Master Card
Luistin ja Pyora Oy	American Express
Abc Mountain Bikes	Visa

Figuur 17: *Mogelijk resultaat rapport*

Aandachtspunten en beperkingen
Eerste implementaties van een ADO.NET XML bestand als datasource hebben enkele aandachtspunten opgeleverd (getest met Crystal Reports versie 9).

- Het attribuut “fractionDigits” uit het XSD schema wordt genegeerd in Crystal Reports. Dit attribuut definieert het aantal decimalen achter de komma van een veld dat als datatype “decimal” of “float” heeft.
  In plaats van het gedefinieerde aantal decimalen, wordt het aantal decimalen weergegeven dat in Crystal Reports is ingesteld voor getallen met een “floating point”.
  Op dit moment is hiervoor nog geen afdoende oplossing voor gevonden.

- Velden die in het XSD schema het datatype “date” hebben, worden in Crystal Reports standaard weergegeven als datatype “DateTime”. Dit datatype bestaat uit de elementen “datum” en “tijd”. Dit is niet gewenst, omdat de datums uit het XML bestand meestal geen tijdselement bevatten.
  Dit probleem kan worden opgelost door de weergave van het datatype DateTime in Crystal Reports zodanig aan te passen, dat alleen het datumelement wordt

weergegeven. (Zie hiervoor de eigenschappen voor de weergave van het datatype DateTime in Crystal Reports: File – Options – Fields – Date and time)

- De performance van het inlezen van rapportdata valt in bepaalde gevallen tegen.
  - Het inlezen van rapportdata uit één tabel verloopt zeer snel. Dit is getest met XML bestanden van diverse grootte. Zowel bij bestanden van kleiner dan één MB als bij bestanden van ruim 5 MB werd data uit één tabel binnen één seconde ingelezen (getest tot 13970 records).

  - Wanneer echter rapporten worden opgebouwd waarbij data uit verschillende tabellen benodigd is, daalt de performance sterk.
    Bij XML bestanden met een grootte tot 1 MB is de performance nog vergelijkbaar met het inlezen van data uit één tabel (getest tot bijna 2000 records).
    Wanneer echter XML bestanden worden ingelezen met een grootte van enkele MB's (getest tot ruim 5 MB) en daarbij meerdere tabellen bevatten (getest tot 5 tabellen), daalt de performance in veel gevallen sterk.
    De mate van daling van de performance verschilt per combinatie van tabellen. Op dit moment is het onduidelijk wat daarvan de reden is.
    In het slechtste geval worden nog geen 50 records per seconde ingelezen (getest op de combinatie van de tabellen Customer en Order uit de Sports2000 database).

  - In een whitepaper waarin een overzicht wordt gegeven van de functionaliteiten van Crystal Reports versie 10 belooft Business Objects een betere performance voor het inlezen van ADO.NET DataSets.

    Faster DataSet Processing
    In Crystal Reports 10, architectural changes have dramatically improved DataSet performance. Testing on a 6.8MB sample DataSet showed that the new driver is approximately 2.2 times faster than previous versions, uses 74% less private memor, and uses 63% less working set memory. Results may vary for your specific implementation.
    (Bron: Business Objects, Whitepaper "Crystal Reports 10 for .NET Feature Overview)

5.7 Conclusie

Eerste implementaties van interfacing tussen Progress en .NET hebben uitgewezen dat deze interfacing naar tevredenheid functioneert. Het is mogelijk om een .NET gebruikersinterface te implementeren voor een Progress applicatie.
De interfacing tussen Progress, .NET en Crystal Reports is technisch realiseerbaar, maar de genoemde aandachtspunten en beperkingen staan momenteel een succesvolle implementatie van een ADO.NET XML datasource in de weg. Het is mogelijk dat met Crystal Reports versie 10 deze knelpunten zijn opgelost.

Conclusie

Uit het onderzoek naar de definitie en mogelijkheden van Microsoft .NET is gebleken dat .NET niet in één enkele zin te definiëren is. Het omvat een groot scala aan producten, diensten en mogelijkheden. Samengevat kan worden gezegd dat .NET tot doel heeft om een standaard communicatiemodel te bieden en dat de twee belangrijkste componenten daarin XML en XML-webservices zijn.

Tot op heden is een grote doorbraak van .NET uitgebleven. Bedrijven hebben een afwachtende houding ten opzichte van .NET, die veroorzaakt is door onduidelijkheid over de mogelijkheden van .NET. De problemen waar .NET op dit moment mee te kampen heeft, houden een grote doorbraak nog tegen, maar wanneer het product verder wordt uitontwikkeld en zijn waarde zal tonen in de praktijk, zal een groter draagvlak ontstaan. Bedrijven en consumenten voorzien een grote toekomst voor .NET.

De mogelijkheden van .NET voor Reflecta bestaan uit de volgende punten:

- Verbetering van de gebruikersinterface van Progress applicaties.
  OpenEdge 10 ondersteunt zogenaamde Open Clients: de gebruikersinterface van een applicatie is hierbij onafhankelijk van Progress geworden. Het is mogelijk om .NET gebruikersinterfaces te bouwen, waarbij de achterliggende business logica in de Progress-omgeving wordt uitgevoerd.

- Verbetering van de opbouw van de datasource van Crystal Reports
  Door gebruik te maken van .NET DataSets kan de opbouw van datasources voor rapporten worden verbeterd.

- XML-webservices
  XML-webservices bieden mogelijkheden op het gebied van communicatie met mobiele apparaten. Voordat hier aan begonnen wordt, verdient het aanbeveling om te wachten tot de .NET-technologieën verder zjin uitontwikkeld en hun nut hebben bewezen in de praktijk.

Eerste implementaties van interfacing tussen Progress en .NET hebben uitgewezen dat deze interfacing naar tevredenheid functioneert. Het is mogelijk om een .NET gebruikersinterface te implementeren voor een Progress applicatie.
De interfacing tussen Progress, .NET en Crystal Reports is technisch realiseerbaar, maar enkele knelpunten staan momenteel een succesvolle implementatie van een.NET datasource in de weg. Het is mogelijk dat deze knelpunten in de nieuwe versie van Crystal Reports (versie 10) zijn opgelost.

Geraadpleegde literatuur

Hieronder is per hoofdstuk uit dit rapport een overzicht gegeven van de geraadpleegde literatuur. De waardering van elke informatiebron is beoordeeld met een cijfer tussen 1 en 10.

• Hoofdstuk 2: Microsoft .NET

- **Definitie .NET**

  Definitie .NET voor IT Professionals
  Waardering: 8
  http://www.microsoft.com/technet/itsolutions/net/evaluate/itpronet.mspx

  Beschrijving .NET Framework
  Waardering: 7
  http://msdn.microsoft.com/netframework/using/understanding/netframework/
  http://www.microsoft.com/netherlands/net/default.asp

  Definitie .NET volgens Ars Technica
  Waardering: 7
  http://arstechnica.com/paedia/n/net/net-1.html

  Artikel “Net wat een developer nodig heeft: .NET”
  Waardering: 6
  http://www.lysias.be/Texts/NL_DotNet.htm

  PowerPoint presentatie .NET
  Waardering: 6
  http://www.os3.nl/~roeland/011.ppt

- **XML**

  Beschrijving werking XML
  Waardering: 8
  http://www.surfnet.nl/publicaties/bulletin/01-3/h6.html

- **Hailstorm (.NET MyServices)**

  Artikel “Microsoft schuift Hailstorm op lange baan”
  Waardering: 7
  http://www.webwereld.nl/nieuws/10878.phtml

  Artikel “Microsoft decks Hailstorm project”
  Waardering: 6
  http://www.theinquirer.net/?article=3206

  Beschrijving Hailstorm
  Waardering: 7
  http://www.pcbuyersguide.com/editorials/Editorial-EndOfTheFreebies.html

- **Huidige problemen .NET**

  Artikel “.Net name ties Microsoft in knots”
  Waardering: 9
  http://zdnet.com.com/2100-1104-948838.html

Artikel “Clarifying the .NET message”
Waardering: 8
http://zdnet.com.com/2100-1107-957998.html

Artikel “Gates: Slow going for .NET”
Waardering: 7
http://news.com.com/2100-1001-946087.html

Nieuwsbericht “Microsoft alters branding of confusing .NET strategy”, 10 januari 2003
Waardering: 7
http://www.usatoday.com/tech/news/2003-01-10-microsoft-branding_x.htm

Artikel “Thought for the day: .net or .not?”
Waardering: 6
http://www.computerweekly.com/Article119191.htm

- **Voorbeelden XML-webservices**

  Microsoft case studies
  Waardering: 8
  http://www.microsoft.com/resources/casestudies/Company.asp?SearchString=L&pageMode=4

  Referenties in Nederland
  Waardering: 7
  http://www.microsoft.com/netherlands/referenties/netoplossing/default.asp

  .NET Passport
  Waardering: 6
  http://www.passport.net

  .NET Alerts
  Waardering: 7
  http://www.microsoft.com/net/services/alerts/

• Hoofdstuk 3: Standaarden en technieken van .NET

- **.NET Framework**

  Beschrijving Framework
  Waardering: 8
  http://www.microsoft.com/netherlands/msdn/aspnet/nodig.asp

  Artikel “Managed, Unmanaged, Native: What Kind of Code Is This?”
  Waardering: 7
  http://www.developer.com/net/cplus/print.php/2197621

  Technische werking Framework
  Waardering: 7
  http://www.cswl.com/whiteppr/white/microsoftnetdoc.html#intro

  .NET Framework Achitecture
  Waardering: 6
  http://www.gotmono.com/docs/architecture/architecture.html

PowerPoint Presentatie .NET Framework
Waardering: 7
http://docs.msdnaa.net/ICIS/minder_REVISED.ppt

Document ".NET Framework Achitecture"
Waardering: 8
http://media.wiley.com/product_data/excerpt/52/04712228/0471222852.pdf

- **SOAP**

  Document "Advanced SOAP"
  Waardering: 9
  http://www.perfectxml.com/ph/advancedsoap.pdf

  Beschrijving door MSDN
  Waardering: 7
  http://www.microsoft.com/netherlands/msdn/products/soap.asp

  Officiële W3 specificaties
  Waardering: 6
  http://www.w3.org/TR/SOAP/

  SOAP handleiding
  Waardering: 6
  http://www.w3schools.com/soap/default.asp

- **UDDI**

  Beschrijving door MSDN
  Waardering: 8
  http://www.microsoft.com/netherlands/msdn/products/uddi.asp

  Artikel "Dienstverlening met de 'w' van web"
  Waardering: 6
  http://www.computable.nl/artikels/archief1/d17ag101.htm

  Artikel "UDDI voor dynamische webservices"
  Waardering: 7
  http://www.r20.nl/artikel_UDDI.htm

- **WSDL**

  WSDL Essentials
  Waardering: 8
  http://www.developer.com/services/article.php/1602051

- **Beveiliging .NET**

  WS-Security
  Waardering: 8
  http://www.verisign.com/wss/wss.pdf

  Document "Security in the Microsoft .NET Framework" (onafhankelijk onderzoek)
  Waardering: 8
  http://www.foundstone.com/pdf/dotnet-security-framework.pdf

  Artikel "Webapplicaties en security"
  Waardering: 6
  http://www.apache.nl/~jvd/paper.html

Microsoft over beveiliging in .NET
Waardering: 7
http://www.microsoft.com/netherlands/net/develop/security.asp

- **Risico's**

  Document "Advanced SOAP – Some concerns about webservices"
  Waardering: 8
  http://www.perfectxml.com/ph/advancedsoap.pdf

  Abernathy, P., "Een wereld van servicegerichte integratie", Software Release Magazine, september 2003, p.7-12.
  Waardering: 6

  Widdows, C., ".NET webservices, een object te ver?", Software Release Magazine,mei 2003, p.23-27
  Waardering: 8

- **Applicaties ontwikkelen in .NET**

  Introductie C#
  Waardering: 8
  http://www.andymcm.com/csharpfaq.htm#1.3
  http://www.microsoft.com/netherlands/vstudio/vcsharp/taal.asp#1

  Visual Studio .NET 2003
  Waardering: 8
  http://www.microsoft.com/netherlands/vstudio/default.asp

  Overbeek, J.W., "C# Builder: .NET ontwikkelomgeving van Borland", Software Release Magazine, mei 2003
  Waardering: 6

• Hoofdstuk 5: Interfacing tussen Progress en .NET

- **Progress OpenEdge 10**

  Officiële documentatie door Progress Software Corporation
  Waardering: 8
  http://www.progress.com/products/documentation/start_openedge10/index.ssp

  PowerPoint Presentaties uit OpenEdge 10 Beta Training
  Waardering: 7
  http://www.progress.com/beta/openedge10a/members/presentations/index.ssp

- **Progress ProDataSet**

  OpenEdge 10A documentatie
  Waardering: 9
  http://www.progress.com/products/documentation/start_openedge10/index.ssp

  Voorbeelden implementatie ProDataSet
  Waardering: 7
  http://www.progress.com/progress/products/documentation/docs/dataset/examples.zip

- **ADO.NET**

  ADO.NET beschreven op Microsoft MSDN
  Waardering: 8
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconadonetarchitecture.asp

  Voorbeeld implementatie ADO.NET DataSet
  Waardering: 9
  http://www.net-language.com/CodeExample.aspx?i=376

  Tutorial gebruik ADO.NET
  Waardering: 7
  http://samples.gotdotnet.com/quickstart/howto/doc/adoplus/adoplusoverview.aspx

- **Progress OpenEdge 10 en .NET**

  .NET Open Client introductie
  Waardering: 8
  http://www.progress.com/progress/products/documentation/docs/dvoci/dvoci.pdf

  Documentatie .NET Open Client Development
  Waardering: 7
  http://www.progress.com/progress/products/documentation/docs/dvnet/dvnet.pdf

  Voorbeelden .NET Open Client applicaties
  Waardering: 8
  Zie *OpenEdge-install-directory*\src\samples\dotnet

- **.NET applicaties**

  Assemblies registeren d.m.v. *strong names*
  Waardering: 9
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconAssigningAssemblyStrongName.asp
  http://www.developer.com/net/cplus/article.php/3292231

  Uitgebreide informatie over alle aspecten van het ontwikkelen van .NET applicaties is te vinden in de MSDN Library van Visual Studio .NET 2003 of op MSDN online
  Waardering: 9
  http://msdn.microsoft.com

  “Ondertekenen” (signing with a strong name) van assemblies
  Waardering: 7
  http://www.developer.com/net/cplus/article.php/3292231
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconassigningassemblystrongname.asp

  Informatie over Global Assembly Cache
  Waardering: 7
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconGlobalAssemblyCache.asp

- **Crystal Reports for Visual Studio .NET**

  Informatie op Microsoft MSDN
  Waardering: 8
  http://msdn.microsoft.com/library/?url=/library/en-us/crystlmn/html/crconcrystalreports.asp?frame=true

  Business Objects .NET Zone
  Waardering: 6
  http://www.businessobjects.com/products/dev_zone/net/default.asp?ref=devzone_main

- **XSD/XML**

  XSD Schema tutorial
  Waardering: 8
  http://www.w3schools.com/schema/default.asp

  XML en de ADO.NET DataSet
  Waardering: 8
  http://samples.gotdotnet.com/quickstart/howto/doc/Xml/XMLDataSet.aspx

Bijlage 3

Rapport Definitiestudie

Voorwoord

Voor u ligt het rapport definitiestudie voor het project .NET interfacing. Dit project wordt uitgevoerd in opdracht van Reflecta Automation B.V. te Reeuwijk-Dorp en omvat het verrichten van onderzoek naar en de bouw van interfacing tussen de 4GL Progress en Microsoft .NET.
Wat betreft de bouw van interfacing is het doel van dit project tweeledig:

- Op basis van een menu-object een data- en gebruikersinterface voor de applicatie XL-Enz opleveren;
- Crystal Reports laten werken op basis van een .NET dataobject als datasource.

De definitiestudie is de eerste fase in het IAD systeemontwikkelingtraject. Hierin worden de eisen aan de data- en gebuikersinterface van XL-Enz en aan het dataobject van Crystal Reports vastgesteld en een systeemconcept ontwikkeld, wat een eerste stap in de richting van implementatie is.
Ook worden in deze fase delen van het systeemconcept geclusterd tot eenheden die kunnen dienen als zelfstandig te implementeren pilots.
Dit rapport vormt de basis voor de volgende fase van IAD, de pilotontwikkeling. Tijdens de pilotontwikkeling worden de pilots op iteratieve wijze gedetailleerd ontworpen en geïmplementeerd.

Dit document is tot stand gekomen in overleg met dhr. T. Dekker, programmeur bij Reflecta Automation.

Reeuwijk-Dorp, 20 januari 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

Samenvatting

Doel rapport
Dit rapport is het resultaat van de eerste fase van de systeemontwikkelmethode Iterative Application Development (IAD). Het doel van deze fase is om de eisen aan het menu van XL-Enz en aan het dataobject van Crystal Reports vast te stellen en een systeemconcept te ontwikkelen. Delen van dit systeemconcept worden geclusterd tot eenheden die dienen als zelfstandig te implementeren pilots.
Dit rapport vormt de basis voor de volgende fase van IAD, de pilotontwikkeling. In deze fase worden de pilots op iteratieve wijze gedetailleerd ontworpen en geïmplementeerd.

Resultaten fase definitiestudie
De eisen aan beide toepassingen zijn geprioriteerd door middel van de *MoSCoW* analyse. Deze analyse verdeelt de eisen in Must haves, Should haves, Could haves en Would like to but probably cannot haves. De volgende pilots zijn in dit document gedefinieerd en worden in onderstaande volgorde iteratief ontwikkeld en geïmplementeerd:

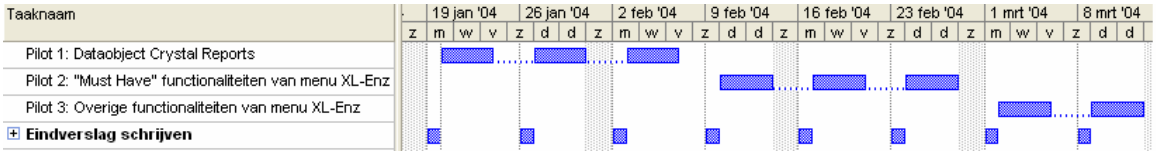
- **Pilot 1: Dataobject Crystal Reports**
  - Een gebruiker is in staat om Crystal Reports een rapport te laten genereren op basis van een .NET dataobject als datasource;
  - Het dataobject kan gegevens uit een bestaande Progress database inlezen;
  - Het dataobject is te converteren van een Progress object naar een ADO.NET object;
  - Het dataobject wordt opgebouwd op de server als een Progress object, vervolgens geconverteerd naar een ADO.NET object en daarna verstuurd naar de client.

- **Pilot 2: "Must Have" functionaliteiten van menu XL-Enz**
  - Een gebruiker heeft de beschikking over een grafisch menu waarmee programma's van XL-Enz kunnen worden opgestart;
  - Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer;
  - Een gebruiker kan het menu afsluiten;
  - Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database;
  - Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa;
  - Het menu-dataobject wordt opgebouwd op de server als een Progress object en vervolgens doorgegeven aan de client, waar het object wordt geconverteerd naar een ADO.NET object;
  - Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven;
  - Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd.

- **Pilot 3: Overige functionaliteiten van menu XL-Enz**
  - Een gebruiker kan menu-items en programma's uit het menu knippen, slepen, kopiëren, plakken, hernoemen, sorteren, toevoegen en verwijderen;
  - Zolang de gebruiker de wijzigingen in de opbouw van het menu niet opslaat, worden de wijzigingen bijgehouden in het ADO.NET object dat zich op de client bevindt;
  - Een gebruiker beschikt over een "Favorietenfolder" waarin de gebruiker naar eigen inzicht programma's kan plaatsen, bewerken en verwijderen;
  - Een gebruiker kan gemaakte wijzigingen in de opbouw van het menu opslaan;
  - Bij het opslaan van wijzigingen in de opbouw van het menu worden de wijzigingen opgeslagen in de Progress database;
  - Een gebruiker kan wijzigingen in het menu die nog niet zijn opgeslagen ongedaan maken;
  - Het menu bevat autorisatie per gebruikersgroep en per gebruiker;
  - Een gebruiker kan alleen die programma's starten en die bewerkingen uitvoeren waar hij autorisatie voor heeft;
  - Een beheerder (administrator) ziet de menu's van alle gebruikersgroepen en kan deze wijzigen;
  - Het menu wordt opgebouwd en weergegeven in minder dan 10 seconden.

Er is geconcludeerd dat het menu van XL-Enz en het dataobject voor Crystal Reports kan worden geïmplementeerd en ingevoerd op de huidige fysieke technische structuur. Om een .NET omgeving te creëren op de client dient echter wel het .NET Framework te worden geïnstalleerd op de client. Deze is gratis te downloaden van de website van Microsoft.

<u>Planning komende fase</u>
Op basis van een schatting is onderstaand overzicht gemaakt voor de volgende fase in het ontwikkeltraject, de pilotontwikkeling.
De fase pilotontwikkeling heeft een geplande doorlooptijd van 8 weken: van 19 januari 2004 tot en met 12 maart 2004.



<u>Voortgang project</u>
Na goedkeuring van dit document door de begeleider en/of de opdrachtgever kan de fase pilotontwikkeling worden gestart.
Op dit moment ligt het project enkele dagen voor op de oorspronkelijke planning. De planning voor de fase pilotontwikkeling is hierop aangepast.

1. Inleiding

De interfacing tussen Progress en Microsoft .NET wordt ontworpen en geïmplementeerd volgens de systeemontwikkelmethode Iterative Application Development (IAD). De definitiestudie vormt de eerste fase van deze methode.

Het doel van deze fase is om de eisen aan het menu van XL-Enz en aan het dataobject van Crystal Reports vast te stellen en een systeemconcept te ontwikkelen. Delen van dit systeemconcept worden geclusterd tot eenheden die dienen als zelfstandig te implementeren pilots.
Dit rapport vormt de basis voor de volgende fase van IAD, de pilotontwikkeling. In deze fase worden de pilots op iteratieve wijze gedetailleerd ontworpen en geïmplementeerd.

De opbouw van dit rapport is als volgt. Als eerste worden in hoofdstuk 2 de systeemeisen gedefinieerd. Op basis van deze eisen wordt in hoofdstuk 3 een systeemconcept ontwikkeld. Vervolgens wordt in hoofdstuk 4 de technische structuur geanalyseerd. Als laatste wordt in hoofdstuk 5 een pilotplan opgesteld. Dit plan bestaat uit een geprioriteerde lijst van pilots, die sequentieel ontwikkeld en ingevoerd gaan worden tijdens de fase pilotontwikkeling.
Bijlage A bevat een plan van aanpak voor de fase definitiestudie, waarin het eerder opgestelde plan van aanpak wordt gedetailleerd voor deze fase.

2. Systeemeisen

2.1 Inleiding

Het doel van dit hoofdstuk is om een geprioriteerde lijst van systeemeisen op te stellen. Deze eisen dienen een juiste weergave te zijn van de gewenste eigenschappen van het menu van XL-Enz en het dataobject voor Crystal Reports. De lijst van eisen vormt de basis voor alle volgende ontwerpactiviteiten. Bij het uitvoeren van de acceptatietest kunnen de systeemeisen als acceptatiecriteria dienen om te verifiëren of aan de eis is voldaan.
Bij het opstellen van de eisen is onderscheid gemaakt in eisen aan het menu van XL-Enz en eisen aan het dataobject van Crystal Reports. Dit vanwege het feit dat dit twee los van elkaar staande toepassingen zijn.

2.2 Eisen aan menu XL-Enz

De eisen aan het menu van XL-Enz zijn onderverdeeld in functionele en niet-functionele eisen. Ze worden geprioriteerd door middel van het uitvoeren van een *MoSCoW* analyse.
De *MoSCoW* analyse kent de volgende indeling in prioriteiten:

- Must Have;
- Should Have;
- Could Have;
- Would like to but probably cannot have.

Het menu van XL-Enz dient minimaal aan de eisen te voldoen die als Must Have zijn geprioriteerd. De eisen uit de overige categorieën worden naar gelang hun prioriteit geïmplementeerd, indien daarvoor de benodigde tijd beschikbaar is.

2.2.1 Functionele eisen

De onderstaande tabel bevat een opsomming van de functionele eisen aan het menu van XL-Enz, geprioriteerd door middel van een *MoSCoW* analyse. De functionele eisen beschrijven de functies die de gebruiker met het menu kan uitvoeren.

ID	Omschrijving	MoSCoW
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma's van XL-Enz kunnen worden opgestart	M
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer	M
MF3	Een gebruiker kan het menu afsluiten	M
MF4	Een gebruiker kan menu-items en programma's uit het menu knippen, slepen, kopiëren, plakken, hernoemen, sorteren, toevoegen en verwijderen	S
MF5	Een gebruiker kan gemaakte wijzigingen in de opbouw van het menu opslaan	S
MF6	Een gebruiker kan wijzigingen in het menu die nog niet zijn opgeslagen ongedaan maken	W

2.2.2 Niet-functionele eisen

De onderstaande tabel bevat een opsomming van de niet-functionele eisen aan het menu van XL-Enz, geprioriteerd door middel van een *MoSCoW* analyse. Niet-functionele eisen zijn eisen die niet inhoudelijk met het systeem te maken hebben, maar randvoorwaarden stellen aan de manier waarop de functionele eisen dienen te worden geïmplementeerd.

ID	Omschrijving	MoSCoW
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database	M
MN2	Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa	M
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven	M
MN4	Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd	M
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress dataobject en vervolgens doorgegeven aan de .NET client, waar het object wordt geconverteerd naar een ADO.NET dataobject	M
MN6	Zolang de gebruiker de wijzigingen in de opbouw van het menu niet opslaat, worden de wijzigingen bijgehouden in het ADO.NET dataobject dat zich op de .NET client bevindt	S
MN7	Bij het opslaan van wijzigingen in de opbouw van het menu worden de wijzigingen opgeslagen in de Progress database	S
MN8	Het menu wordt opgebouwd en weergegeven in minder dan 10 seconden	S
MN9	Het menu bevat autorisatie per gebruikersgroep en per gebruiker	C
MN10	Een gebruiker kan alleen die programma's starten en die bewerkingen uitvoeren waar hij autorisatie voor heeft	C
MN11	Een beheerder (administrator) ziet de menu's van alle gebruikersgroepen en kan deze wijzigen	C
MN12	Een gebruiker beschikt over een "Favorietenfolder" waarin de gebruiker naar eigen inzicht programma's kan plaatsen, bewerken en verwijderen	C

2.3 Eisen aan dataobject Crystal Reports

De eisen aan het dataobject dat als datasource dient voor Crystal Reports worden op gelijke wijze opgesteld als hierboven.

2.3.1 Functionele eisen

ID	Omschrijving	MoSCoW
DF1	Een gebruiker is in staat om Crystal Reports een rapport te laten genereren op basis van een .NET dataobject als datasource	M

2.3.2 Niet-functionele eisen

ID	Omschrijving	MoSCoW
DN1	Het dataobject kan gegevens uit een bestaande Progress database inlezen	M
DN2	Het dataobject is te converteren van een Progress object naar een ADO.NET object	M
DN3	Het dataobject wordt opgebouwd op de server als een Progress object, vervolgens geconverteerd naar een ADO.NET object en daarna verstuurd naar client	M
DN4	De tijd die benodigd is om een rapport op te bouwen op basis van een .NET dataobject als datasource, dient minstens gelijk te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject	M

3. Systeemconcept

3.1 Inleiding

Op basis van de vastgestelde systeemeisen wordt in dit hoofdstuk een systeemconcept ontworpen. Het concept is de basis voor alle verdere ontwikkeling in de volgende fase van IAD, de pilotontwikkeling.
Bij het ontwerpen van het systeemconcept spelen alleen de functionele systeemeisen een rol. In dit onderdeel wordt gedefinieerd wát het systeem dient te kunnen. In het volgende hoofdstuk, technische structuur, worden de niet-functionele eisen meegenomen in het ontwerp. Hierbij wordt ingegaan op hoé de functionele eisen dienen te worden geïmplementeerd.
Het systeemconcept wordt gemodelleerd met behulp van technieken van Unified Modeling Language (UML).
De opbouw van dit hoofdstuk is als volgt. Als eerste worden in paragraaf 3.2 de functionele systeemeisen van het menu van XL-Enz vertaald naar UML use-cases. Deze use-cases vormen de basis voor het UML use-case-diagram in paragraaf 3.3. In de paragrafen 3.4 worden de use-cases voor het dataobject voor Crystal Reports ontworpen. In paragraaf 3.5 wordt een klassediagram voor het menu samengesteld. Als laatste wordt in paragraaf 3.6 het concept beschreven voor de ontwikkeling van het dataobject voor Crystal Reports.

3.2 Use-cases menu XL-Enz

De functionele systeemeisen vormen de basis voor de use-cases en een use-case diagram, waarin de interactie tussen de gebruiker en het systeem wordt gemodelleerd. Use-cases dienen als basis voor het verdere systeemontwerp.
Elke use-case bestaat uit een tekstuele beschrijving van de functie die een gebruiker kan uitvoeren met het systeem.

Naam:	Menu initialiseren
Aannamen:	XL-Enz is zojuist opgestart, gebruiker is ingelogd
Beschrijving:	Het menu wordt opgebouwd met daarin menu-items van XL-Enz. De gebruiker ziet alle programma's die tot zijn gebruikersgroep behoren
Uitzonderingen:	Geen
Resultaat:	Het menu wordt getoond
Dekt systeemeis:	MF1

Naam:	Menu navigeren
Aannamen:	Het menu wordt getoond
Beschrijving:	1. Een gebruiker dubbelklikt op de naam van een menu-item; 2. De inhoud (submenu's, programma's) van het gekozen menu-item wordt getoond of verborgen
Uitzonderingen:	Geen
Resultaat:	De inhoud van een menu-item wordt getoond of verborgen
Dekt systeemeis:	MF2

Naam:	Programma uit menu starten
Aannamen:	Het menu wordt getoond
Beschrijving:	1. De gebruiker navigeert door het menu totdat de naam van het betreffende programma is gevonden; 2. De gebruiker geeft aan het programma te willen starten; 3. Het systeem controleert of de gebruiker geautoriseerd is om het programma te starten; 4. Het programma wordt gestart.
Uitzonderingen:	De gebruiker is niet geautoriseerd om het programma te starten. Het systeem geeft hiervan een melding en start het programma niet
Resultaat:	Het betreffende programma is gestart en wordt getoond op het scherm
Dekt systeemeis:	MF1

Naam:	Menu afsluiten
Aannamen:	Menu wordt getoond
Beschrijving:	1. De gebruiker geeft aan het menu te willen afsluiten; 2. Wanneer het menu niet opgeslagen wijzigingen bevat, vraagt het systeem of de wijzigingen opgeslagen dienen te worden; Nadat de gebruiker een keuze heeft gemaakt, wordt het menu afgesloten; 3. Wanneer er geen wijzigingen zijn die niet opgeslagen zijn, vraagt het systeem om een bevestiging om het menu af te sluiten. Na bevestiging wordt de gebruiker uitgelogd en wordt het menu afgesloten.
Uitzonderingen:	Geen
Resultaat:	Het menu is afgesloten
Dekt systeemeis:	MF3

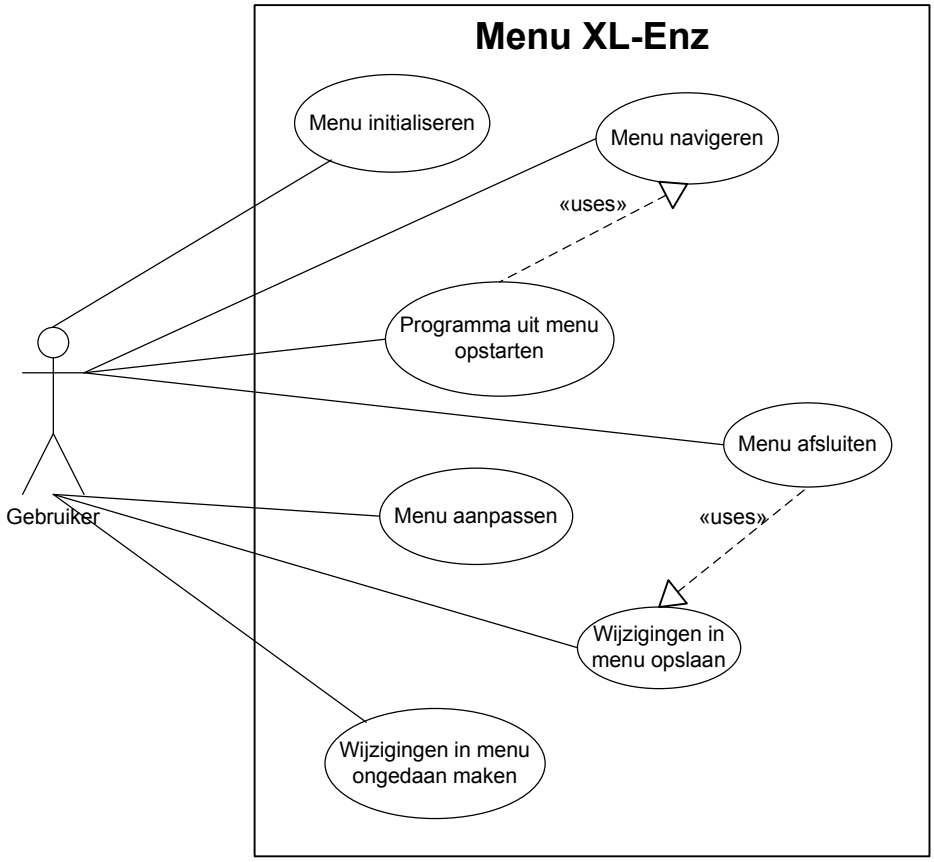
Naam:	Menu aanpassen
Aannamen:	Menu wordt getoond
Beschrijving:	1. De gebruiker kiest de gewenste aanpassing (knippen, slepen, kopiëren, plakken, hernoemen, sorteren, toevoegen of verwijderen) van een menu-item; 2. Het systeem controleert of de gebruiker geautoriseerd is om de aanpassing uit te voeren; 2. De gebruiker voert de aanpassing uit en bevestigt deze; 3. Het menu geeft het aangepaste menu-item weer.
Uitzonderingen:	Aanpassing is niet toegestaan: 1. Het systeem geeft een melding dat de aanpassing niet is toegestaan; 2. De gebruiker wijzigt de aanpassing of annuleert deze.
Resultaat:	Het menu is aangepast en wordt correct weergegeven
Dekt systeemeis:	MF4

Naam:	Wijzigingen in menu opslaan
Aannamen:	Menu wordt getoond en één of meer menu-items zijn gewijzigd
Beschrijving:	1. De gebruiker geeft aan wijzigingen in het menu op te willen slaan; 2. Het systeem vraagt een bevestiging; 3. Na bevestiging slaat het systeem de wijzigingen op in de database; 4. Het systeem meldt dat de wijzigingen zijn opgeslagen.
Uitzonderingen:	Opslaan mislukt: Het systeem geeft een melding dat het opslaan in de database niet is gelukt; het opslaan wordt geannuleerd
Resultaat:	De wijzigingen die de gebruiker in het menu heeft gemaakt, zijn opgeslagen in de database
Dekt systeemeis:	MF5

Naam:	Wijzigingen in menu ongedaan maken
Aannamen:	Menu wordt getoond en één of meer menu-items zijn gewijzigd
Beschrijving:	1. De gebruiker geeft aan wijzigingen in het menu ongedaan te willen maken; 2. Het systeem vraagt om een bevestiging; 3. Na bevestiging maakt het systeem de wijzigingen ongedaan.
Uitzonderingen:	Geen
Resultaat:	Het menu wordt weergegeven in de vorm zoals deze is opgeslagen in de database
Dekt systeemeis:	MF6

3.3 Use-case-diagram menu XL-Enz

In het use-case-diagram op de volgende pagina wordt het systeem weergegeven door een rechthoek waarbinnen de use-cases getekend worden. Buiten het systeem staat de actor Gebruiker, die gebruik maakt van de functionaliteiten van het systeem.
Tussen de use-cases onderling zijn enkele <<uses>> relaties opgenomen. Deze relaties geven aan dat een bepaalde use-case gebruik maakt van de functionaliteiten van een andere use-case.
Het diagram voegt niet veel informatie toe, maar geeft een overzicht van de mogelijkheden die het menu aan de gebruiker biedt. Voor details van de use-cases wordt verwezen naar de tekst uit de use-casebeschrijvingen.



Figuur 1: *Use-case-diagram menu XL-Enz*

3.4 Use-case dataobject Crystal Reports

De implementatie van een .NET dataobject als datasource voor Crystal Reports biedt voor de gebruiker geen nieuwe functionaliteiten. Momenteel is de gebruiker reeds in staat om op basis van een XML bestand in Crystal Reports een rapport te genereren. Dit project heeft alleen gevolgen voor de opbouw van het dataobject en niet voor de functionaliteiten voor de gebruiker.
De enige use-case voor dit onderdeel van het project is daardoor de onderstaande.

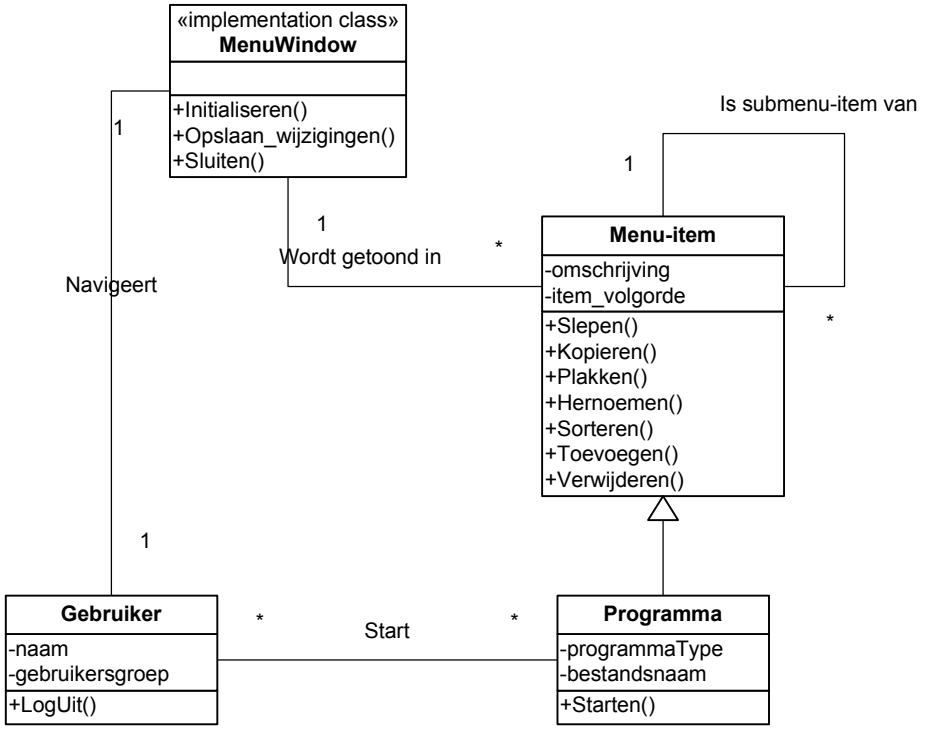
Naam:	Rapport genereren in Crystal Reports
Aannamen:	De gebruiker heeft aangegeven welke informatie het rapport dient te bevatten en heeft opdracht gegeven het rapport te genereren
Beschrijving:	Het systeem zoekt in de database de benodigde gegevens om het rapport te kunnen genereren en geeft deze door aan de gebruiker
Uitzonderingen:	Geen
Resultaat:	Het rapport wordt getoond
Dekt systeemeis:	DF1

Omdat hier sprake is van slechts één use-case is besloten het use-case-diagram voor dit gedeelte van het project achterwege te laten. Dit diagram zou namelijk bestaan uit slechts één actor die deelneemt in slechts één use-case.

3.5 Klassediagram menu XL-Enz

In een klassediagram wordt de statische structuur van een systeem weergegeven. Het diagram toont de elementen waaruit het systeem bestaat en hun onderlinge relaties. Het klassediagram bevat de klassen in het systeem, hun attributen, operaties en associaties. Het diagram vormt de basis voor het verdere ontwerp van het menu tijdens de pilotontwikkeling.

Om tot een klassediagram te komen, zijn eerst de mogelijke kandidaat-klassen geïnventariseerd door alle zelfstandige naamwoorden uit de use-cases te identificeren. Hierna is een beslissing genomen of de kandidaat-klasse al dan niet wordt opgenomen als een klasse.
Vervolgens zijn de associaties, attributen en operaties binnen het diagram gedefinieerd. Dit resulteerde in het onderstaande klassediagram.



Figuur 2: *Klassediagram menu XL-Enz*

Het diagram is gebaseerd op de bestaande opbouw van het menu.
De klasse Menu-item heeft een associatie met zichzelf, omdat een menu-item een submenu-item kan zijn van een ander menu-item.
De klasse Programma is een specialisatie van de klasse Menu-item. Een Menu-item is een Programma wanneer het attribuut “is_submenu_van” van Menu-item de waarde “0” heeft.

In de onderstaande modeldictionary worden alle termen uit het klassediagram gedefinieerd.

Modeldictionary bij klassediagram menu XL-Enz	
Term	**Omschrijving**
Gebruiker	Representatie binnen het systeem van een reëel persoon
Gebruiker.Gebruikersgroep	De gebruikersgroep waar de gebruiker toe behoort
Gebruiker.LogUit()	Het uitloggen van een gebruiker wanneer deze het menu afsluit
Gebruiker.Naam	De gebruikersnaam van de gebruiker
Is_submenu-item_van	Een menu-item is een submenu-item van precies één ander menu-item en een menu-item kan nul of meer submenu-items bevatten
Menu-item	Representatie binnen het systeem van een item uit het menu
Menu-item.Hernoemen()	Het wijzigen van de naam van een menu-item of van een programmanaam
Menu-item.Item_volgorde	De waarde van de plaats van het menu-item in het menu
Menu-item.Kopieren()	Het kopiëren van een menu-item of programma naar een ander deel van het menu
Menu-item.Omschrijving	De tekstuele omschrijving van het menu-item
Menu-item.Plakken()	Het plaatsen van een menu-item of programma dat zich in het geheugen van de computer bevindt
Menu-item.Slepen()	Het verplaatsen van een menu-item of programma naar een ander deel van het menu
Menu-item.Sorteren()	Het sorteren van menu-items en programma's op alfabetische volgorde
Menu-item.Toevoegen()	Het toevoegen van een menu-item of programma aan het menu
Menu-item.Verwijderen()	Het verwijderen van een menu-item of programma uit het menu
MenuWindow	Representatie van een verzameling menu-items
MenuWindow .Sluiten()	Het afsluiten van het menu
MenuWindow.Initialiseren()	Het opstarten en opbouwen van het menu op het scherm
MenuWindow.Opslaan_wijzigingen()	Het opslaan in de database van de gemaakte wijzigingen in het menu
Navigeert	Eén gebruiker navigeert door precies één menu-window
Programma	Een programma van XL-Enz dat bij een menu-item behoort en kan worden opgestart
Programma.Bestandsnaam	De naam van het bestand dat dient te worden opgestart wanneer het programma wordt geactiveerd
Programma.ProgrammaType	De omschrijving van het type programma
Programma.Starten()	Het opstarten van een programma uit het menu
Start	Een gebruiker start nul of meer programma's uit het menu. Een programma kan worden gestart door meerdere gebruikers
Wordt getoond in	Een menu-item of programma wordt getoond in precies één menu-window. Een menu-window toont nul of meer menu-items of programma's

3.6 Systeemconcept dataobject Crystal Reports

Omdat het reeds vrijwel geheel bekend is hoe het dataobject voor Crystal Reports zal worden geïmplementeerd, is hieronder een beschrijving opgenomen van de wijze waarop het object wordt gegenereerd.

- De gebruiker geeft het systeem opdracht een rapport te genereren;
- De Progress AppServer ontvangt de aanvraag en stelt een query samen;
- De query wordt op de Progress DBMS uitgevoerd;
- De Progress DBMS verzamelt op basis van de query de benodigde data;
- De Progress DBMS verstuurt de data naar de AppServer;
- De AppServer maakt een ProDataSet-object aan dat de ontvangen data bevat;
- Het ProDataSet-object wordt geconverteerd naar een ADO.NET DataSet-object, in de vorm van een ADO.NET XML bestand;
- Het ADO.NET XML bestand wordt naar de client verstuurd;
- Het ADO.NET XML bestand wordt ingelezen in Crystal Reports.

Tijdens de pilotontwikkeling zal deze beschrijving van het systeemconcept worden uitgewerkt in UML diagrammen.

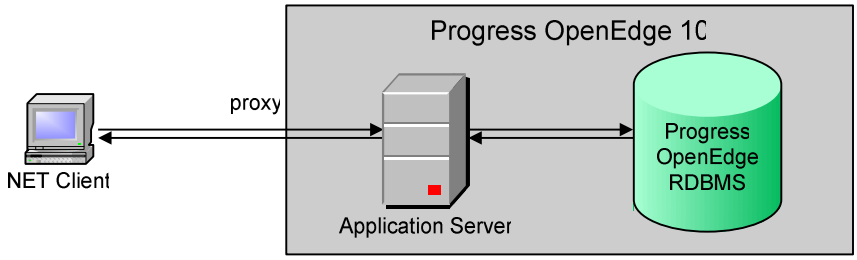
4. Technische structuur

4.1 Inleiding

In dit hoofdstuk wordt beschreven of en hoe de huidige technische structuur gebruikt kan worden om het systeemconcept te implementeren. De niet-functionele systeemeisen die gedefinieerd zijn in hoofdstuk 2 spelen hierbij een grote rol. Er zijn namelijk zeer specifieke eisen gesteld aan de technische structuur. Dit hoofdstuk beschrijft of en hoe aan deze eisen kan worden voldaan.

4.2 Technische architectuur

De applicatie XL-Enz is volledig geschreven in de 4GL Progress. XL-Enz maakt gebruik van een gelaagde architectuur: de client communiceert met de Progress OpenEdge Application Server, die op zijn beurt communiceert met de Progress OpenEdge DBMS. Deze omgeving biedt te weinig mogelijkheden om het menu van XL-Enz aan de gestelde functionele eisen te kunnen laten voldoen. Daarom wordt gebruik gemaakt van de Microsoft .NET omgeving, waarbinnen het mogelijk is om een menu op te bouwen volgens de gestelde eisen.
OpenEdge 10 biedt mogelijkheden om de Progress omgeving te laten communiceren met een .NET omgeving. Zo kunnen dataobjecten uit Progress via een proxy één op één worden geconverteerd naar .NET dataobjecten (en vice versa). Ook is het mogelijk om vanuit de .NET client business logica uit te laten voeren dat zich op de Progress Application Server bevindt.
Ook bij de implementatie van het dataobject voor Crystal Reports wordt gebruik gemaakt van conversie van een Progress dataobject naar een .NET dataobject (hierbij wordt echter geen gebruik gemaakt van een proxy).
De technische architectuur die hiervoor benodigd is, ziet er schematisch als volgt uit:



Figuur 3: *Technische architectuur*

Tijdens de fase pilotontwikkeling wordt de manier van conversie van dataobjecten tussen Progress en .NET gedetailleerd ontworpen.

4.3 Technische veranderingen

Het menu van XL-Enz en het dataobject voor Crystal Reports kan worden geïmplementeerd en ingevoerd op de huidige fysieke technische structuur.
Om een .NET omgeving te creëren op de client dient het .NET Framework te worden geïnstalleerd op de client. Deze is gratis te downloaden van de website van Microsoft.

5. Pilotplan

5.1 Inleiding

Het pilotplan is de richtsnoer voor verdere ontwikkeling van het systeem. Een pilotplan bestaat uit een geprioriteerde lijst pilots. Op basis van het systeemconcept worden delen van het systeem geclusterd tot zelfstandig te ontwikkelen en in te voeren eenheden, die pilots worden genoemd. Een pilot biedt na oplevering een zelfstandige, volledig operationele subset van het gehele systeem en kan daardoor als mijlpaal worden gezien.
Het pilotplan wordt bij elke iteratieslag opnieuw bekeken en eventueel aangepast.
De opbouw van dit hoofdstuk is als volgt. Als eerste wordt in paragraaf 5.2 een pilotontwikkelplan opgesteld, waarin de pilots worden gedefinieerd en geprioriteerd. Paragraaf 5.3 bevat een planning van de fase pilotontwikkeling. Vervolgens wordt in paragraaf 5.4 een pilotacceptatieplan opgesteld. Als laatste wordt in paragraaf 5.5 een globaal invoeringsplan ontworpen.

5.2 Pilotontwikkelplan

5.2.1 Pilotstructuur

Het menu van XL-Enz en het dataobject voor Crystal Reports zijn los van elkaar staande toepassingen, die dus los van elkaar kunnen worden ontwikkeld. Vanwege dit feit is ervoor gekozen om de ontwikkeling van deze toepassingen als twee pilots te definiëren.

De pilot voor het ontwikkelen van het menu wordt opgedeeld in twee pilots.

1. Ontwikkelen “Must Have” functionaliteiten van het menu;
2. Ontwikkelen overige functionaliteiten van het menu (functionaliteiten waaraan een lagere prioriteit is toegekend dan “Must Have”).

5.2.2 Prioriteiten pilots

De pilot voor het implementeren van het dataobject voor Crystal Reports wordt als eerste uitgevoerd. Dit vanwege het feit dat componenten uit deze pilot kunnen worden hergebruikt bij het ontwikkelen van het menu voor XL-Enz. Beide toepassingen maken namelijk gebruik van interfacing tussen Progress en .NET op basis van (deels) dezelfde technieken.

Vervolgens wordt de pilot ontwikkeld waarin de “Must Have” functionaliteiten van het menu van XL-Enz worden ontworpen en geïmplementeerd. Als laatste wordt de pilot ontwikkeld waarin de functionaliteiten met een lagere prioriteit worden gedekt.

Elke pilot wordt opgeleverd met een bijbehorend pilotontwikkelrapport, dat een ontwerp van de betreffende pilot bevat. Alle pilots worden sequentieel ontwikkeld volgens een iteratief proces. Bij elke iteratie wordt het pilotontwikkelrapport, indien nodig, aangepast of verfijnd. Vervolgens worden deze aanpassingen of verfijningen geïmplementeerd.
Na ontwikkeling van de pilots worden deze tegelijk ingevoerd.

Eventueel kan ervoor worden gekozen wegens tijdsdruk de laatste pilot gedeeltelijk of niet te ontwikkelen en alleen de eerste twee pilots in te voeren met eventueel een gedeelte van de laatste pilot.

5.2.3 Inhoud pilots

In deze paragraaf wordt de inhoud van de pilots nader gedefinieerd. Deze delen bestaan uit activiteiten die (achtereenvolgens) worden uitgevoerd tijdens de fase pilotontwikkeling.

Pilot 1: Dataobject Crystal Reports		
Pilotdeel	**Omschrijving**	**Dekt systeemeis**
1.1	Ontwikkelen dataobject in Progress OpenEdge 10	DN1
1.2	Converteren Progress dataobject naar .NET dataobject	DN2, DN3
1.3	Invoegen dataobject in Crystal Reports	DF1, DN4

Pilot 2: "Must Have" functionaliteiten van menu XL-Enz		
Pilotdeel	**Omschrijving**	**Dekt systeemeis**
2.1	Ontwikkelen menu-dataobject in Progress OpenEdge 10	MN1
2.2	Converteren Progress menu-dataobject naar .NET menu-dataobject	MN2, MN5
2.3	Ontwikkelen functionaliteiten menu	MN3, MF2, MF3
2.4	Menu als ActiveX object invoegen in Progress	MN4
2.5	Ontwikkelen mogelijkheid opstarten van programma's	MF1

Pilot 3: Overige functionaliteiten van menu XL-Enz		
Pilotdeel	**Omschrijving**	**Dekt systeemeis**
3.1	Ontwikkelen mogelijkheid om menu aan te passen	MF4, MN6, MN12
3.2	Ontwikkelen mogelijkheid om gemaakte wijzigingen op te slaan	MF5, MN7
3.3	Ontwikkelen mogelijkheid om gemaakte wijzigingen ongedaan te maken	MF6
3.4	Ontwikkelen autorisatie per gebruikersgroep en per gebruiker	MN9, MN10, MN11, (MN8)

5.3 Planning pilotontwikkeling

De fase pilotontwikkeling wordt doorlopen in de periode van 19 januari 2004 tot en met 12 maart 2004. Wekelijks wordt één dag ingepland om te werken aan het eindverslag. Hiermee komt het aantal beschikbare dagen voor deze fase op 32 dagen.

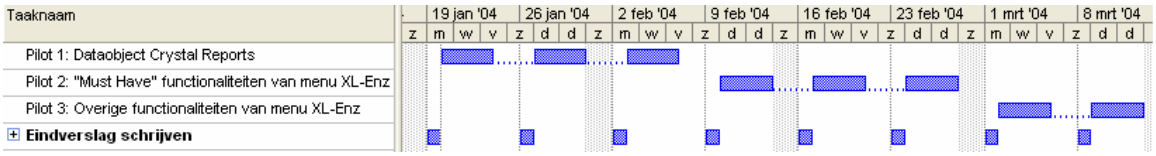
Per pilot worden (indien relevant) de volgende activiteiten uitgevoerd:

- Opstellen plan van aanpak;
- Specificeren globaal-functionele structuur pilot;
- Specificeren globaal-technische structuur pilot;
- Opstellen pilotontwikkelplan;
- Ontwerpen software-bouweenheden;
- Bouw software-bouweenheden;
- Ontwerpen handmatige procedures pilot;
- Samenstellen handleiding pilot;
- Opstellen invoeringsprocedure pilot;
- Integreren bouweenheden;
- Beoordelen en testen pilot.

Hieronder is een schatting gemaakt van de tijd die elke pilot in beslag zal nemen.

Pilot	**Geschat percentage**	**Aantal dagen**
Pilot 1: Dataobject Crystal Report	40 %	12
Pilot 2: “Must Have” functionaliteiten van menu XL-Enz	40 %	12
Pilot 3: Overige functionaliteiten van menu XL-Enz	20 %	8
Totaal:	**100 %**	**32**

Op basis van bovenstaande schatting is onderstaand overzicht gemaakt van de fase pilotontwikkeling.



Per pilot wordt een gedetailleerde versie van de planning gemaakt.

5.4 Pilotacceptatieplan

Aan het eind van elke iteratie wordt de pilot beoordeeld en getest op de wijze die is beschreven in paragraaf 7.2 “Systeemtest” uit het plan van aanpak.
Het beoordelen of de pilot voldoet aan de beschreven normen wordt uitgevoerd door de begeleider van dit project, dhr. T. Dekker, en eventueel door de opdrachtgever, dhr. G.A. Lenselink.
Wanneer niet aan de normen wordt voldaan, kan worden besloten om de fase pilotontwikkeling voor de betreffende pilot opnieuw te doorlopen, net zolang tot aan de normen wordt voldaan.

5.5 Globaal invoeringsplan

Nadat alle pilots voldoen aan de gestelde eisen en normen, kan worden overgegaan tot invoering van de pilots.
Pilot 1: Dataobject Crystal Reports kan worden ingevoerd door het huidige dataobject te vervangen door het .NET dataobject en door de client geschikt te maken om met een .NET omgeving te kunnen werken.
Het menu van XL-Enz kan worden ingevoerd indien het menu zoveel functionaliteiten bevat dat het zinvol is om het menu in te voeren. Deze beslissing wordt genomen door de begeleider en/of de opdrachtgever.
Invoering van het menu bestaat uit het vervangen van het huidige menu met het nieuwe menu. Daarnaast dient de client geschikt te worden gemaakt om met een .NET omgeving te kunnen werken.
Na invoering vindt een acceptatietest plaats, waarbij de pilots aan de hand van de gestelde eisen en normen getest worden op hun werking. Deze test wordt uitgevoerd door de begeleider en/of de opdrachtgever.

Conclusie

In dit document is een akkoord bereikt over de te ontwikkelen functionaliteiten van het menu van XL-Enz en van het dataobject voor Crystal Reports. Vervolgens is in de vorm van een pilotplan een plan opgesteld om deze functionaliteiten te ontwerpen en te implementeren.

Na goedkeuring van dit document door de begeleider kan het ontwikkeltraject worden vervolgd. De volgende fase is de pilotontwikkeling, waarin de hier gedefinieerde pilots worden ontwikkeld en geïmplementeerd.

Bijlage A: Plan van aanpak definitiestudie

Inleiding
Bij aanvang van dit project is een plan van aanpak opgesteld voor het gehele project. Deze bijlage bevat het plan van aanpak voor de fase definitiestudie, waarin het eerder gemaakte plan van aanpak verder wordt gedetailleerd en waar nodig bijgesteld.
Als eerste worden de resultaten beschreven die de fase definitiestudie oplevert. Vervolgens wordt een planning gemaakt voor deze fase, waarbij de planning uit het eerder gemaakte plan van aanpak als uitgangspunt fungeert. Als laatste wordt een ontwikkelscenario opgesteld. Dit scenario bestaat uit een beschrijving van hoe het traject van ontwerpen, implementeren en invoeren van het systeem verloopt.
Voor informatie over kwaliteitsborging en de projectorganisatie wordt verwezen naar het plan van aanpak voor het gehele project.

Producten en activiteiten definitiestudie
Hieronder volgt een overzicht van de activiteiten die worden uitgevoerd tijdens de fase definitiestudie. Per activiteit worden deelactiviteiten beschreven en er wordt aangegeven wat het resultaat is van deze activiteit.

- Definiëren ontwikkelscenario
  - Beschrijven globale pilotstrategie;
  - Beschrijven globale teststrategie;
  - Resultaat: uitbreiding op het plan van aanpak voor de definitiestudie.
- Definiëren eisen aan menu (data- en gebruikersinterface) van XL-Enz en dataobject voor Crystal Reports
  - Definiëren functionele eisen;
  - Definiëren niet-functionele eisen;
  - Prioriteren eisen;
  - Resultaat: geprioriteerde lijst van vastgestelde eisen.
- Bepalen systeemconcept
  - Specificeren functionele eisen;
  - Specificeren niet-functionele eisen;
  - Bepalen systeemconcept m.b.v. UML technieken;
  - Resultaat: beschrijving van het systeemconcept.
- Beschouwen technische structuur
  - Beschouwen technische architectuur (bepalen of systeemconcept te realiseren is in huidige architectuur);
  - Definiëren technische veranderingen;
  - Resultaat: beschrijving van technische structuur en eventuele benodigde aanpassingen hierop.
- Opstellen pilotplan
  - Definiëren pilotstructuur;
  - Structureren pilotontwikkeling;
  - Prioriteren pilots;
  - Opstellen pilotacceptatieplan;
  - Opstellen globaal invoeringsplan;
  - Opstellen globaal pilotontwikkelplan;
  - Resultaat: pilotplan.

De volgende activiteiten die IAD voorschrijft worden tijdens dit project niet uitgevoerd:

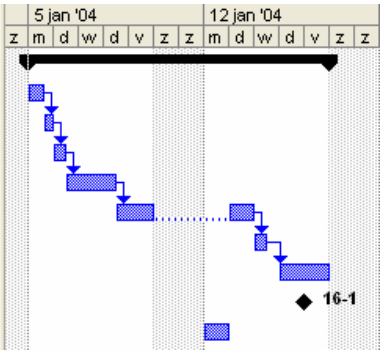
- Voorbereiden pilotplan-workshop
  Reden: Het project is te klein om een uitgebreide workshop te houden. Daarnaast zijn er te weinig betrokkenen bij het project om een workshop zinvol te laten zijn. De systeemeisen en het systeemconcept kunnen in overleg met de begeleider van het project worden vastgesteld.
- Evalueren pilot
  Reden: er zijn geen eerdere pilots om te evalueren. Gekozen is om de definitiestudie eenmalig uit te voeren, waardoor deze activiteit overbodig wordt.
- Beschouwen organisatorische inrichting
  Reden: De invoering van de pilots heeft geen impact op de organisatie, waardoor deze activiteit overbodig wordt.

Het resultaat van de fase definitiestudie is een rapport definitiestudie. Dit rapport bevat de resultaten van de bovengenoemde activiteiten.

Faseplanning

In deze paragraaf wordt een gedetailleerde planning van de fase definitiestudie opgesteld. Hierbij wordt de planning die is opgesteld in het plan van aanpak voor het gehele project als uitgangspunt gebruikt en waar nodig bijgesteld.

	Taaknaam	Duur	Begindatum	Einddatum
1	**Uitvoeren definitiestudie**	**80 uur**	**ma 5-1-04**	**vr 16-1-04**
2	Orientatie IAD / Definitiestudie	6 uur	ma 5-1-04	ma 5-1-04
3	Opstellen plan van aanpak definitiestudie	2 uur	ma 5-1-04	ma 5-1-04
4	Definiëren ontwikkelscenario	4 uur	di 6-1-04	di 6-1-04
5	Definiëren systeemeisen	16 uur	di 6-1-04	do 8-1-04
6	Bepalen systeemconcept	20 uur	do 8-1-04	di 13-1-04
7	Beschouwen technische structuur	4 uur	wo 14-1-04	wo 14-1-04
8	Opstellen pilotplan	16 uur	do 15-1-04	vr 16-1-04
9	Opleveren definitiestudie	1 uur	vr 16-1-04	vr 16-1-04
10	**Eindverslag schrijven**	**8 uur**	**ma 12-1-04**	**ma 12-1-04**



Ontwikkelscenario
In het plan van aanpak voor het gehele project is reeds de iteratiestrategie “incrementeel ontwikkelen” gekozen. Hierbij wordt de fase definitiestudie eenmalig doorlopen. De pilotontwikkeling wordt in iteraties uitgevoerd. De invoering van de pilots vindt in één keer plaats.
Op basis van de huidige inzichten kan gezegd worden dat het systeem waarschijnlijk wordt opgedeeld in de volgende pilots:

- Pilot 1: “Ontwikkeling en implementatie van het .NET dataobject voor Crystal Reports”;
- Pilot 2: “Ontwikkeling en implementatie van het menu van XL-Enz”.

De globale pilotstrategie ziet er hierdoor als volgt uit:

- Doorlopen fase definitiestudie;
- (Meerdere malen) doorlopen fase pilotontwikkeling voor pilot 1;
- (Meerdere malen) doorlopen fase pilotontwikkeling voor pilot 2;
- Invoeren pilots.

Na implementatie van elk pilotdeel wordt deze getest zoals is beschreven in het hoofdstuk “Testaanpak en Kwaliteitscontrole” van het plan van aan aanpak voor het gehele project.
Nadat de pilots zijn ingevoerd, wordt een acceptatietest uitgevoerd door de begeleider en/of opdrachtgever.

Bijlage 4

Pilotontwikkelrapport
Pilot 1: Dataobject Crystal Reports

Voorwoord

Voor u ligt het pilotontwikkelrapport dat ter ondersteuning dient bij de implementatie van de eerste pilot uit het project .NET interfacing. Tijdens deze pilot wordt het .NET dataobject ontwikkeld dat als datasource zal dienen voor Crystal Reports.

In het IAD systeemontwikkelingtraject volgt de fase pilotontwikkeling na de fase definitiestudie. Tijdens de pilotontwikkeling worden de pilots die tijdens de definitiestudie zijn gedefinieerd, gedetailleerd ontworpen en gebouwd.

Dit document is tot stand gekomen in overleg met dhr. T. Dekker, programmeur bij Reflecta Automation.

Reeuwijk-Dorp, 6 februari 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

Samenvatting

Doel rapport
Het doel van dit rapport is om het eerder ontworpen systeemconcept voor het .NET dataobject zodanig verder te ontwerpen, dat het mogelijk is om op basis van dit ontwerp het dataobject te implementeren. Daarnaast worden in dit document testgevallen beschreven, die als doel hebben om de geïmplementeerde delen te verifiëren en te valideren. Op basis van de resultaten van deze tests dient een beslissing te worden genomen over verdere ontwikkeling van de pilot.

Resultaten eerste iteratie pilotontwikkeling
In dit document is de pilot opgedeeld in bouweenheden. Bouweenheden zijn componenten van de pilot die door hun aard en reikwijdte achtereenvolgens of parallel kunnen worden ontwikkeld.
De volgende bouweenheden zijn gedefinieerd en vervolgens sequentieel geïmplementeerd:

1. Opzetten ontwikkelomgeving;
2. Bouwen functie voor het opnemen van de rapportdata in een Progress ProDataSet-object;
3. Converteren ProDataSet-object naar ADO.NET dataobject;
4. Invoegen dataobject in Crystal Reports.

Er is een invoeringsprocedure opgesteld voor deze pilot. Deze procedure dient te worden gevolgd wanneer de pilot wordt ingevoerd in de organisatie. Globaal bestaat deze procedure uit de volgende stappen:

1. Controleren aanwezigheid benodigde hard- en software;
2. Vervangen huidige functie voor genereren Progress dataobject dat rapportdata bevat met nieuwe functie;
3. Vervangen huidige functie voor genereren ADO XML bestand met nieuwe functie;
4. Wijzigen configuratie van datasources in Crystal Reports.

Na implementatie van elke bouweenheid, is deze getest op een correcte werking, functionaliteit en performance. De belangrijkste conclusies uit deze tests zijn:

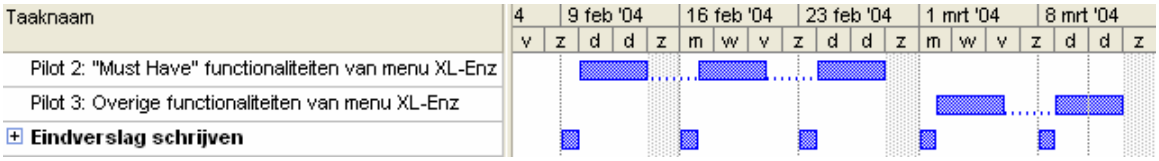
- De conversie van enkele datatypen verloopt niet correct. In Crystal Reports worden de datatypen “decimal”, “float” en “date” niet weergegeven zoals deze zijn gedefinieerd in het ADO.NET XML bestand;
- Bij een ADO.NET XML bestand dat meerdere tabellen bevat, voldoet de performance van het opbouwen van een rapport in Crystal Reports niet aan de eis dat deze gelijk of beter moet zijn dan de performance van het opbouwen van een rapport op basis van het huidige ADO XML dataobject;
- Het opbouwen van een rapport op basis van een ADO.NET XML bestand dat één tabel bevat, verloopt zeer snel: tijdens tests waren dergelijke rapporten binnen één seconde opgebouwd.

De slechte performance bij het opbouwen van een rapport dat is opgebouwd uit meerdere tabellen is momenteel de bottleneck voor een succesvolle implementatie van deze pilot. In een volgende iteratie van de fase pilotontwikkeling dient (onder meer) aandacht te worden besteed aan verbetering van deze performance.

In overleg met de begeleider en/of opdrachtgever van dit project dient te worden besloten of er een tweede iteratieslag voor deze pilot wordt uitgevoerd.

Planning komende fase

Volgens de planning voor de fase pilotontwikkeling die tijdens de definitiestudie is opgesteld, is de volgende stap het ontwikkelen van de tweede pilot uit dit project, de pilot waarin de “Must Have” functionaliteiten worden ontwikkeld van het menu voor XL-Enz.
De planning voor het vervolg van het project is hieronder opgenomen.



Deze planning wijzigt echter wanneer wordt besloten om voor de eerste pilot een tweede iteratieslag uit te voeren. Geschat wordt dat een tweede iteratie van de eerste pilot ongeveer twee weken zal kosten. Dit heeft tot gevolg dat het ontwikkelen van de tweede pilot ongeveer twee weken wordt uitgesteld.

Voortgang project

De eerste iteratie van de eerste pilot is verlopen volgens planning. De voortgang van het project voldoet hiermee aan de planning voor de fase pilotontwikkeling die is opgesteld tijdens de definitiestudie.

1. Inleiding

Tijdens de fase definitiestudie is het project .NET interfacing verdeeld in drie pilots. Deze pilots worden tijdens de fase pilotontwikkeling sequentieel ontwikkeld. De eerste pilot bestaat uit het ontwikkelen van het .NET dataobject dat als datasource zal dienen voor Crystal Reports. Dit rapport dient ter ondersteuning bij de ontwikkeling van deze pilot.

Het doel van dit rapport is om het eerder ontworpen systeemconcept voor het .NET dataobject zodanig verder te ontwerpen, dat het mogelijk is om op basis van dit ontwerp het dataobject te implementeren. Daarnaast worden in dit document testgevallen beschreven, die als doel hebben om de geïmplementeerde delen te verifiëren en te valideren. Op basis van de resultaten van deze tests dient een beslissing te worden genomen over verdere ontwikkeling van de pilot.

De opbouw van dit rapport is als volgt. Als eerste wordt in hoofdstuk 2 de functionele structuur van de pilot ontworpen. Vervolgens wordt de technische structuur in hoofdstuk 3 beschreven. In hoofdstuk 4 wordt een pilotontwikkelplan opgesteld, waarin diverse bouweenheden worden gedefinieerd. Deze bouweenheden worden in hoofdstuk 5 zodanig ontworpen, dat het mogelijk dient te zijn om op basis van dit ontwerp deze bouweenheden te implementeren. Hoofdstuk 6 beschrijft de procedure die dient te worden gevolgd wanneer de pilot wordt ingevoerd in de organisatie. Als laatste wordt in hoofdstuk 7 de geïmplementeerde pilot getest en beoordeeld. Hier worden de testgevallen uitgevoerd die in hoofdstuk 5 zijn opgesteld. Op basis van de resultaten van deze tests wordt de pilot beoordeeld.

2. Functionele structuur

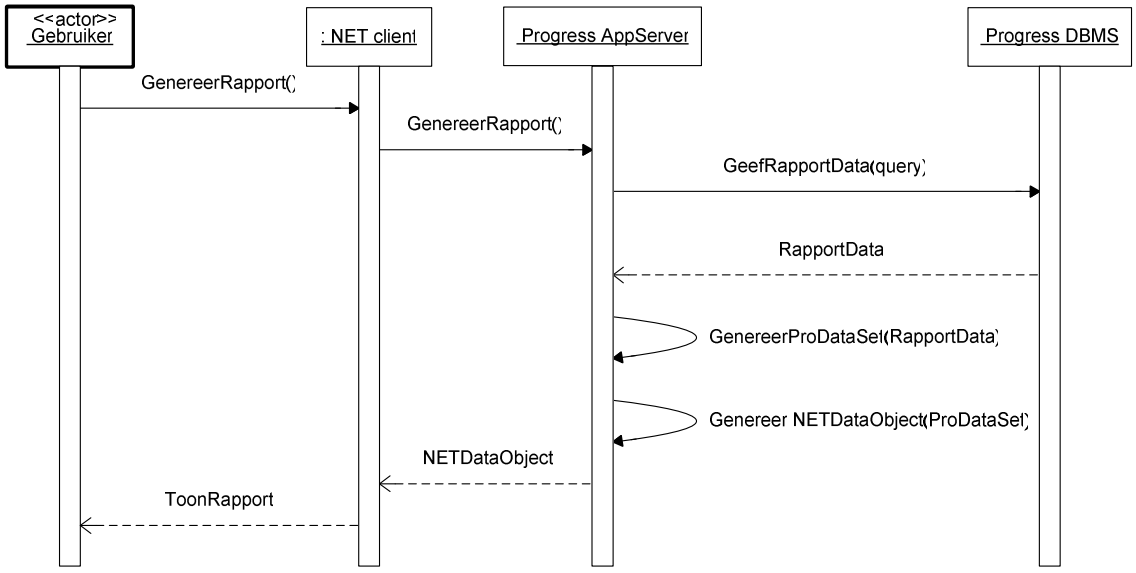
2.1 Inleiding

Het doel van dit hoofdstuk is om de functionele inhoud van de pilot zodanig te ontwerpen, dat deze als basis kan dienen voor de bouw van de pilot. Het systeemconcept uit de definitiestudie wordt in dit hoofdstuk verder verfijnd.
In paragraaf 2.2 wordt de interactie gemodelleerd tussen de objecten die benodigd is om het dataobject te genereren. Dit wordt gedaan met behulp van het UML sequencediagram.

2.2 Sequencediagram

Het UML sequencediagram geeft inzicht in de interactie tussen de objecten die nodig is om het gewenste systeemgedrag te verwezenlijken. Het diagram is direct gebaseerd op de use-case die gemaakt is tijdens de fase definitiestudie.
Aangezien het systeemconcept voor het genereren van een .NET dataobject voor Crystal Reports bestaat uit slechts één use-case die geen uitzonderingen bevat, kan hier worden volstaan met één sequencediagram.

Het onderstaande sequencediagram toont de interactie tussen de verschillende objecten die benodigd is om een .NET dataobject te genereren dat als datasource kan worden gebruikt in Crystal Reports.



Figuur 1: *Sequencediagram "Genereren rapport"*

3. Technische structuur

3.1 Inleiding

Dit hoofdstuk beschrijft de technische aspecten met betrekking tot de ontwikkeling van het .NET dataobject voor Crystal Reports. Deze aspecten zijn reeds globaal gedefinieerd tijdens de fase definitiestudie en zullen hier verder worden gedetailleerd.
Als eerste wordt in paragraaf 3.2 de fysieke allocatie gedetailleerd door de te ontwikkelen functionaliteiten toe te wijzen aan fysieke onderdelen. Vervolgens wordt in paragraaf 3.3 ingegaan op de technische aanpassingen die noodzakelijk zijn om Crystal Reports te laten werken op basis van een .NET dataobject als datasource.

3.2 Fysieke allocatie

In het voorgaande hoofdstuk is reeds te zien hoe de totstandkoming van het .NET dataobject verloopt. Verschillende componenten leveren elk een functionaliteit om tot een .NET dataobject te komen. Hieronder wordt deze technische architectuur verder gedetailleerd.

- Progress AppServer
  De AppServer bevat alle business logica van de applicatie XL-Enz. Op dit moment bestaat er reeds een functie die de aanvraag voor het samenstellen van een rapport afhandelt. Deze functie stelt op basis van de aanvraag een query samen, die wordt uitgevoerd op de Progress DBMS.
  - Er dient een functie te worden geschreven die het resultaat van de uitgevoerde query in een Progress ProDataSet-object opneemt;
  - Er dient een functie te worden geschreven die het ProDataSet-object converteert naar een ADO.NET dataobject.

- Progress DBMS
  De query die is samengesteld door de AppServer wordt door de Progress DBMS uitgevoerd. Het resultaat wordt ontvangen door de AppServer. Er zijn geen aanpassingen of uitbreidingen voor de Progress DBMS noodzakelijk.

- .NET Client
  De .NET Client ontvangt het ADO.NET dataobject van de AppServer. Dit object wordt als datasource ingelezen in Crystal Reports, waarna het rapport wordt gegenereerd en getoond.

3.3 Technische aanpassingen Crystal Reports

Crystal Reports gebruikt momenteel een XML bestand als datasource voor het creëren van rapporten. Dit XML bestand wordt vervangen door het ADO.NET DataSet-object. Hiervoor dient de configuratie van het type datasource in Crystal Reports te worden aangepast. Crystal Reports ondersteunt de mogelijkheid om een .NET dataobject te gebruiken als datasource (in de vorm van een ADO.NET XML bestand).

4. Pilotontwikkelplan

4.1 Inleiding

Het pilotontwikkelplan beschrijft hoe de pilot gaat worden gebouwd. Het pilotplan dat tijdens de definitiestudie is gemaakt wordt hier uitgebreid en eventueel aangepast. Per pilotdeel worden één of meer bouweenheden gedefinieerd. Bouweenheden zijn componenten van de pilot die door hun aard en reikwijdte achtereenvolgens of parallel kunnen worden ontwikkeld.
In paragraaf 4.2 worden de eerder gedefinieerde pilotdelen onderverdeeld in bouweenheden en wordt aangegeven in welke volgorde de bouweenheden worden ontwikkeld. Paragraaf 4.3 beschrijft de iteratiestrategie die wordt gevolgd om deze pilot iteratief te ontwikkelen.

4.2 Pilotdelen en bouweenheden

Tijdens de fase definitiestudie is de pilot reeds opgedeeld in de onderstaande pilotdelen.

1. Ontwikkelen dataobject in Progress OpenEdge 10;
2. Converteren Progress dataobject naar .NET dataobject;
3. Invoegen dataobject in Crystal Reports.

Hieronder worden per pilotdeel bouweenheden gedefinieerd.

4.2.1 Ontwikkelen dataobject in Progress OpenEdge 10

Als eerste wordt het Progress dataobject in de vorm van een ProDataSet-object gebouwd. Dit object bevat de data die de Progress DBMS levert op basis van de uitgevoerde query.
De volgende bouweenheden worden voor dit pilotdeel achtereenvolgens ontwikkeld:

1. Opzetten ontwikkelomgeving;
2. Bouwen functie voor het opnemen van de rapportdata in een Progress ProDataSet-object.

4.2.2 Converteren Progress dataobject naar .NET dataobject

Dit pilotdeel bestaat uit één bouweenheid:

3. Converteren ProDataSet-object naar ADO.NET dataobject.

Deze bouweenheid kan worden ontwikkeld wanneer de voorgaande bouweenheden zijn ontwikkeld en getest.

4.2.3 Invoegen dataobject in Crystal Reports

Dit pilotdeel bestaat uit één bouweenheid:

4. Invoegen dataobject in Crystal Reports.

Deze bouweenheid kan worden ontwikkeld wanneer de voorgaande bouweenheden zijn ontwikkeld en getest.

4.3 Iteratiestrategie

De pilot wordt op een iteratieve wijze ontwikkeld. Dit houdt in dat de fase pilotontwikkeling meerdere malen kan worden doorlopen, waardoor de pilot op een evoluerende wijze wordt geïmplementeerd.
Tijdens de eerste iteratie wordt deze pilot op de volgende wijze geïmplementeerd:

- De implementatie vindt plaats in een lokale omgeving.
  Deze keuze komt voort uit het feit dat het gebruik van nieuwe technieken (.NET, ProDataSet-object) onverwachte nadelige gevolgen kan hebben op de bestaande hard- en software. Door eerst de pilot in een lokale omgeving te implementeren, kunnen deze gevolgen worden onderzocht, zonder dat deze verdere gevolgen hebben voor de hard- en software die wordt gebruikt voor de ontwikkeling van de applicatie XL-Enz;
- Het Progress dataobject (ProDataSet-object) wordt als een statisch opgebouwd object geïmplementeerd.
  Hierdoor wordt de doorlooptijd van de eerste iteratie verkort, waardoor het mogelijk wordt eerder de pilot te testen (implementatie van een dynamisch opgebouwd dataobject zou namelijk meer tijd kosten dan implementatie van een statisch opgebouwd dataobject).

Na de implementatie van elke bouweenheid wordt deze getest op een correcte werking, functionaliteit en performance. Wanneer hieruit blijkt dat de pilot verder kan worden ontwikkeld, kan in een volgende iteratie de implementatie plaatsvinden in een gedistribueerde omgeving en kan het statisch opgebouwde dataobject worden vervangen door een dynamisch opgebouwd dataobject.

5. Ontwerp software-bouweenheden

5.1 Inleiding

Het doel van dit hoofdstuk is om van de hiervoor gedefinieerde bouweenheden de technische structuur te specificeren. Daarnaast worden in dit hoofdstuk testspecificaties opgesteld voor de bouweenheden. Op basis van het ontwerp en de testspecificaties uit dit hoofdstuk dient het mogelijk te zijn om de bouweenheden te implementeren en vervolgens te testen.
De opbouw van dit hoofdstuk is als volgt. In paragraaf 5.2 wordt voor elke bouweenheid een ontwerp gemaakt, op basis waarvan het mogelijk dient te zijn de bouweenheden te implementeren. Vervolgens worden in paragraaf 5.3 testspecificaties opgesteld, die dienen te worden uitgevoerd nadat een bouweenheid is geïmplementeerd.

5.2 Ontwerp bouweenheden

5.2.1 Bouweenheid 1: Opzetten ontwikkelomgeving

Als eerste dient de ontwikkelomgeving te worden opgezet. De benodigde onderdelen om de pilot te kunnen ontwikkelen, zijn:

- Microsoft Windows besturingssysteem (bij voorkeur Windows XP);
- Progress OpenEdge 10 ontwikkelomgeving;
- Progress AppServer;
- Progress DBMS (Sports2000 database (meegeleverd met OpenEdge 10) of database van XL-Enz);
- Crystal Reports (versie 9 of hoger);
- Microsoft .NET Framework (versie 1.1 of hoger);
- Visual Studio .NET 2003 ontwikkelomgeving.

5.2.2 Bouweenheid 2: Bouwen Progress ProDataSet

Wanneer de ontwikkelomgeving is opgezet en succesvol is getest, kan het Progress ProDataSet-object worden geïmplementeerd. Tijdens de eerste iteratieslag wordt een implementatie gemaakt op basis van een statisch opgebouwde query die willekeurige gegevens uit een Progress database opvraagt.
Wanneer hiermee een correct functionerende implementatie van een ProDataSet-object is verkregen, worden de volgende bouweenheden geïmplementeerd.
Wanneer alle bouweenheden zijn geïmplementeerd, worden de dynamisch opgebouwde query en de dynamisch opgebouwde ProDataSet geïmplementeerd. Deze implementatie is gebaseerd op de opbouw van het huidige dataobject en kan worden uitgevoerd als een iteratieslag van deze pilot.

Voor gedetailleerde informatie over de wijze van implementatie van een Progress ProDataSet-object wordt verwezen naar hoofdstuk 5 van het rapport “Interfacing tussen Progress en Microsoft .NET”. Daarnaast biedt de documentatie die Progress beschikbaar stelt over de implementatie van een ProDataSet-object voldoende informatie om deze bouweenheid te kunnen implementeren.

5.2.3 Bouweenheid 3: Converteren ProDataSet-object naar ADO.NET dataobject

Wanneer het (statisch opgebouwde) ProDataSet-object is geïmplementeerd en getest, kan deze worden geconverteerd naar een ADO.NET dataobject.
Dit dataobject bestaat uit een ADO.NET DataSet, die wordt opgeslagen in de vorm van een ADO.NET XML bestand.
Progress biedt functionaliteiten om een XML bestand te genereren. Tijdens de implementatie van deze bouweenheid wordt een functie geschreven die gebruik maakt van de genoemde functionaliteiten van Progress. In deze functie wordt de (meta)data uit het eerder gebouwde ProDataSet-object geconverteerd naar een XML bestand dat voldoet aan de eisen van een ADO.NET XML bestand. Deze functie dient een generiek karakter te hebben: elke ProDataSet dient te kunnen worden geconverteerd naar een ADO.NET XML bestand.

Voor gedetailleerde informatie over de eisen aan de opbouw van een ADO.NET XML bestand en over de wijze van conversie van een ProDataSet-object naar een ADO.NET XML bestand wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET".

5.2.4 Bouweenheid 4: Invoegen dataobject in Crystal Reports

Wanneer het ADO.NET XML bestand is aangemaakt, kan deze worden getransporteerd naar de client. Deze kan het bestand vervolgens als datasource gebruiken om rapporten te genereren in Crystal Reports.
Crystal Reports (versie 9 of hoger) biedt functionaliteiten om ADO.NET XML bestanden als datasource te kunnen gebruiken. Voor gedetailleerde informatie over het gebruik van ADO.NET XML bestanden als datasource wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET".

5.3 Testspecificaties

Na implementatie van een bouweenheid wordt deze getest aan de hand van testspecificaties. Zoals is vastgesteld in het document plan van aanpak van dit project, wordt de teststrategie "black box testing" gehanteerd. Hierbij wordt niet gekeken naar de inwendige werking van de bouweenheden, maar alleen naar het resultaat dat dient te worden geleverd op basis van een bepaalde invoer.
Hieronder zijn per bouweenheid testspecificaties opgesteld die worden uitgevoerd nadat een bouweenheid is geïmplementeerd.

5.3.1 Testspecificaties bouweenheid 1

In deze bouweenheid ("Opzetten ontwikkelomgeving") wordt geen software ontwikkeld die dient te worden getest. Om deze reden zijn voor bouweenheid 1 geen testspecificaties opgesteld.
Controleren of de benodigde onderdelen van de ontwikkelomgeving zijn geïnstalleerd, kan worden uitgevoerd door de lijst van geïnstalleerde software op te vragen en vervolgens na te gaan of het betreffende onderdeel zich op deze lijst bevindt. Deze lijst kan als volgt worden opgevraagd: Klik "Start" – "Control Panel" – "Add or remove programs".

Testen of de benodigde Progress database correct functioneert, kan worden gedaan door gebruik te maken van de functionaliteiten die de Progress omgeving hiervoor biedt.

5.3.2 Testspecificaties bouweenheid 2

Deze bouweenheid heeft als resultaat een functie die in een Progress omgeving een (statisch opgebouwd) ProDataSet-object oplevert, dat data bevat uit een Progress database.
Hieronder worden testspecificaties gegeven waarbij getest wordt of het ProDataSet-object de juiste data en relaties tussen de tabellen bevat.
Om de testspecificaties overzichtelijk te houden, is geen broncode in de testspecificaties opgenomen, maar alleen een tekstuele beschrijving.

De onderstaande tests dienen te worden uitgevoerd op een ProDataSet-object dat aan de volgende kenmerken voldoet:

- Het object bevat 2 TempTables: “ttCustomer” en “ttOrder”. Deze tabellen zijn gebaseerd op de tabellen “Customer” en “Order” uit de Sports2000 database;
- Er is een relatie gedefinieerd tussen deze tabellen, op basis van het veld “custnum”;
- De ProDataSet is gevuld op basis van bovenstaande relatie.

Testspecificaties bouweenheid 2: Bouwen Progress ProDataSet			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
2.1	ProDataSet is correct gevuld	Opvragen aantal records uit tabellen “ttCustomer” en “ttOrder”	# records “ttCustomer”: 1117 # records “ttOrder”: 3953
2.2	Relatie correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	“custnum, custnum”

5.3.3 Testspecificaties bouweenheid 3

Bouweenheid 3 “Converteren ProDataSet-object naar ADO.NET dataobject” levert een ADO.NET XML bestand op. De opbouw van dit bestand dient aan eisen te voldoen om correct te kunnen functioneren als datasource voor Crystal Reports.
Door middel van het uitvoeren van onderstaande testcases kan worden getest of het ADO.NET XML bestand aan deze eisen voldoet. Om een overzichtelijk beeld te verkrijgen van de opbouw van het ADO.NET XML bestand, kan deze worden ingelezen in Visual Studio .NET 2003.

Bij deze tests wordt uitgegaan van een ADO.NET XML bestand dat dezelfde data bevat als genoemd in voorgaande paragraaf.

Testspecificaties bouweenheid 3: Converteren ProDataSet-object naar ADO.NET dataobject			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
3.1	ADO.NET XML bestand bestaat	Controleren of ADO.NET XML bestand in gedefinieerde output directory staat	ADO.NET XML bestand bestaat
3.2	ADO.NET XML bestand bevat een syntactisch correcte ADO.NET DataSet (inclusief XSD schema)	ADO.NET XML bestand openen in Visual Studio .NET 2003	Overzicht van de tabellen en de data die het ADO.NET XML bestand bevat
3.3	Schema in ADO.NET XML bestand bevat correcte definitie van de juiste tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Schema in het ADO.NET XML bestand met gelijke definities van de tabellen “ttCustomer” en “ttOrder” als de definities van deze tabellen in Progress database
3.4	ADO.NET XML bestand bevat juiste data	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Data in het ADO.NET XML bestand met gelijke waarden als de waarden van de tabellen “ttCustomer” en “ttOrder” in Progress database
3.5	ADO.NET XML bestand bevat een correcte definitie van de juiste relaties tussen de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de relatie tussen de tabellen “ttCustomer” en “ttOrder”, op basis van het veld “custnum”
3.6	ADO.NET XML bestand bevat een correcte definitie van de juiste primaire sleutels van de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de primaire sleutels “Custnum” voor “ttCustomer” en “Ordernum” voor “ttOrder”

5.3.4 Testspecificaties bouweenheid 4

In bouweenheid 4 wordt het ADO.NET XML bestand ingelezen in Crystal Reports als datasource voor een rapport. De testspecificaties die voor deze bouweenheid zijn opgesteld, omvatten hoofdzakelijk tests voor het controleren op een juiste conversie van de rapportdata vanuit de Progress omgeving naar een ADO .NET XML bestand.

Bij deze tests wordt (tenzij anders aangegeven) uitgegaan van het ADO.NET XML bestand dat in de voorgaande bouweenheid gebruikt is als testbestand.

Testspecificaties bouweenheid 4: Invoegen dataobject in Crystal Reports			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
4.1	ADO.NET XML kan worden ingelezen in Crystal Reports	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	De tabellen “ttCustomer” en “ttOrder” worden weergegeven in het “Database expert”-window in Crystal Reports
4.2	De gedefinieerde relaties tussen de tabellen in het ADO.NET XML bestand worden correct herkend	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	De relatie tussen “ttCustomer” en “ttOrder” op basis van het veld “Custnum” wordt correct weergegeven in het “Database expert”-window in Crystal Reports
4.3	De tabelvelden die het datatype “string” hebben, worden correct weergegeven in het rapport (voldoen aan opgegeven stringlengte)	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat. Controle op veld “Country” uit tabel “ttCustomer” in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld “Country” heeft als datatype “string” en de waarden van het veld worden correct weergegeven in de Preview van het rapport. De waarde voor de maximale lengte van de string wordt correct weergegeven in de Field Explorer
4.4	De tabelvelden die het datatype “int” hebben, worden als geheel getal weer-gegeven in het rapport	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat. Controle op veld “Custnum” uit tabel “ttCustomer” in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld “Custnum” heeft als datatype “Number” en de waarden van het veld worden correct weergegeven in de Preview van het rapport
4.5	De tabelvelden die het datatype “decimal” of “float” hebben, worden als decimaal getal weergegeven in het rapport	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat. Controle op veld “CreditLimit” uit tabel “ttCustomer” in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld “CreditLimit” heeft als datatype “Number” en de waarden van het veld worden correct weergegeven in de Preview van het rapport
4.6	De tabelvelden die het datatype “date” hebben, worden als datums weergegeven in het rapport	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat. Controle op veld “OrderDate” uit tabel “ttOrder” in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld “OrderDate” heeft als datatype “Date” en de waarden van het veld worden correct weergegeven in de Preview van het rapport

Case #	Test	Invoer	Verwachte uitvoer
4.7	De tabelvelden die het datatype “boolean” hebben, worden als “true” of “false” weergegeven in het rapport	Een ADO.NET XML bestand dat de tabel “ttFamily” bevat (gebaseerd op de Sports2000 tabel “Family”). Controle op veld “CoveredOnBenefits” uit tabel “ttFamily” in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld “CoveredOnBenefits” heeft als datatype “Boolean” en de waarden van het veld worden correct weergegeven in de Preview van het rapport
4.8.1	De tijd die benodigd is om een rapport in te lezen is minstens gelijk aan de tijd die benodigd is om de reeds bestaande datasource in te lezen	Alle data uit een ADO.NET XML bestand dat bestaat uit alleen de tabel “ttCustomer” en alle data bevat uit de tabel “Customer”	De data is ingelezen in dezelfde of in minder tijd dan de tijd die benodigd is om dezelfde data in te lezen door gebruik te maken van het dataobject dat op dit moment als datasource wordt gebruikt
4.8.2		Alle data uit een ADO.NET XML bestand dat bestaat uit de tabellen “ttCustomer” en “ttOrder” en alle data bevat uit de tabellen “Customer” en “Order”	
4.8.3		Alle data uit een ADO.NET XML bestand dat bestaat uit de tabellen “ttCustomer”, “ttOrder”, “ttOrderLine”, “ttItem” en “ttBin” en alle data bevat uit de tabellen “Customer”, “Order”, “OrderLine”, “Item” en “Bin”	

6. Invoeringsprocedure pilot

6.1 Inleiding

Wanneer de pilot functioneert volgens de opgestelde eisen en de testgevallen succesvol heeft doorlopen, kan worden overgegaan tot het operationeel maken van de pilot. Deze beslissing wordt genomen door de opdrachtgever. Dit hoofdstuk beschrijft de procedure die dient te worden gevolgd wanneer de pilot wordt ingevoerd.
In paragraaf 6.2 wordt een plan opgesteld dat als uitgangspunt dient bij de invoering van de pilot. Vervolgens worden in paragraaf 6.3 testspecificaties opgesteld. De testgevallen kunnen worden uitgevoerd nadat de pilot is ingevoerd en dienen ter controle op een correcte invoering. Als laatste wordt in paragraaf 6.4 een noodprocedure opgesteld die kan worden uitgevoerd wanneer het invoeren mislukt.

6.2 Invoeringsplan

Hieronder is een plan voor invoering van de pilot gegeven in de vorm van een stapsgewijze beschrijving. Hierbij wordt ervan uitgegaan dat de pilot wordt ingevoerd als onderdeel van de applicatie XL-Enz.

1. Controleren aanwezigheid benodigde hard- en software:
   - Ontwikkelomgeving Progress (OpenEdge 10);
   - Toegang tot de broncode van de applicatie XL-Enz;
   - Toegang tot de database waar XL-Enz gebruik van maakt;
   - Crystal Reports (versie 9 of hoger);
   - Microsoft .NET Framework (versie 1.1 of hoger) is geïnstalleerd op de client.
2. Vervangen huidige functie voor genereren Progress dataobject dat rapportdata bevat met nieuwe functie.
   De functie waarin een TempTable wordt aangemaakt die alle rapportdata bevat, dient te worden vervangen met de nieuwe functie waarin een ProDataSet-object wordt aangemaakt. Er dient een backup te worden gemaakt van de huidige functie.
3. Vervangen huidige functie voor genereren ADO XML bestand met nieuwe functie.
   De functie die op dit moment wordt gebruikt om een ADO XML bestand te genereren, dient te worden vervangen met de nieuwe functie waarmee een ProDataSet-object wordt geconverteerd naar een ADO.NET dataobject dat wordt opgeslagen in een XML bestand. Er dient een backup te worden gemaakt van de huidige functie.
4. Wijzigen configuratie van datasources in Crystal Reports.
   Momenteel is Crystal Reports geconfigureerd op het gebruik van ADO XML datasources (d.m.v. ODBC). Deze configuratie dient te worden gewijzigd in het gebruik van een datasource op basis van ADO.NET XML.

De verantwoordelijkheden rond de invoering van de pilot worden opgenomen door het ontwikkelteam van Reflecta Automation.

6.3 Testspecificaties invoeringsprocedure

Hieronder is een lijst met testspecificaties opgenomen die dienen ter controle op een correcte werking van het betreffende deel uit het invoeringsplan.
Bij deze testspecificaties wordt de teststrategie “black box testing” gehanteerd.

Testspecificaties invoering pilot 1			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
1	Alle benodigde hard- en software is aanwezig	Opvragen lijst met geïnstalleerde hard- en software op betreffende systeem	Benodigde hard- en software is aanwezig op het juiste systeem
2	ProDataSet-object is aangemaakt en correct gevuld	Opdracht tot genereren rapport dat gegevens bevat uit de tabellen “zbm_Menu” en “zbm_MenuDescr”	ProDataSet-object is aangemaakt en bevat de correcte gegevens uit de tabellen “zbm_Menu” en “zbm_MenuDescr”
3	ADO.NET XML bestand is aangemaakt en bevat de correcte gegevens	Het ProDataSet-object uit testcase # 2	ADO.NET XML bestand dat correcte definitie en data uit tabellen “zbm_Menu” en “zbm_MenuDescr” bevat
4	Configuratie van datasources in Crystal Reports is correct gewijzigd	ADO.NET XML bestand	Data uit ADO.NET XML bestand kan worden gebruikt om een rapport te ontwerpen

6.4 Noodprocedure

Wanneer de invoeringsprocedure mislukt, dient het systeem in de oorspronkelijke staat te worden teruggebracht.
Dit kan worden gedaan door één of meer van de onderstaande stappen te volgen (al naar gelang het invoeringsplan is uitgevoerd):

1. Terugzetten van de oorspronkelijke functie waarmee het dataobject wordt gegenereerd dat de rapportdata bevat;
2. Terugzetten van de oorspronkelijke functie waarmee het ADO XML bestand wordt gegenereerd;
3. Herstellen van de configuratie voor het gebruik van ADO (ODBC) datasources in Crystal Reports.

7. Testen en beoordelen pilot

7.1 Inleiding

Na implementatie van elke bouweenheid is deze getest aan de hand van de testspecificaties uit hoofdstuk 5 van dit document. Dit hoofdstuk beschrijft de testresultaten na implementatie van de bouweenheden tijdens de eerste iteratieslag van de pilot. In paragraaf 7.2 wordt de testomgeving beschreven die is gebruikt om de pilot te testen. Vervolgens worden in de paragrafen 7.3 tot en met 7.6 de testresultaten beschreven. In paragraaf 7.7 wordt de pilot beoordeeld op basis van de testresultaten.

7.2 Testomgeving

Er is voor gekozen om de pilot in de eerste iteratie te implementeren in een lokale omgeving. Het systeem dat voor deze implementatie is gebruikt, had de onderstaande specificaties:

- HP Intel Pentium 4, 2 x 2.60 GHz;
- 248 MB RAM.

De onderstaande software was hierop geïnstalleerd:

- Microsoft Windows XP besturingssysteem;
- Progress OpenEdge 10A ontwikkelomgeving;
- Progress AppServer;
- Progress DBMS (Sports2000 database);
- Crystal Reports 9;
- Microsoft .NET Framework 1.1;
- Visual Studio .NET 2003 ontwikkelomgeving.

De pilot is geïmplementeerd op basis van een statisch opgebouwd Progress ProDataSet-object.

7.3 Testen bouweenheid 1

Deze bouweenheid (“Opzetten ontwikkelomgeving”) is getest door de lijst met geïnstalleerde software op te vragen. Volgens deze lijst was (onder meer) de volgende software geïnstalleerd:

- Progress OpenEdge 10A ontwikkelomgeving;
- Crystal Reports 9;
- Microsoft .NET Framework 1.1;
- Visual Studio .NET 2003 ontwikkelomgeving.

Het Windows XP besturingssysteem en de Progress AppServer functioneerde naar behoren.
De werking van de Progress Sports2000 database is getest door in de Progress omgeving een connectie te maken met deze database (d.m.v. de tool “Data Administration”). Deze tool gaf aan dat de verbinding met de database succesvol was. Deze bouweenheid heeft geen problemen opgeleverd.

7.4 Testen bouweenheid 2

In bouweenheid 2 “Bouwen Progress ProDataSet” is in een Progress omgeving een functie geïmplementeerd waarin een statisch ProDataSet-object wordt gecreëerd, die data bevat uit de Progress Sports2000 database.
Om deze bouweenheid te testen, is een ProDataSet-object gecreëerd dat aan de kenmerken voldeed die zijn opgesteld in de testspecificaties (hoofdstuk 5):

- Het object bevat 2 TempTables: “ttCustomer” en “ttOrder”. Deze tabellen zijn gebaseerd op de tabellen “Customer” en “Order” uit de Sports2000 database;
- Er is een relatie gedefinieerd tussen deze tabellen, op basis van het veld “custnum”;
- De ProDataSet is gevuld op basis van bovenstaande relatie.

Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 2: Bouwen Progress ProDataSet				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
2.1	ProDataSet is correct gevuld	Opvragen aantal records uit tabellen “ttCustomer” en “ttOrder”	# records “ttCustomer”: 1117 # records “ttOrder”: 3953	OK
2.2	Relatie correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	“custnum, custnum”	OK

7.5 Testen bouweenheid 3

In bouweenheid 3 is het Progress ProDataSet-object geconverteerd naar een ADO.NET dataobject. Dit dataobject is opgeslagen in een bestand in ADO.NET XML formaat.
Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid. Bij deze tests is uitgegaan van een ADO.NET XML bestand dat dezelfde data bevat als genoemd in voorgaande paragraaf.

Testspecificaties bouweenheid 3: Converteren ProDataSet-object naar ADO.NET dataobject				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
3.1	ADO.NET XML bestand bestaat	Controleren of ADO.NET XML bestand in gedefinieerde output directory staat	ADO.NET XML bestand bestaat	OK
3.2	ADO.NET XML bestand bevat een syntactisch correcte ADO.NET DataSet (inclusief XSD schema)	ADO.NET XML bestand openen in Visual Studio .NET 2003	Overzicht van de tabellen en de data die het ADO.NET XML bestand bevat	OK

Case #	Test	Invoer	Verwachte uitvoer	Uitvoer
3.3	Schema in ADO.NET XML bestand bevat correcte definitie van de juiste tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Schema in het ADO.NET XML bestand met gelijke definities van de tabellen “ttCustomer” en “ttOrder” als de definities van deze tabellen in Progress database	OK
3.4	ADO.NET XML bestand bevat juiste data	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Data in het ADO.NET XML bestand met gelijke waarden als de waarden van de tabellen “ttCustomer” en “ttOrder” in Progress database	OK
3.5	ADO.NET XML bestand bevat een correcte definitie van de juiste relaties tussen de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de relatie tussen de tabellen “ttCustomer” en “ttOrder”, op basis van het veld “custnum”	OK
3.6	ADO.NET XML bestand bevat een correcte definitie van de juiste primaire sleutels van de tabellen	Conversie van een ProDataSet-object naar ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	Correcte definitie van de primaire sleutels “Custnum” voor “ttCustomer” en “Ordernum” voor “ttOrder”	OK

7.6 Testen bouweenheid 4

In bouweenheid 4 is het ADO.NET XML bestand ingelezen in Crystal Reports als datasource voor een rapport. Bij deze tests is (tenzij anders aangegeven) uitgegaan van het ADO.NET XML bestand dat in de voorgaande bouweenheid gebruikt is als testbestand.
Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid. Meer informatie over mislukte testgevallen wordt gegeven onder het overzicht.

Testspecificaties bouweenheid 4: Invoegen dataobject in Crystal Reports				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
4.1	ADO.NET XML kan worden ingelezen in Crystal Reports	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	De tabellen “ttCustomer” en “ttOrder” worden weergegeven in het “Database expert”-window in Crystal Reports	OK
4.2	De gedefinieerde relaties tussen de tabellen in het ADO.NET XML bestand worden correct herkend	Het ADO.NET XML bestand dat de tabellen “ttCustomer” en “ttOrder” bevat	De relatie tussen “ttCustomer” en “ttOrder” op basis van het veld “Custnum” wordt correct weergegeven in het “Database expert”-window in Crystal Reports	OK

Case #	Test	Invoer	Verwachte uitvoer	Uitvoer
4.3	De tabelvelden die het datatype "string" hebben, worden correct weergegeven in het rapport (voldoen aan opgegeven stringlengte)	Het ADO.NET XML bestand dat de tabellen "ttCustomer" en "ttOrder" bevat. Controle op veld "Country" uit tabel "ttCustomer" in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld "Country" heeft als datatype "string" en de waarden van het veld worden correct weergegeven in de Preview van het rapport. De waarde voor de maximale lengte van de string wordt correct weergegeven in de Field Explorer	OK
4.4	De tabelvelden die het datatype "int" hebben, worden als geheel getal weer-gegeven in het rapport	Het ADO.NET XML bestand dat de tabellen "ttCustomer" en "ttOrder" bevat. Controle op veld "Custnum" uit tabel "ttCustomer" in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld "Custnum" heeft als datatype "Number" en de waarden van het veld worden correct weergegeven in de Preview van het rapport.	OK
4.5	De tabelvelden die het datatype "decimal" of "float" hebben, worden als decimaal getal weergegeven in het rapport	Het ADO.NET XML bestand dat de tabellen "ttCustomer" en "ttOrder" bevat. Controle op veld "CreditLimit" uit tabel "ttCustomer" in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld "CreditLimit" heeft als datatype "Number" en de waarden van het veld worden correct weergegeven in de Preview van het rapport.	Test mislukt: aantal decimalen achter de komma niet correct weergegeven
4.6	De tabelvelden die het datatype "date" hebben, worden als datums weergegeven in het rapport	Het ADO.NET XML bestand dat de tabellen "ttCustomer" en "ttOrder" bevat. Controle op veld "OrderDate" uit tabel "ttOrder" in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld "OrderDate" heeft als datatype "Date" en de waarden van het veld worden correct weergegeven in de Preview van het rapport.	Test mislukt: velden hebben datatype "DateTime"

Case #	Test	Invoer	Verwachte uitvoer	Uitvoer
4.7	De tabelvelden die het datatype "boolean" hebben, worden als "true" of "false" weergegeven in het rapport	Een ADO.NET XML bestand dat de tabel "ttFamily" bevat (gebaseerd op de Sports2000 tabel "Family"). Controle op veld "CoveredOnBenefits" uit tabel "ttFamily" in de Field Explorer van Crystal Reports. Dit veld wordt in het rapport opgenomen	Het veld "CoveredOnBenefits" heeft als datatype "Boolean" en de waarden van het veld worden correct weergegeven in de Preview van het rapport.	OK
4.8.1	De tijd die benodigd is om een rapport in te lezen is minstens gelijk aan de tijd die benodigd is om de reeds bestaande datasource in te lezen	Alle data uit een ADO.NET XML bestand dat bestaat uit alleen de tabel "ttCustomer" en alle data bevat uit de tabel "Customer"	De data is ingelezen in dezelfde of in minder tijd dan de tijd die benodigd is om dezelfde data in te lezen door gebruik te maken van het dataobject dat op dit moment als datasource dient	OK
4.8.2		Alle data uit een ADO.NET XML bestand dat bestaat uit de tabellen "ttCustomer" en "ttOrder" en alle data bevat uit de tabellen "Customer" en "Order"		Test mislukt
4.8.3		Alle data uit een ADO.NET XML bestand dat bestaat uit de tabellen "ttCustomer", "ttOrder", "ttOrderLine", "ttItem" en "ttBin" en alle data bevat uit de tabellen "Customer", "Order", "OrderLine", "Item" en "Bin"		Test mislukt

Mislukte tests:

- Testcase 4.5: Controle op datatype "decimal" of "float"
  Het aantal decimalen achter de komma wordt niet correct weergegeven. In het schema van de ADO.NET XML datasource is door middel van het attribuut "fractionDigits" het aantal decimalen achter de komma gedefinieerd. Wanneer de waarden echter in Crystal Reports worden ingelezen, wordt de waarde van dit attribuut genegeerd en wordt hiervoor de waarde gebruikt die is ingesteld in Crystal Reports (deze waarde is standaard "2").
  Op dit moment is hiervoor nog geen afdoende oplossing voor gevonden.

- Testcase 4.6: Controle op datatype “Date”
  Wanneer in de ADO.NET XML datasource het datatype van een veld “Date” is, wordt deze in Crystal Reports ingelezen als datatype “DateTime”. Dit datatype bestaat uit de elementen “datum” en “tijd”. Dit is niet gewenst, omdat de datums uit het XML bestand meestal geen tijdselement bevatten.
  Dit probleem kan worden opgelost door de weergave van het datatype DateTime in Crystal Reports zodanig aan te passen, dat alleen het datumelement wordt weergegeven. (Zie hiervoor de eigenschappen voor de weergave van het datatype DateTime in Crystal Reports: File – Options – Fields – Date and time.)

- Test 4.8.2: Performancetest inlezen data uit twee tabellen
  Wanneer data wordt ingelezen die bestaat uit een combinatie van gegevens uit twee tabellen, is de inleessnelheid lager dan wanneer dezelfde data wordt ingelezen door gebruik te maken van de huidige (ADO XML) datasource (Deze uitspraak is gebaseerd op een schatting).
  Hieronder is een overzicht opgenomen van de uitkomsten van de performancetests.

Case #	# velden ttCustomer	# velden ttOrder	Totaal # records	Totale inleestijd (sec)	# records per seconde
1	1	1	3953	~40	~98
2	3	3	3953	~51	~75
3	6	6	3953	~63	~62
4	6	1	3953	~60	~66
5	1	6	3953	~35	~113
6	10	10	3953	~71	~55
7	10	1	3953	~72	~54
8	1	10	3953	~43	~92

Hieruit kan het volgende worden geconcludeerd:

1. De performance daalt naarmate er meer velden uit beide tabellen in een rapport worden opgenomen;
2. Er is vrijwel geen verschil in performance tussen gevallen waarbij veel velden uit beide tabellen in het rapport zijn opgenomen en gevallen waarbij veel velden uit de tabel “ttCustomer” en weinig velden uit de tabel “ttOrder” zijn opgenomen;
3. De performance daalt sterk in gevallen waarbij weinig velden uit de tabel “ttCustomer” en veel velden uit de tabel “ttOrder” zijn opgenomen.

Verwacht wordt dat de slechte performance veroorzaakt wordt doordat de data in het ADO.NET XML bestand niet is geïndexeerd op het veld “CustNum” uit de tabel “ttOrder”. Als gevolg hiervan dient het volledige XML bestand te worden doorlopen om de juiste combinatie van records te vinden. Dit werkt sterk vertragend.
Op dit moment is nog geen afdoende oplossing voor dit probleem gevonden. Onderzoek heeft uitgewezen dat ADO.NET XML bestanden geen indexering ondersteunen.

- Test 4.8.3: Performancetest inlezen data uit vijf tabellen
  Evenals bij testcase 4.8.2 is de inleessnelheid van het inlezen van gegevens uit meer dan twee tabellen lager dan wanneer dezelfde data wordt ingelezen door gebruik te maken van de huidige (ADO XML) datasource (Deze uitspraak is gebaseerd op een schatting).
  Hieronder is een overzicht opgenomen van de uitkomsten van de performancetests.

Case #	# velden					Totaal # records	Totale inlees tijd (sec)	# records/ sec
	ttCus-tomer	ttOrder	ttOrder Line	ttItem	ttBin			
1	1	1	0	0	0	3953	~34	~116
2	1	1	1	0	0	4620	~203	~23
3	1	1	1	1	0	5022	~225	~24
4	1	1	1	1	1	14955	~672	~22
5	0	0	0	1	1	770	~0.3	~2600
6	0	0	1	1	1	195580	~85	~2300
7	0	1	1	1	1	4589	~140	~32
8	0	1	1	0	0	13970	~458	~30
9	0	5	5	0	0	1025	~54	~19
10	0	10	8	0	0	1782	~121	~15
11	10	10	8	10	5	2429	~237	~10

De resultaten bevestigen de conclusies die zijn getrokken uit de tests met alleen de tabellen “ttCustomer” en “ttOrder”.
Er is geen duidelijke verklaring voor de resultaten uit testcases 5 en 6. Een mogelijke oorzaak kan de definitie van de relaties tussen de tabellen zijn. Bij deze tests is gebruik gemaakt van definities van relaties vanuit de “child” tabel naar de “parent” tabel (bijvoorbeeld vanuit de tabel “ttBin” naar “ttItem”, op basis van het veld “ItemNum”). Tests met definities van relaties vanuit de “parent” tabel naar de “child” tabel hebben echter geen verbetering opgeleverd.

7.7 Beoordeling pilot

In deze paragraaf wordt beoordeeld of de pilot (na de eerste iteratie) voldoet aan de eisen die zijn opgesteld in het rapport definitiestudie.
In het onderstaande overzicht is te zien aan welke eisen is voldaan.

ID	Omschrijving	MoSCoW	Voldaan
DF1	Een gebruiker is in staat om Crystal Reports een rapport te laten genereren op basis van een .NET dataobject als datasource	M	Ja
DN1	Het dataobject kan gegevens uit een bestaande Progress database inlezen	M	Ja
DN2	Het dataobject is te converteren van een Progress object naar een ADO.NET object	M	Ja
DN3	Het dataobject wordt opgebouwd op de server als een Progress object, vervolgens geconverteerd naar een ADO.NET object en daarna verstuurd naar client	M	Nee: dataobject wordt lokaal opgebouwd
DN4	De tijd die benodigd is om een rapport op te bouwen op basis van een .NET dataobject als datasource, dient minstens gelijk te zijn aan de tijd die benodigd is om een rapport op te bouwen op basis van het huidige dataobject	M	Nee: zie resultaten tests 4.8.2 en 4.8.3 (paragraaf 7.6)

Aan de eis DN3 is niet voldaan, omdat ervoor gekozen is tijdens de eerste iteratie het dataobject lokaal op te bouwen. Een volgende iteratie kan tot doel hebben om (onder meer) aan deze eis te voldoen.
Bij een ADO.NET XML bestand dat meerdere tabellen bevat, voldoet de performance van het opbouwen van een rapport in Crystal Reports niet aan de eis dat deze gelijk of beter moet zijn dan de performance van het opbouwen van een rapport op basis van het huidige ADO XML dataobject.
Het opbouwen van een rapport op basis van een ADO.NET XML bestand dat één tabel bevat, verloopt zeer snel: tijdens tests waren dergelijke rapporten binnen één seconde opgebouwd.

De slechte performance bij het opbouwen van een rapport dat is opgebouwd uit meerdere tabellen is momenteel de bottleneck voor een succesvolle implementatie van deze pilot. In een eventuele volgende iteratie van de fase pilotontwikkeling dient aandacht te worden besteed aan verbetering van deze performance.

Conclusie

In dit document is een ontwerp gemaakt dat heeft gediend ter ondersteuning bij de implementatie van de eerste pilot uit het project .NET interfacing, waarin een .NET dataobject is ontwikkeld dat als datasource dient voor Crystal Reports.
Na implementatie van elke bouweenheid zijn de opgestelde testspecificaties uitgevoerd. De resultaten van deze tests hebben uitgewezen dat de pilot na de eerste iteratie nog niet voldoet aan alle eisen die zijn vastgesteld tijdens de fase definitiestudie.
Vooral de slechte performance bij het opbouwen van een rapport dat gegevens uit meerdere tabellen bevat, vormt momenteel de bottleneck voor een succesvolle implementatie van deze pilot.

In overleg met de begeleider en/of opdrachtgever van dit project dient te worden besloten of er een tweede iteratieslag voor deze pilot wordt uitgevoerd.

Bijlage A: Plan van aanpak ontwikkeling pilot 1

Inleiding
Deze bijlage bevat het plan van aanpak voor de ontwikkeling van de eerste pilot, waarin het .NET dataobject voor Crystal Reports wordt geïmplementeerd. Dit plan van aanpak is een detaillering van het eerder gemaakte plan van aanpak voor het gehele project. Als eerste worden de resultaten beschreven die deze pilot oplevert. Vervolgens wordt een planning gemaakt voor de uitvoering van de pilot, waarbij de planning uit het eerder gemaakte plan van aanpak als uitgangspunt fungeert.
Voor informatie over kwaliteitsborging en de projectorganisatie wordt verwezen naar het plan van aanpak voor het gehele project.

Producten en activiteiten pilotontwikkeling
Hieronder volgt een overzicht van de activiteiten die worden uitgevoerd tijdens de ontwikkeling van de eerste pilot. Per activiteit worden deelactiviteiten beschreven en er wordt aangegeven wat het resultaat is van deze activiteit.

- Specificeren globaal-functionele structuur pilot
  - Detailleren scenario's d.m.v. UML diagram(men);
  - Resultaat: ontwerp van functionele structuur.
- Specificeren globaal-technische structuur pilot
  - Bepalen fysieke allocatie;
  - Specificeren interfaces met externe componenten;
  - Resultaat: ontwerp van technische structuur.
- Opstellen pilotontwikkelplan
  - Definiëren pilotdelen;
  - Definieren bouweenheden;
  - Synchroniseren bouweenheden;
  - Voorbereiden beoordeling en testen pilotdelen;
  - Resultaat: ontwikkelplan voor pilot.
- Ontwerpen software-bouweenheden
  - Specificeren integratie en testen bouweenheden;
  - Opstellen detailspecificaties;
  - Opstellen testspecificaties;
  - Resultaat: gedetailleerd ontwerp pilotdelen.
- Bouwen software-bouweenheden
  - Coderen pilotdelen;
  - Preparen component-testgegevens;
  - Uitvoeren component-test ;
  - Corrigeren component;
  - Resultaat: implementatie van pilot.

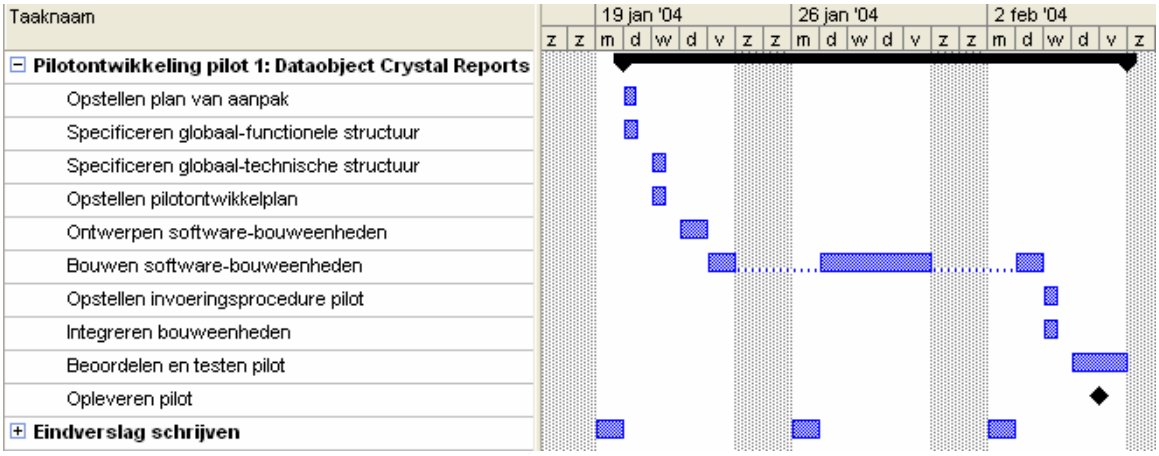
- Opstellen invoeringsprocedure pilot
  - Ontwerpen invoeringsprocedure;
  - Beschrijven verantwoordelijkheden rond invoering;
  - Specificeren tests invoeringsprocedures;
  - Testen invoeringsprocedures;
  - Corrigeren invoeringsprocedures;
  - Resultaat: beschrijving invoeringsprocedure.
- Integreren bouweenheden
  - Integreren bouweenheden;
  - Opzetten integratietestomgeving;
  - Testen integratie bouweenheden;
  - Corrigeren integratie bouweenheden;
  - Resultaat: bouweenheden zijn samengevoegd tot één geheel.
- Beoordelen en testen pilot
  - Opzetten testomgeving;
  - Beoordelen en testen pilot;
  - Analyseren uitkomsten;
  - Corrigeren fouten;
  - Rapporteren over beoordelen en testen.

De volgende activiteiten die IAD voorschrijft worden tijdens dit project niet uitgevoerd:

- Voorbereiden pilotontwerp-workshop
  Reden: De pilot is te klein om een workshop zinvol te laten zijn.
- Specificeren globaal-organisatorische inrichting
  Reden: De pilot heeft geen invloed op organisatorische aspecten.
- Aanpassen externe componenten
  Reden: Er is geen noodzaak tot aanpassing van externe componenten.
- Wijzigen andere informatiesystemen
  Reden: Er is geen noodzaak tot aanpassing van andere informatiesystemen.
- Bouwen conversie-tools
  Reden: Er zijn geen conversie-tools benodigd voor deze pilot.
- Ontwerpen handmatige procedures
  Reden: Er is geen sprake van handmatige procedures bij deze pilot.
- Samenstellen opleidingsmateriaal pilot
  Reden: Er is geen behoefte aan opleidingsmateriaal voor deze pilot.
- Samenstellen handleiding pilot
  Reden: Deze activiteit is gepland tijdens de fase invoering.

Iteratiestrategie
Na het uitvoeren van bovenstaande activiteiten zal de pilot worden getoetst of deze voldoet aan de gestelde eisen en normen die in de definitiestudie zijn vastgesteld. Mocht dit niet het geval zijn, dan kan de fase pilotontwikkeling nogmaals worden doorlopen. Deze beslissing wordt genomen in overleg met de begeleider en/of opdrachtgever. Tijdens een volgende iteratie wordt het ontwerp aangepast of verfijnd om deze aanpassingen vervolgens te implementeren en opnieuw te beoordelen.
Hoewel de fase definitiestudie eenmalig is uitgevoerd, zal het pilotplan eventueel worden bijgesteld aan nieuwe ontwikkelingen.

Faseplanning pilot
Hieronder wordt een gedetailleerde planning van de ontwikkeling van de eerste pilot opgesteld. Hierbij wordt de planning die is opgesteld in het plan van aanpak voor het gehele project als uitgangspunt gebruikt en waar nodig bijgesteld.
De doorlooptijd van deze pilot is gepland op drie weken, van 19 januari 2004 tot en met 6 februari 2004.



Bijlage 5

Pilotontwikkelrapport

Pilot 2: “Must Have” functionaliteiten menu XL-Enz

Voorwoord

Voor u ligt het pilotontwikkelrapport dat ter ondersteuning dient bij de implementatie van de tweede pilot uit het project .NET interfacing. Tijdens deze pilot worden de functionaliteiten van het menu van XL-Enz ontwikkeld, die in het rapport definitiestudie de prioriteit "Must Have" hebben gekregen.

In het IAD systeemontwikkelingtraject volgt de fase pilotontwikkeling na de fase definitiestudie. Tijdens de pilotontwikkeling worden de pilots die tijdens de definitiestudie zijn gedefinieerd, gedetailleerd ontworpen en gebouwd.

Dit document is tot stand gekomen in overleg met dhr. T. Dekker, programmeur bij Reflecta Automation.

Reeuwijk-Dorp, 11 maart 2004

Aris Schlingmann
Reflecta Automation B.V.

Inhoudsopgave

Samenvatting

Doel rapport
Het doel van dit rapport is om het eerder ontworpen systeemconcept voor het menu van XL-Enz zodanig verder te ontwerpen, dat het mogelijk is om op basis van dit ontwerp de "Must Have" functionaliteiten van het menu te implementeren. Daarnaast worden in dit document testgevallen beschreven, die als doel hebben om de geïmplementeerde delen te verifiëren en te valideren. Op basis van de resultaten van deze tests dient een beslissing te worden genomen over de verdere ontwikkeling van de pilot.

Resultaten eerste iteratie pilotontwikkeling
In dit document is de pilot opgedeeld in bouweenheden. Bouweenheden zijn componenten van de pilot die door hun aard en reikwijdte achtereenvolgens of parallel kunnen worden ontwikkeld.
De volgende bouweenheden zijn gedefinieerd en vervolgens sequentieel geïmplementeerd:

1. Opzetten ontwikkelomgeving;
2. Bouwen Progress menu-dataobject;
3. Converteren Progress dataobject naar .NET dataobject;
4. Menu initialiseren op basis van menu-dataobject;
5. Ontwikkelen navigatiemogelijkheid;
6. Ontwikkelen mogelijkheid tot afsluiten menu;
7. Menu compileren en registreren als ActiveX object;
8. Menu als ActiveX object invoegen in Progress;
9. Ontwikkelen mogelijkheid opstarten van programma's.

Er is een invoeringsprocedure opgesteld voor deze pilot. Deze procedure dient te worden gevolgd wanneer de pilot wordt ingevoerd in de organisatie. Globaal bestaat deze procedure uit de volgende stappen:

1. Controleren aanwezigheid benodigde hard- en software;
2. Installeren Progress functie waarmee menu-data wordt opgehaald uit de database;
3. Plaatsen proxy-object en benodigde assembly bestanden in de juiste directory;
4. Vervangen huidige menu van XL-Enz met nieuw menu.

Na implementatie van elke bouweenheid, is deze getest op een correcte werking, op functionaliteit en op performance. De belangrijkste conclusies uit deze tests zijn:

- Op dit moment is het mogelijk om in een .NET omgeving het menu te initialiseren en te navigeren;
- Wanneer het menu in een Progress omgeving wordt ingevoegd als een ActiveX object (zie bouweenheid 8), loopt de Progress ontwikkelomgeving vast of wordt deze afgesloten. Op dit moment is hiervoor nog geen reden en/of oplossing voor gevonden.
  Dit probleem heeft tot gevolg dat de functionaliteit voor het opstarten van programma's nog niet is ontwikkeld, omdat deze functionaliteit alleen vanuit een Progress omgeving kan worden geïmplementeerd.

Als gevolg van bovenstaand probleem kan de pilot niet als succesvol afgerond worden beschouwd.
Wanneer de reden bekend is voor het probleem en een oplossing beschikbaar is, kan de ontwikkeling van bouweenheid 8 worden voortgezet door de oplossing te implementeren. Wanneer de geboden oplossing dit vereist, kan de fase pilotontwikkeling in zijn geheel opnieuw worden doorlopen, waarbij de eisen en het ontwerp worden aangepast aan de nieuwe inzichten.
Wanneer er geen oplossing mogelijk blijkt te zijn voor het genoemde probleem, dient met de begeleider en/of opdrachtgever van dit project overleg te worden gepleegd over de verdere ontwikkeling van het menu.

Planning komende fase
De planning voor de komende fase is afhankelijk van de beslissing die door de begeleider en/of opdrachtgever wordt genomen over de verdere ontwikkeling van het menu.
In de komende fase kan één van de volgende activiteiten worden uitgevoerd:

- Nogmaals doorlopen fase pilotontwikkeling pilot 1: Dataobject Crystal Reports.
  In deze iteratie kan onderzoek worden verricht naar verbetering van de performance van het inlezen van een rapport op basis van een ADO.NET XML document;
- Nogmaals doorlopen fase pilotontwikkeling pilot 2: "Must Have" functionaliteiten menu XL-Enz.
  In deze iteratie kan onderzoek worden verricht naar een oplossing voor het probleem van invoegen van het menu in Progress.

Beide activiteiten zullen ongeveer drie weken in beslag nemen.
Op dit moment is het implementeren van pilot 3, waarin de overige functionaliteiten van het menu van XL-Enz worden ontwikkeld, niet zinvol. Dit vanwege het feit dat het menu in de huidige vorm nog niet bruikbaar is. Wanneer blijkt dat het technisch niet mogelijk is om het menu als ActiveX object in te voegen in Progress, kan de ontwikkeling van pilot 3 worden geannulleerd.

Voortgang project
Wegens technische problemen bij het invoegen van het menu in Progress, is het project twee weken uitgelopen op de planning zoals die is opgesteld in het plan van aanpak voor deze pilot. In overleg met de begeleider en/of opdrachtgever dient een beslissing te worden genomen over het verdere verloop van het project.

1. Inleiding

Tijdens de fase definitiestudie is het project .NET interfacing verdeeld in drie pilots. Deze pilots worden tijdens de fase pilotontwikkeling sequentieel ontwikkeld. De eerste pilot bestond uit het ontwikkelen van het .NET dataobject dat als datasource dient voor Crystal Reports. Na de eerste iteratieslag van deze pilot is in overleg besloten om het project te vervolgen met de ontwikkeling van de tweede pilot. Dit rapport dient ter ondersteuning bij de ontwikkeling van deze pilot, waarin de "Must Have" functionaliteiten van het menu van XL-Enz worden ontworpen en geïmplementeerd.

Het doel van dit rapport is om het eerder ontworpen systeemconcept voor het menu van XL-Enz zodanig verder te ontwerpen, dat het mogelijk is om op basis van dit ontwerp de betreffende functionaliteiten te implementeren. Daarnaast worden in dit document testgevallen beschreven, die als doel hebben om de geïmplementeerde functionaliteiten te verifiëren en te valideren. Op basis van de resultaten van deze tests dient een beslissing te worden genomen over verdere ontwikkeling van de pilot.

De opbouw van dit rapport is als volgt. Als eerste wordt in hoofdstuk 2 de functionele structuur van de pilot ontworpen. Vervolgens wordt de technische structuur in hoofdstuk 3 beschreven. In hoofdstuk 4 wordt een pilotontwikkelplan opgesteld, waarin diverse bouweenheden worden gedefinieerd. Deze bouweenheden worden in hoofdstuk 5 zodanig ontworpen, dat het mogelijk dient te zijn om op basis van dit ontwerp deze bouweenheden te implementeren. Hoofdstuk 6 beschrijft de procedure die dient te worden gevolgd wanneer de pilot wordt ingevoerd in de organisatie. Als laatste wordt in hoofdstuk 7 de geïmplementeerde pilot getest en beoordeeld. Hier worden de testgevallen uitgevoerd die in hoofdstuk 5 zijn opgesteld. Op basis van de resultaten van deze tests wordt de pilot beoordeeld.

2. Functionele structuur

2.1 Inleiding

Het doel van dit hoofdstuk is om de functionele inhoud van de pilot zodanig te ontwerpen, dat deze als basis kan dienen voor de bouw van de pilot. Het systeemconcept uit de definitiestudie wordt hierbij verder verfijnd.
De opbouw van dit hoofdstuk is als volgt. In paragraaf 2.2 is een overzicht opgenomen van de functionaliteiten die worden ontwikkeld tijdens deze pilot. Paragraaf 2.3 bevat een ontwerp van de interactie die benodigd is om de functionaliteiten uit te kunnen voeren. Als laatste is in paragraaf 3.4 een prototype van de gebruikersinterface van het menu opgenomen.

2.2 Functionele reikwijdte

In deze pilot worden de eisen geïmplementeerd die tijdens de fase definitiestudie de prioriteit "Must Have" hebben gekregen. Hieronder is een overzicht opgenomen van deze eisen.

2.2.1 Functionele eisen

ID	Omschrijving
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma's van XL-Enz kunnen worden opgestart
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer
MF3	Een gebruiker kan het menu afsluiten

2.2.2 Niet-functionele eisen

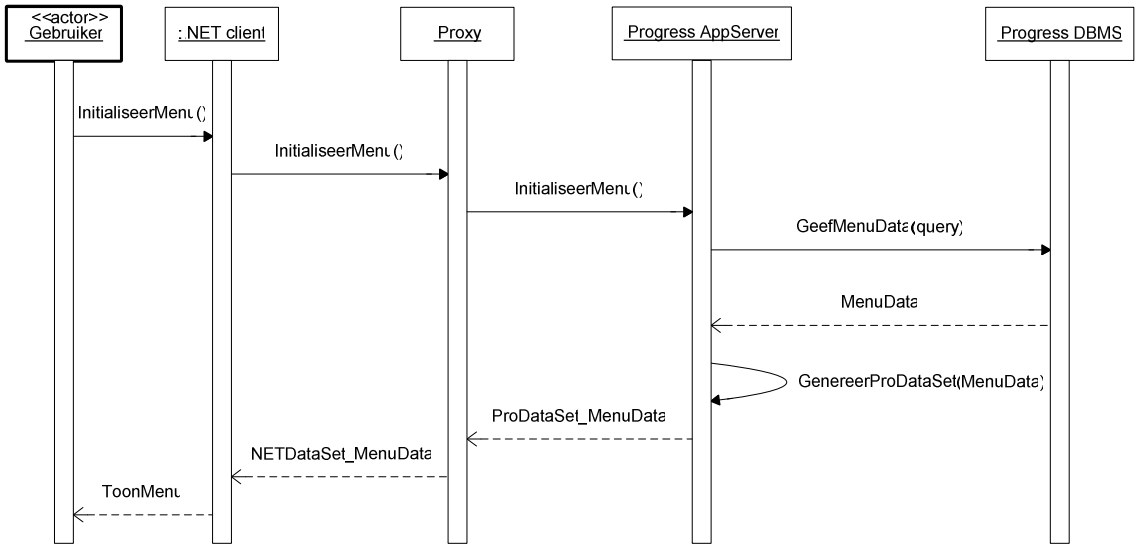
ID	Omschrijving
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database
MN2	Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven
MN4	Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress dataobject en vervolgens doorgegeven aan de .NET client, waar het object wordt geconverteerd naar een ADO.NET dataobject

2.3 Sequencediagrammen

In het rapport definitiestudie zijn UML use-cases ontworpen, waarin de interactie tussen de gebruiker en het systeem is gemodelleerd. Deze use-cases vormen het uitgangspunt voor het modelleren van UML sequencediagrammen. Het sequencediagram geeft inzicht in de interactie tussen de objecten die nodig is om het gewenste systeemgedrag te verwezenlijken.
Van elke use-case die de afhandeling van één van de "Must Have" functionaliteiten beschrijft, is hieronder een sequencediagram opgenomen.

2.3.1 Menu initialiseren

Het onderstaande sequencediagram geeft de interactie tussen de verschillende objecten weer, die benodigd is om een instantie van het menu te initialiseren.



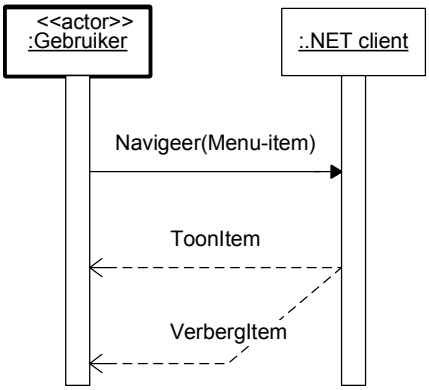
Figuur 1: *Sequencediagram menu initialiseren*

De volgende activiteiten worden uitgevoerd om het menu te initialiseren:

- De gebruiker geeft het systeem opdracht het menu te initialiseren;
- De .NET client ontvangt deze aanvraag en geeft deze door aan het proxy-object;
- Het proxy-object ontvangt de aanvraag en start een functie op de Progress AppServer die de benodigde menudata opvraagt;
- De Progress AppServer voert de betreffende functie uit en vraagt daarbij de menudata op uit de Progress DBMS d.m.v. een query;
- De Progress DBMS voert de query uit en geeft het resultaat ervan door aan de Progress AppServer;
- De functie die wordt uitgevoerd op de Progress AppServer maakt een ProDataSet-object aan dat de menudata bevat;
- Het ProDataSet-object wordt verstuurd naar het proxy-object;
- Het proxy-object vertaalt het ProDataSet-object naar een ADO.NET DataSet object en geeft deze door aan de .NET client;
- De .NET client bouwt het menu op en toont deze aan de gebruiker.

2.3.2 Menu navigeren

Het onderstaande sequencediagram geeft de interactie tussen de verschillende objecten weer, die benodigd is om het menu te kunnen navigeren.



Figuur 2: *Sequencediagram menu navigeren*

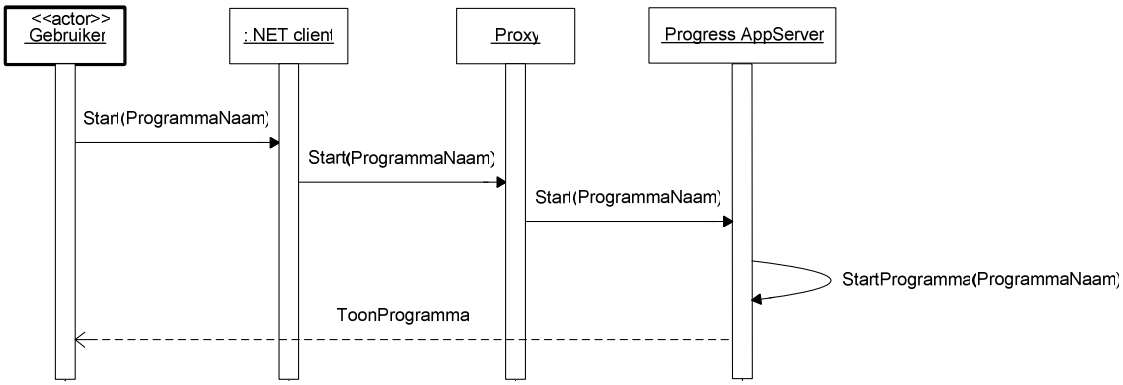
Het navigeren van het menu heeft geen gevolgen voor andere objecten. De volgende activiteiten worden uitgevoerd om het menu te navigeren:

- De gebruiker (dubbel)klikt op een item uit het menu;
- De .NET client voert een actie uit waardoor de inhoud van het menu-item wordt getoond als dit nog niet het geval was en wordt verborgen wanneer de inhoud reeds zichtbaar was.

2.3.3 Programma starten

Het sequencediagram op de volgende pagina geeft de interactie tussen de verschillende objecten weer, die benodigd is om een programma te starten. Bij de use-case die in het rapport definitiestudie is opgenomen voor deze interactie, is ook rekening gehouden met autorisatie-eisen die worden gesteld aan het starten van programma's. In deze pilot wordt autorisatie echter nog niet geïmplementeerd. Om deze reden is dit aspect ook niet meegenomen in het sequencediagram.

.



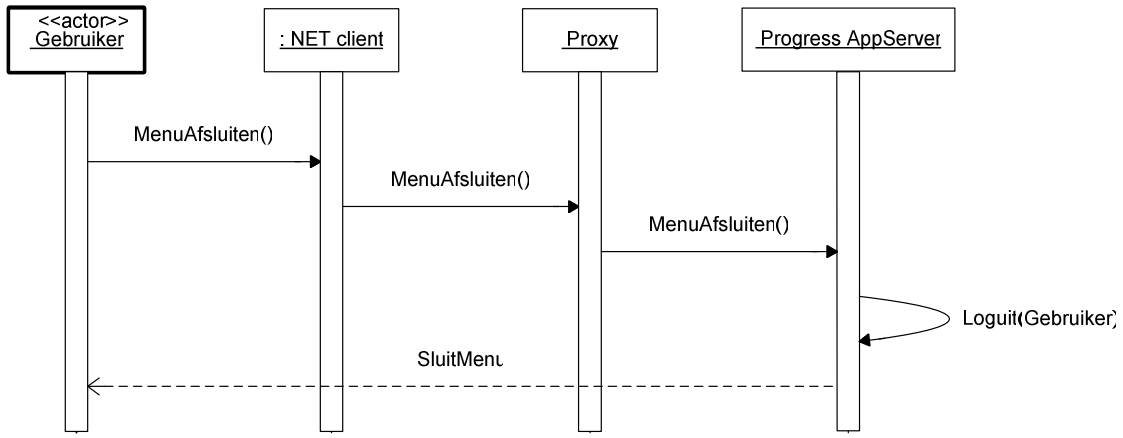
Figuur 3: *Sequencediagram programma starten*

De volgende activiteiten worden uitgevoerd om een programma te starten:

- De gebruiker geeft het systeem opdracht een programma te starten;
- De .NET client ontvangt de opdracht en geeft deze door aan het proxy-object;
- Het proxy-object ontvangt de opdracht en start een functie op de Progress AppServer waarmee programma's worden gestart;
- De Progress AppServer voert de betreffende functie uit, zodat het programma wordt gestart;
- Het programma wordt getoond aan de gebruiker.

2.3.4 Menu afsluiten

Het onderstaande sequencediagram geeft de interactie tussen de verschillende objecten weer, die benodigd is om een menu af te sluiten. Bij de use-case die in het rapport definitiestudie is opgenomen voor deze interactie, is ook rekening gehouden met de mogelijkheid van het opslaan van wijzigingen in het menu wanneer het menu wordt afgesloten. In deze pilot wordt deze functionaliteit echter nog niet geïmplementeerd. Om deze reden is dit aspect ook niet meegenomen in het onderstaande sequencediagram.



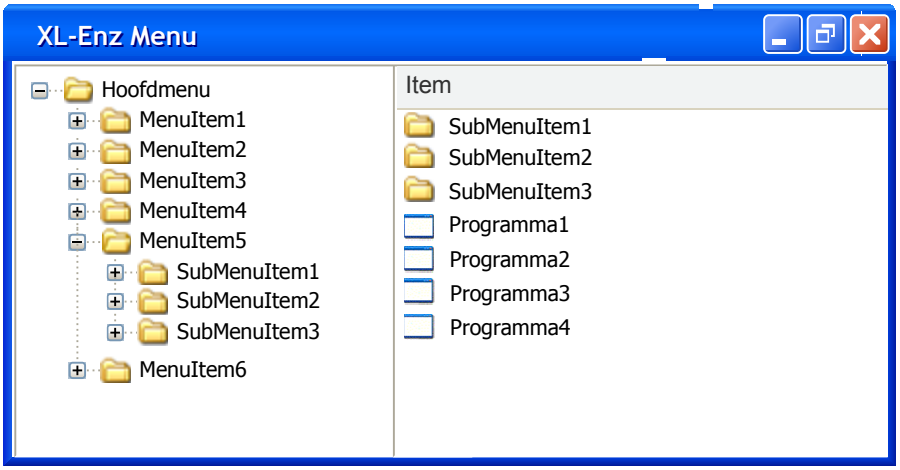
Figuur 4: *Sequencediagram menu afsluiten*

De volgende activiteiten worden uitgevoerd om een programma te starten:

- De gebruiker geeft het systeem opdracht het menu af te sluiten;
- De .NET client ontvangt de opdracht en geeft deze door aan het proxy-object;
- Het proxy-object ontvangt de opdracht en start een functie op de Progress AppServer waarmee de gebruiker wordt uitgelogd;
- Het menu wordt gesloten.

2.4 Prototype gebruikersinterface

Hieronder is een prototype opgenomen van de gebruikersinterface van het te ontwikkelen menu.



Figuur 5: *Prototype gebruikersinterface menu*

De belangrijkste kenmerken van de gebruikersinterface zijn:

- Aan de linkerkant van het scherm staan de menu-items in een boomstructuur. In deze boom zijn de programma's niet opgenomen;
- De rechterkant van het scherm bevat een overzicht van de inhoud van het menu-item dat in de boom is geselecteerd. In dit overzicht zijn zowel submenu-items als programma's opgenomen.

3. Technische structuur

3.1 Inleiding

Dit hoofdstuk beschrijft de technische aspecten met betrekking tot de ontwikkeling van de "Must Have" functionaliteiten van het menu. Deze aspecten zijn reeds globaal gedefinieerd tijdens de fase definitiestudie en zullen hier verder worden gedetailleerd. Als eerste worden in paragraaf 3.2 de verantwoordelijkheden van de objecten beschreven. Vervolgens wordt in paragraaf 3.3 het databasemodel weergegeven dat dient te worden gebruikt wanneer de menudata wordt opgevraagd uit de Progress database.

3.2 Fysieke allocatie

In het voorgaande hoofdstuk is reeds te zien hoe de interactie tussen de verschillende objecten verloopt. Elk object uit de sequencediagrammen levert één of meer functionaliteiten om een interactie uit te kunnen voeren. Hieronder worden de verantwoordelijkheden van de objecten uit beschreven.

- .NET client
  De .NET client toont het menu aan de gebruiker en ontvangt alle interacties die de gebruiker met het menu uitvoert. Tevens communiceert de .NET client met het proxy-object.

- Proxy
  Het proxy-object fungeert als een vertaler tussen de .NET omgeving en de Progress omgeving. De proxy neemt de data en communicatie vanuit de Progress of .NET omgeving in ontvangst en vertaalt deze naar begrijpelijke data voor de andere omgeving. De proxy bestaat uit een DLL (Dynamic Link Library).

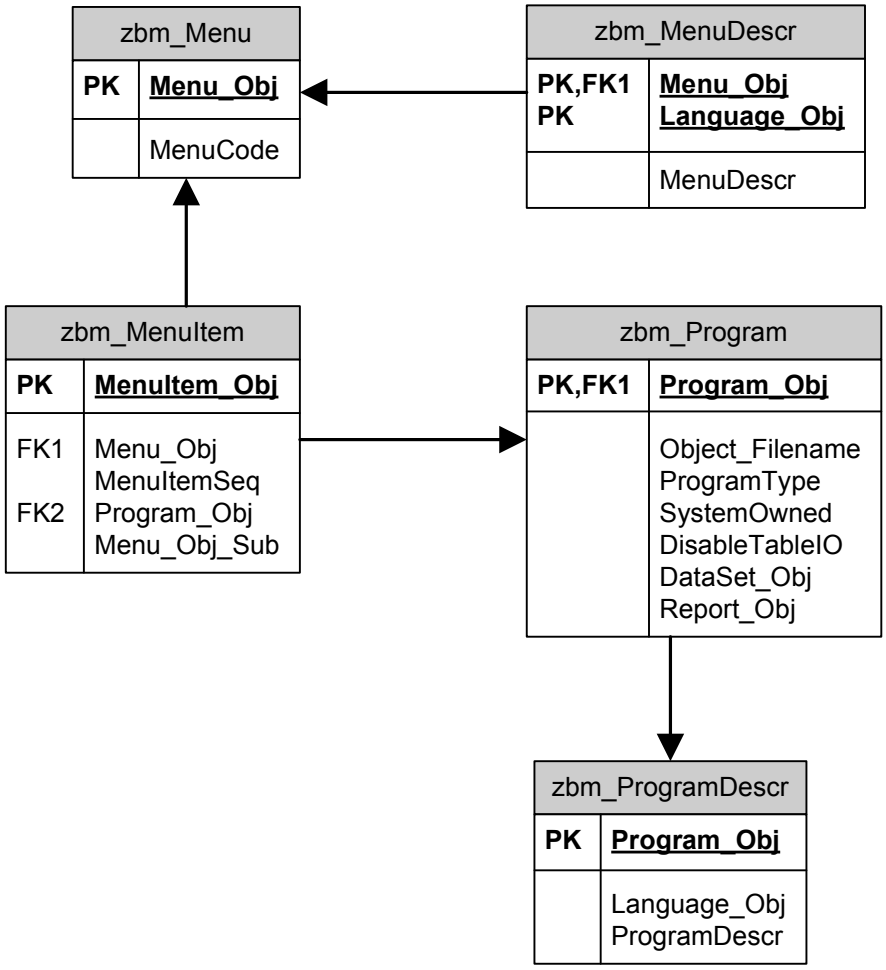
- Progress AppServer
  De AppServer bevat de business logica die benodigd is om functionaliteiten uit te kunnen voeren, zoals:
  - Menudata opvragen uit de Progress DBMS;
  - Programma starten;
  - Gebruiker uitloggen.

- Progress DBMS
  De Progress DBMS bevat de menudata die wordt opgevraagd door de AppServer.

Voor een gedetailleerde beschrijving van de technische aanpassingen die benodigd zijn om bovenstaande technische structuur te implementeren, wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET".

3.3 Databasemodel

De gegevens die benodigd zijn om het menu op te bouwen, zijn opgeslagen in een reeds bestaande Progress database. Hieronder is een schematisch overzicht gegeven van de structuur van deze gegevens.
In dit model zijn alleen de tabellen opgenomen die tijdens deze pilot benodigd zijn. De datatypen en definities van de relaties zijn niet in dit model opgenomen, vanwege het feit dat de tabellen reeds bestaan en de exacte definities kunnen worden opgevraagd vanuit de database van de applicatie XL-Enz.



Figuur 6: *Databasemodel menu*

Zoals hierboven te zien is, is de menudata opgeslagen in meerdere tabellen. Gegevens over menu-items en gegevens over programma's zijn opgeslagen in aparte tabellen.

Dit databasemodel is als volgt opgebouwd:

- De tabel "zbm_Menu" bevat informatie over alle menu-items uit het menu;
- De tabel "zbm_MenuDescr" bevat omschrijvingen van menu-items. Deze omschrijvingen worden opgenomen in het menu om de menu-items te identificeren;

- De tabel "zbm_MenuItem" bevat overige informatie over elk menu-item:
    - "MenuItemSeq" wordt gebruikt om de positie van het menu-item of het programma vast te leggen;
    - "Program_Obj" heeft een waarde ongelijk aan "0" wanneer het menu-item een programma is. In dit geval verwijst de waarde van het veld naar een record uit de tabel "zbm_Program", waarin meer informatie over het programma is vastgelegd;
    - "Menu_Obj_Sub" heeft een waarde ongelijk aan "0" wanneer het menu-item een submenu is van een ander menu-item. In dit geval verwijst de waarde van het veld naar een record uit de tabel "zbm_Menu".
- De tabel "zbm_Program" bevat informatie over alle programma's;
- De tabel "zbm_ProgramDescr" bevat de omschrijvingen van de programma's. Deze omschrijvingen worden opgenomen in het menu om de programma's te identificeren.

4. Pilotontwikkelplan

4.1 Inleiding

Het pilotontwikkelplan beschrijft hoe de pilot gaat worden gebouwd. Het pilotplan dat tijdens de definitiestudie is gemaakt wordt hier uitgebreid en eventueel aangepast. Per pilotdeel worden één of meer bouweenheden gedefinieerd. Bouweenheden zijn componenten van de pilot die door hun aard en reikwijdte achtereenvolgens of parallel kunnen worden ontwikkeld.
In paragraaf 4.2 worden de eerder gedefinieerde pilotdelen onderverdeeld in bouweenheden en wordt aangegeven in welke volgorde de bouweenheden worden ontwikkeld.

4.2 Pilotdelen en bouweenheden

Tijdens de fase definitiestudie is de pilot reeds opgedeeld in de onderstaande pilotdelen.

1. Ontwikkelen menu-dataobject in Progress OpenEdge 10;
2. Converteren Progress menu-dataobject naar .NET menu-dataobject;
3. Ontwikkelen functionaliteiten menu;
4. Menu als ActiveX object invoegen in Progress;
5. Ontwikkelen mogelijkheid opstarten van programma's.

Hieronder worden per pilotdeel bouweenheden gedefinieerd.

4.2.1 Ontwikkelen menu-dataobject in Progress OpenEdge 10

Als eerste wordt het Progress dataobject in de vorm van een ProDataSet-object gebouwd. Dit object bevat alle data uit de tabellen die zijn gemodelleerd in het databasemodel in het voorgaande hoofdstuk.
De volgende bouweenheden worden voor dit pilotdeel achtereenvolgens ontwikkeld:

1. Opzetten ontwikkelomgeving;
2. Bouwen functie voor het opnemen van de menudata in een Progress ProDataSet-object.

4.2.2 Converteren Progress menu-dataobject naar .NET menu-dataobject

Nadat het Progress dataobject is ontwikkeld, dient dit object te worden geconverteerd naar een .NET dataobject, zodat de menu-data toegankelijk wordt in de .NET omgeving.
Dit pilotdeel bestaat uit één bouweenheid:

3. Converteren Progress menu-dataobject naar .NET menu-dataobject.

4.2.3 Ontwikkelen functionaliteiten menu

Wanneer de menudata toegankelijk is in de .NET omgeving, kunnen de functionaliteiten van het menu in deze omgeving worden geïmplementeerd.
De volgende bouweenheden worden voor dit pilotdeel achtereenvolgens ontwikkeld:

4. Menu initialiseren op basis van menu-dataobject;
5. Ontwikkelen navigatiemogelijkheid;
6. Ontwikkelen mogelijkheid tot afsluiten menu.

4.2.4 Menu als ActiveX object invoegen in Progress

Wanneer de bovenstaande functionaliteiten zijn geïmplementeerd, kan het menu in de Progress omgeving worden ingevoegd als een ActiveX object.
De volgende bouweenheden worden voor dit pilotdeel achtereenvolgens ontwikkeld:

7. Menu compileren en registreren als ActiveX object;
8. Menu als ActiveX object invoegen in Progress.

4.2.5 Ontwikkelen mogelijkheid opstarten van programma's

Het laatste pilotdeel bestaat uit het implementeren van de mogelijkheid om programma's op te starten.

9. Ontwikkelen mogelijkheid opstarten van programma's.

4.3 Iteratiestrategie

De pilot wordt op een iteratieve wijze ontwikkeld. Dit houdt in dat de fase pilotontwikkeling meerdere malen kan worden doorlopen, waardoor de pilot op een evoluerende wijze wordt geïmplementeerd.
Er is voor gekozen om tijdens de eerste iteratie de pilot te implementeren in een lokale omgeving. Deze keuze komt voort uit het feit dat het gebruik van nieuwe technieken (.NET, ProDataSet-object) onverwachte nadelige gevolgen kan hebben op de bestaande hard- en software. Door eerst de pilot in een lokale omgeving te implementeren, kunnen deze gevolgen worden onderzocht, zonder dat deze verdere gevolgen hebben voor de hard- en software die wordt gebruikt voor de ontwikkeling van de applicatie XL-Enz.

Na de eerste iteratie wordt de pilot getest op een correcte werking, op functionaliteit en op performance. Wanneer hieruit blijkt dat de pilot verder kan worden ontwikkeld, kan in een volgende iteratie de implementatie plaatsvinden in een gedistribueerde omgeving.

5. Ontwerp software-bouweenheden

5.1 Inleiding

Het doel van dit hoofdstuk is om van de hiervoor gedefinieerde bouweenheden de technische structuur te specificeren. Daarnaast worden in dit hoofdstuk testspecificaties opgesteld voor de bouweenheden. Op basis van het ontwerp en de testspecificaties uit dit hoofdstuk dient het mogelijk te zijn om de bouweenheden te implementeren en vervolgens te testen.
De opbouw van dit hoofdstuk is als volgt. In paragraaf 5.2 wordt voor elke bouweenheid een ontwerp gemaakt, op basis waarvan het mogelijk dient te zijn de bouweenheden te implementeren. Vervolgens worden in paragraaf 5.3 testspecificaties opgesteld, die dienen te worden uitgevoerd nadat een bouweenheid is geïmplementeerd.

5.2 Ontwerp bouweenheden

5.2.1 Bouweenheid 1: Opzetten ontwikkelomgeving

De ontwikkelomgeving die is gebruikt tijdens de voorgaande pilot (Ontwikkelen van .NET dataobject voor Crystal Reports), kan tijdens deze pilot eveneens worden gebruikt.
De benodigde onderdelen om deze pilot te kunnen ontwikkelen, zijn:

- Microsoft Windows besturingssysteem (bij voorkeur Windows XP);
- Progress OpenEdge 10 ontwikkelomgeving;
- Progress AppServer;
- Progress DBMS (database van XL-Enz);
- Microsoft .NET Framework (versie 1.1 of hoger);
- Visual Studio .NET 2003 ontwikkelomgeving.

5.2.2 Bouweenheid 2: Bouwen Progress menu-dataobject

Wanneer de ontwikkelomgeving is opgezet, kan het Progress ProDataSet-object worden geïmplementeerd dat de menudata zal bevatten. Het databasemodel uit hoofdstuk 3 van dit document kan bij deze bouweenheid als uitgangspunt fungeren. Het ProDataSet-object dient alle data uit de tabellen uit het databasemodel te bevatten.

Voor gedetailleerde informatie over de wijze van implementatie van een Progress ProDataSet-object wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET". Daarnaast biedt de documentatie die Progress beschikbaar stelt over de implementatie van een ProDataSet-object voldoende informatie om deze bouweenheid te kunnen implementeren.

5.2.3 Bouweenheid 3: Converteren Progress dataobject naar .NET dataobject

Wanneer het ProDataSet-object is geïmplementeerd en getest, kan deze worden geconverteerd naar een .NET dataobject. Hiermee wordt de menudata toegankelijk in de .NET omgeving. Ook de functie waarin het Progress dataobject wordt samengesteld dient toegankelijk te zijn in de .NET omgeving.
Deze bouweenheid kan worden geïmplementeerd door gebruik te maken van de functionaliteiten van de tool "Proxy Generator" (ProxyGen), die standaard wordt meegeleverd met Progress OpenEdge 10. Deze tool genereert een proxy-object (Dynamic Link Library (DLL)), dat de communicatie en conversie afhandelt tussen de Progress omgeving en de .NET omgeving.

Voor gedetailleerde informatie over het genereren van het proxy-object wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET". Daarnaast biedt de documentatie die Progress beschikbaar stelt over het genereren van proxy-objecten voldoende informatie om deze bouweenheid te kunnen implementeren.

5.2.4 Bouweenheid 4: Menu initialiseren op basis van menu-dataobject

Wanneer het proxy-object is gegenereerd, kan het menu worden gebouwd in de .NET omgeving (met behulp van Visual Studio .NET 2003).
Enkele aandachtspunten bij de implementatie van het menu:

- Omdat het menu als ActiveX object in Progress ingevoegd gaat worden, is het noodzakelijk om het menu te implementeren als een Windows UserControl (en niet als een Windows Application). Hiervoor kan in Visual Studio .NET 2003 de template "Windows Control Library" worden gebruikt;
- Het prototype van de gebruikersinterface van het menu dat in hoofdstuk 2 is opgesteld, kan als uitgangspunt worden gebruikt bij de ontwikkeling van het menu;
- De boomstructuur uit het menu kan worden geïmplementeerd door gebruik te maken van de klasse "TreeView";
- De lijst met items (menu-items en programma's) kan worden geïmplementeerd door gebruik te maken van de klasse "ListView".

5.2.5 Bouweenheid 5: Ontwikkelen navigatiemogelijkheid

Nadat het menu is geïnitialiseerd, wordt de navigatiemogelijkheid ontwikkeld. Deze bouweenheid bestaat uit het implementeren van de functionaliteiten van de boomstructuur. De werking van de navigatiemogelijkheden dient gelijk te zijn aan de werking van de navigatie door een Windows Explorer.
Voor nadere informatie over de implementatie van deze functionaliteiten wordt verwezen naar de documentatie van Visual Studio .NET 2003.

5.2.6 Bouweenheid 6: Ontwikkelen mogelijkheid tot afsluiten menu

Wanneer de gebruiker het menu afsluit, dient een functie op de Progress AppServer te worden uitgevoerd waarmee de gebruiker wordt uitgelogd uit de applicatie en waarmee de verbinding tussen de .NET client en de Progress AppServer wordt beëindigd.
Voor meer informatie over het opbouwen en beëindigen van dergelijke verbindingen, wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET" en naar de documentatie van Visual Studio .NET 2003.

5.2.7 Bouweenheid 7: Menu compileren en registreren als ActiveX object

Wanneer de bovenstaande bouweenheden succesvol zijn geïmplementeerd en getest, kan het menu als een ActiveX object worden ingevoegd in een Progress omgeving. Hiervoor dient het menu als een COM control te worden gecompileerd en vervolgens als zodanig in de Windows Registry te worden geregistreerd.
Enkele aandachtspunten bij de implementatie van deze bouweenheid:

- De optie "Register for COM Interop" in Visual Studio .NET 2003 dient de waarde "True" te hebben;
- Om het menu succesvol te compileren, dient in Visual Studio .NET 2003 "Assembly Info" te worden toegekend aan het menu. (Deze informatie wordt opgeslagen in het DLL bestand dat na compilatie is aangemaakt en maakt het DLL bestand uniek binnen het systeem).

Voor nadere informatie voor de implementatie van deze bouweenheid wordt verwezen naar de documentatie van Visual Studio .NET 2003.

5.2.8 Bouweenheid 8: Menu als ActiveX object invoegen in Progress

Nadat het menu succesvol is gecompileerd en correct is geregistreerd, dient het in Progress te worden ingevoegd als een ActiveX object.
Hiervoor dient met behulp van de Progress AppBuilder een nieuwe Progress applicatie aangemaakt te worden, waarin grafische elementen kunnen worden opgenomen.
Het menu kan worden ingevoegd door deze in de lijst met beschikbare ActiveX objecten te selecteren en vervolgens in de zojuist aangemaakte applicatie op te nemen.

5.2.9 Bouweenheid 9: Ontwikkelen mogelijkheid opstarten van programma's

Wanneer het menu succesvol is ingevoegd in een Progress applicatie, kan de mogelijkheid tot het opstarten van programma's uit menu worden ontwikkeld.
Wanneer een gebruiker in het menu dubbelklikt op de naam van een programma, dient vanuit de Progress applicatie een functie te worden uitgevoerd waarmee het betreffende programma wordt gestart.

5.3 Testspecificaties

Na implementatie van een bouweenheid wordt deze getest aan de hand van testspecificaties. Zoals is vastgesteld in het document plan van aanpak van dit project, wordt de teststrategie "black box testing" gehanteerd. Hierbij wordt niet gekeken naar de inwendige werking van de bouweenheden, maar alleen naar het resultaat dat dient te worden geleverd op basis van een bepaalde invoer.
Hieronder zijn per bouweenheid testspecificaties opgesteld die worden uitgevoerd nadat een bouweenheid is geïmplementeerd.

5.3.1 Testspecificaties bouweenheid 1

In deze bouweenheid ("Opzetten ontwikkelomgeving") wordt geen software ontwikkeld die dient te worden getest. Om deze reden zijn voor bouweenheid 1 geen testspecificaties opgesteld.
Controleren of de benodigde onderdelen uit de ontwikkelomgeving zijn geïnstalleerd, kan worden uitgevoerd door de lijst van geïnstalleerde software op te vragen en vervolgens na te gaan of het betreffende onderdeel zich op deze lijst bevindt. Deze lijst kan als volgt worden opgevraagd: Klik "Start" – "Control Panel" – "Add or remove programs".
Testen of de benodigde Progress database correct functioneert, kan worden gedaan door gebruik te maken van de functionaliteiten die de Progress omgeving hiervoor biedt.

5.3.2 Testspecificaties bouweenheid 2

Deze bouweenheid heeft als resultaat een functie die in een Progress omgeving een ProDataSet-object oplevert, die de menu-data bevat.
Hieronder worden testspecificaties gegeven waarbij getest wordt of het ProDataSet-object de juiste data en relaties tussen de tabellen bevat.
Om de testspecificaties overzichtelijk te houden, is geen broncode in de testspecificaties opgenomen, maar alleen een tekstuele beschrijving.

De tests dienen te worden uitgevoerd op een ProDataSet-object dat aan de volgende kenmerken voldoet:

- Het object bevat 5 TempTables: "tt_zbm_Menu", "tt_zbm_MenuDescr", "tt_zbm_MenuItem", "tt_zbm_Program" en "tt_zbm_ProgramDescr". Deze tabellen zijn gebaseerd op de tabellen die zijn opgenomen in het databasemodel uit hoofdstuk 3 van dit rapport;
- Er zijn relaties tussen de tabellen gedefinieerd zoals die zijn opgenomen in het databasemodel uit hoofdstuk 3 van dit rapport;
- Het ProDataSet-object is gevuld op basis van deze relaties.

Testspecificaties bouweenheid 2: Bouwen Progress menu-dataobject			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
2.1	ProDataSet is correct gevuld	Opvragen aantal records uit alle tabellen (*)	# records "tt_zbm_Menu": 43 # records "tt_zbm_MenuDescr": 43 # records "tt_zbm_MenuItem": 225 # records "tt_zbm_Program": 184 # records "tt_zbm_ProgramDescr": 184
2.2	Relatie tussen "tt_zbm_Menu" en "tt_zbm_MenuDescr" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Menu_obj, Menu_obj"
2.3	Relatie tussen "tt_zbm_Menu" en "tt_zbm_MenuItem" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Menu_obj, Menu_obj"
2.4	Relatie tussen "tt_zbm_MenuItem" en "tt_zbm_Program" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Program_obj, Program_obj"
2.5	Relatie tussen "tt_zbm_Program" en "tt_zbm_ProgramDescr" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Program_obj, Program_obj"

(*) De uitvoer van deze test kan verschillen met de uitvoer die hier wordt verwacht. In dit geval dienen de volgende extra tests te worden uitgevoerd:

- Het aantal records uit "tt_zbm_Menu" is gelijk aan het aantal records uit "tt_zbm_MenuDescr";
- Het aantal records uit "tt_zbm_Program" is gelijk aan het aantal records uit "tt_zbm_ProgramDescr".

5.3.3 Testspecificaties bouweenheid 3

Bouweenheid 3 "Converteren Progress dataobject naar .NET dataobject" levert een proxy-object op dat de communicatie en conversie afhandelt tussen de Progress omgeving en de .NET omgeving.
Door middel van het uitvoeren van onderstaande testcases kan worden getest of het gegenereerde proxy-object het Progress ProDataSet-object correct vertaalt naar een .NET DataSet object.

Om deze tests uit te kunnen voeren, dient in Visual Studio .NET 2003 een nieuw project aangemaakt te worden. Hierin dient in een Windows Form een DataGrid-object te worden opgenomen dat de data uit het ProDataSet-object zal gaan bevatten. Voor een stapsgewijze beschrijving van de wijze waarop een dergelijk Windows Form wordt gebouwd, wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET".

Testspecificaties bouweenheid 3: Converteren Progress dataobject naar .NET dataobject			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
3.1	DataGrid is correct gevuld	Opvragen aantal records uit alle tabellen (*)	# records "tt_zbm_Menu": 43 # records "tt_zbm_MenuDescr": 43 # records "tt_zbm_MenuItem": 225 # records "tt_zbm_Program": 184 # records "tt_zbm_ProgramDescr": 184
3.2	Relaties tussen tabellen zijn correct gedefinieerd	Gebruik maken van "browse" mogelijkheid tussen tabellen in DataGrid	Relaties zijn gelijk aan de relaties zoals die zijn gedefinieerd in het Progress ProDataSet-object (zie testcases 2.2 t/m 2.5)

(*) De uitvoer van deze test kan verschillen met de uitvoer die hier wordt verwacht. In dit geval dienen de volgende extra tests te worden uitgevoerd:

- Het aantal records uit "tt_zbm_Menu" is gelijk aan het aantal records uit "tt_zbm_MenuDescr";
- Het aantal records uit "tt_zbm_Program" is gelijk aan het aantal records uit "tt_zbm_ProgramDescr".

5.3.4 Testspecificaties bouweenheid 4

Bouweenheid 4 levert een .NET UserControl op waarin het menu wordt geïnitialiseerd op basis van het eerder geïmplementeerde dataobject.
Door middel van het uitvoeren van onderstaande testcases kan worden getest of het .NET UserControl aan de gestelde eisen voldoet.

Testspecificaties bouweenheid 4: Menu initialiseren op basis van menu-dataobject			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
4.1	Het menu wordt correct geïnitialiseerd	Compileren en uitvoeren UserControl	Overzicht van het hoofdmenu van XL-Enz
4.2	Menu-items zijn correct geordend	Geïnitialiseerd menu	Menu-items zijn geordend op basis van het veld "MenuItemSeq" uit "tt_zbm_MenuItem"

5.3.5 Testspecificaties bouweenheid 5

In bouweenheid 5 worden de navigatiemogelijkheden van het menu geïmplementeerd. Door middel van het uitvoeren van onderstaande testcases kan worden getest of de werking van de navigatie aan de gestelde eisen voldoet.

Testspecificaties bouweenheid 5: Ontwikkelen navigatiemogelijkheid			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
5.1	Werking navigatie in TreeView	Klik op menu-item	Inhoud van geselecteerd menu-item is zichtbaar in ListView
5.2		Dubbelklik op menu-item in TreeView waarvan inhoud niet zichtbaar is	Submenu('s) van geselecteerd item is/zijn zichtbaar in TreeView en submenu('s) en programma('s) is/zijn zichtbaar in ListView
5.3		Dubbelklik op menu-item in TreeView waarvan inhoud zichtbaar is	Submenu('s) van geselecteerd item is/zijn onzichtbaar in TreeView en submenu('s) en programma('s) is/zijn zichtbaar in ListView
5.4		Klik op een "-" (collapse) icoon voor menu-item	Inhoud van betreffende menu-item is onzichtbaar
5.5		Klik op een "+" (expand) icoon voor menu-item	Inhoud van betreffende menu-item is zichtbaar
5.6	Werking navigatie in ListView	Klik op menu-item in ListView	Geen uitvoer
5.7		Dubbelklik op menu-item in ListView	Inhoud geselecteerd menu-item is zichtbaar in ListView, geselecteerd menu-item is ook in TreeView zichtbaar en is geselecteerd
5.8		Dubbelklik op programmanaam in ListView	Geen uitvoer

5.3.6 Testspecificaties bouweenheid 6

Bouweenheid 6 heeft als resultaat de mogelijkheid tot het afsluiten van het menu opgeleverd. Door middel van het uitvoeren van onderstaande testcases kan worden getest of het afsluiten correct verloopt.

Testspecificaties bouweenheid 6: Ontwikkelen mogelijkheid tot afsluiten menu			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
6.1	Gebruiker is uitgelogd	Opvragen "Status" van Progress AppServer	Aantal "Active Clients" is één lager dan wanneer de gebruiker is ingelogd (wanneer het menu wordt getoond)
6.2	Menu afsluiten	Klik op knop dat menu doet afsluiten	Menu wordt niet meer getoond

5.3.7 Testspecificaties bouweenheid 7

In bouweenheid 7 is het menu zodanig gecompileerd en geregistreerd in de Windows Registry dat het systeem het menu herkent als een ActiveX object. Door middel van het uitvoeren van onderstaande testcases kan worden getest of het compileren en registreren correct is verlopen.

Testspecificaties bouweenheid 7: Menu compileren en registreren als ActiveX object			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
7.1	Menu is gecompileerd als ActiveX object	Controle op uitvoer compilatie	De map *Projectnaam*\bin\debug bevat het menu in de vorm van een DLL bestand
7.2	DLL bestand is "ondertekend" (signed)	Controle op eigenschappen DLL bestand	DLL bestand bevat "Version Information" en kan in de "Global Assembly Cache" worden geplaatst
7.3	ActiveX object is correct geregistreerd in de Windows Registry	Controle op correcte registratie	Zie hieronder

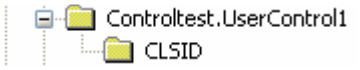
Verwachte uitvoer bij testcase 7.3
Bij deze test dient een controle op enkele waarden uit de Windows Registry te worden uitgevoerd. De Windows Registry kan worden bekeken door het uitvoeren van "regedit". Het register dient enkele waarden te bevatten voor het geregistreerde ActiveX object. Hieronder een overzicht van deze waarden. Hierbij zijn:

- HKCR: de Windows Registry root-entry "HKEY_CLASSES_ROOT";
- ProgID: de naam van het geregistreerde ActiveX object;
- CLSID: de Class ID die is gegenereerd voor het ActiveX object, bijvoorbeeld "D63CD054-1247-4853-AF05-B7D26D993E85".

De Windows Registry dient de volgende entries te bevatten:

```
HKCR\ProgID\(Default)="ActiveX_Object_Name"
HKCR\ProgID\CLSID\(Default)="{CLSID}"
```

Voor een ActiveX object met de naam "Controltest.UserControl1" zouden deze entries er als volgt uitzien:

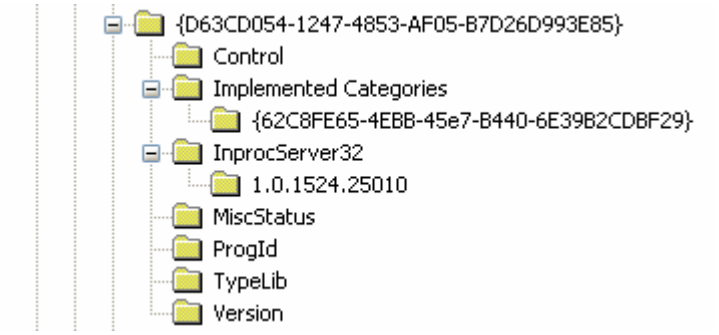


Figuur 7: *Entries in Windows Registry (1)*

Naast bovenstaande entries dienen ook de onderstaande entries te zijn toegevoegd:

```
HKCR\CLSID\{CLSID}\(Default)="ActiveX_Object_Name"
HKCR\CLSID\{CLSID}\Control\(Default)="Value not Set"
HKCR\CLSID\{CLSID}\Implemented Categories\(Default)="Value not
Set"
HKCR\CLSID{CLSID}\Implemented Categories\{62C8FE65-4EBB-45e7-B440-
6E39B2CDBF29}
HKCR\CLSID\{CLSID}\InprocServer32\(Default)="WindowsSystemDirector
y\mscoree.dll"
HKCR\CLSID\{CLSID}\InprocServer32\ThreadingModel="Both"
HKCR\CLSID\{CLSID}\InprocServer32\Class="NamespaceQualifiedClassNa
me"
HKCR\CLSID\{CLSID}\InprocServer32\Assembly="FullAssemblyName"
HKCR\CLSID\{CLSID}\InprocServer32\RuntimeVersion="Version"
HKCR\CLSID\{CLSID}\InprocServer32\CodeBase="FullPath_to_Assembly"
HKCR\CLSID\{CLSID}\MiscStatus\(Default)="131457"
HKCR\CLSID\{CLSID}\ProgId\(Default)="ActiveX_Object_Name"
HKCR\CLSID\{CLSID}\TypeLib\(Default)="66d15bb5-63d8-31c7-b880-
a8e4b1055359"
HKCR\CLSID\{CLSID}\Version\(Default)="1.0"
```

Voor een ActiveX object met de naam "Controltest.UserControl1" zouden deze entries er als volgt uitzien:



Figuur 8: *Entries in Windows Registry (2)*

5.3.8 Testspecificaties bouweenheid 8

In bouweenheid 8 wordt het menu als ActiveX object ingevoegd in een Progress applicatie. Door middel van het uitvoeren van onderstaande testcases kan worden getest of het invoegen van het menu correct is verlopen.

Testspecificaties bouweenheid 8: Menu als ActiveX object invoegen in Progress			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
8.1	Menu komt als ActiveX object voor in de lijst met beschikbare ActiveX objecten	Open "OCX" dialoog box in AppBuilder	Menu staat in de lijst beschikbare ActiveX objecten
8.2	Menu kan worden ingevoegd in Progress applicatie	Selecteer menu uit lijst beschikbare ActiveX objecten en plaats menu op Progress Window	Menu is als ActiveX object in de Progress applicatie opgenomen
8.3	Menu wordt correct geïnitialiseerd	Uitvoeren Progress applicatie waarin menu is opgenomen	Menu is geïnitialiseerd
8.4	Functionaliteiten menu werken correct	Navigatie door menu	Navigatie werkt zoals geïmplementeerd in bouweenheid 5

5.3.9 Testspecificaties bouweenheid 9

Als laatste bouweenheid van deze pilot is de mogelijkheid tot het opstarten van programma's geïmplementeerd. Door middel van het uitvoeren van onderstaande testcase kan worden getest of het opstarten van programma's correct verloopt.

Testspecificaties bouweenheid 9: Ontwikkelen mogelijkheid opstarten van programma's			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
9.1	Starten programma	Dubbelklik op programmanaam uit menu	Het betreffende programma is gestart

6. Invoeringsprocedure pilot

6.1 Inleiding

Wanneer het menu de functionaliteiten bevat die tijdens deze pilot zijn geïmplementeerd en wanneer de testgevallen succesvol zijn doorlopen, kan worden overgegaan tot het operationeel maken van de pilot. Deze beslissing wordt genomen door de opdrachtgever. Dit hoofdstuk beschrijft de procedure die dient te worden gevolgd wanneer de pilot wordt ingevoerd.
In paragraaf 6.2 wordt een plan opgesteld dat als uitgangspunt dient bij de invoering van de pilot. Vervolgens worden in paragraaf 6.3 testspecificaties opgesteld. De testgevallen kunnen worden uitgevoerd nadat de pilot is ingevoerd en dienen ter controle op een correcte invoering.

6.2 Invoeringsplan

Hieronder is een plan voor invoering van de pilot gegeven in de vorm van een stapsgewijze beschrijving. Hierbij wordt ervan uitgegaan dat de pilot wordt ingevoerd als onderdeel van de applicatie XL-Enz.

1. Controleren aanwezigheid benodigde hard- en software:
    - Ontwikkelomgeving Progress (OpenEdge 10);
    - Toegang tot de broncode van de applicatie XL-Enz;
    - Toegang tot de database waar XL-Enz gebruik van maakt;
    - Microsoft .NET Framework (versie 1.1 of hoger) is geïnstalleerd op de client;
    - Het menu in de vorm van een ActiveX (DLL) object;
    - Het gegenereerde proxy-object;
    - Progress applicatie waarin het menu is geplaatst.

2. Installeren Progress functie waarmee menu-data wordt opgehaald uit de database;

3. Plaatsen proxy-object en benodigde assembly bestanden in de juiste directory. Voor gedetailleerde informatie hierover wordt verwezen naar hoofdstuk 5 van het rapport "Interfacing tussen Progress en Microsoft .NET";

4. Vervangen huidige menu van XL-Enz met nieuw menu.

De verantwoordelijkheden rond de invoering van de pilot worden opgenomen door het ontwikkelteam van Reflecta Automation.

6.3 Testspecificaties invoeringsprocedure

Hieronder is een lijst met testspecificaties opgenomen die dienen ter controle op een correcte werking van het betreffende deel uit het invoeringsplan.
Bij deze testspecificaties wordt de teststrategie "black box testing" gehanteerd.

Testspecificaties invoering pilot 2			
Case #	**Test**	**Invoer**	**Verwachte uitvoer**
1	Alle benodigde hard- en software is aanwezig	Opvragen lijst met geïnstalleerde hard- en software op betreffende systeem	Benodigde hard- en software is aanwezig op het juiste systeem
2	ProDataSet-object is aangemaakt en correct gevuld	Uitvoeren van functie waarmee menu-data wordt opgevraagd en in ProDataSet wordt geplaatst	ProDataSet-object is aangemaakt en bevat de correcte gegevens uit juiste de tabellen (zie databasemodel, hoofdstuk 3)
3	Menu wordt geïnitialiseerd	Starten applicatie waar menu in is opgenomen	Menu is geïnitialiseerd en toont het hoofdmenu van de applicatie XL-Enz
4	Navigatie menu werkt correct	Navigatie door menu	Navigatie werkt volgens opgestelde eisen
5	Starten programma	Dubbelklik op programmanaam uit menu	Het betreffende programma is gestart
6	Menu afsluiten	Klik op knop dat menu doet afsluiten	Menu wordt niet meer getoond, gebruiker is uitgelogd

7. Testen en beoordelen pilot

7.1 Inleiding

Na implementatie van elke bouweenheid is deze getest aan de hand van de testspecificaties uit hoofdstuk 5 van dit document. Dit hoofdstuk beschrijft de testresultaten na implementatie van de bouweenheden tijdens de eerste iteratieslag van de pilot. In paragraaf 7.2 wordt de testomgeving beschreven die is gebruikt om de pilot te testen. Vervolgens worden in de paragrafen 7.3 tot en met 7.11 de testresultaten beschreven. In paragraaf 7.12 wordt de pilot beoordeeld op basis van de testresultaten.

7.2 Testomgeving

Er is voor gekozen om de pilot in de eerste iteratie te implementeren in een lokale omgeving. Het systeem dat voor deze implementatie is gebruikt, had de onderstaande specificaties:

- HP Intel Pentium 4, 2 x 2.60 GHz;
- 248 MB RAM.

De onderstaande software was hierop geïnstalleerd:

- Microsoft Windows XP besturingssysteem;
- Progress OpenEdge 10A ontwikkelomgeving;
- Progress AppServer;
- Progress DBMS (XL-Enz database);
- Microsoft .NET Framework 1.1;
- Visual Studio .NET 2003 ontwikkelomgeving.

7.3 Testen bouweenheid 1

Deze bouweenheid ("Opzetten ontwikkelomgeving") is getest door de lijst met geïnstalleerde software op te vragen. Volgens deze lijst was (onder meer) de volgende software geïnstalleerd:

- Progress OpenEdge 10A ontwikkelomgeving;
- Microsoft .NET Framework 1.1;
- Visual Studio .NET 2003 ontwikkelomgeving.

Het Windows XP besturingssysteem en de Progress AppServer functioneerde naar behoren.
De werking van de Progress XL-Enz database is getest door in de Progress omgeving een connectie te maken met deze database (d.m.v. de tool "Data Administration"). Deze tool gaf aan dat de verbinding met de database succesvol was.
Deze bouweenheid heeft geen problemen opgeleverd.

7.4 Testen bouweenheid 2

In bouweenheid 2 is in een Progress omgeving een functie geïmplementeerd waarin een ProDataSet-object wordt gecreëerd, dat de menu-data bevat volgens de opgestelde eisen.
Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 2: Bouwen Progress menu-dataobject				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
2.1	ProDataSet is correct gevuld	Opvragen aantal records uit alle tabellen	# records "tt_zbm_Menu": 43 # records "tt_zbm_MenuDescr": 43 # records "tt_zbm_MenuItem": 225 # records "tt_zbm_Program": 184 # records "tt_zbm_ProgramDescr": 184	OK
2.2	Relatie tussen "tt_zbm_Menu" en "tt_zbm_MenuDescr" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Menu_obj, Menu_obj"	OK
2.3	Relatie tussen "tt_zbm_Menu" en "tt_zbm_MenuItem" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Menu_obj, Menu_obj"	OK
2.4	Relatie tussen "tt_zbm_MenuItem" en "tt_zbm_Program" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Program_obj, Program_obj"	OK
2.5	Relatie tussen "tt_zbm_Program" en "tt_zbm_ProgramDescr" is correct gedefinieerd	Opvragen velden uit relatie (attribuut RELATION-FIELDS)	"Program_obj, Program_obj"	OK

7.5 Testen bouweenheid 3

Bouweenheid 3 "Converteren Progress dataobject naar .NET dataobject" heeft een proxy-object opgeleverd dat de communicatie en conversie afhandelt tussen de Progress omgeving en de .NET omgeving.
Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid. Om deze tests uit te kunnen voeren, is in Visual Studio .NET 2003 een nieuw project aangemaakt. Hierin is in een Windows Form een DataGrid-object opgenomen dat de data uit het ProDataSet-object bevatte.

Testspecificaties bouweenheid 3: Converteren Progress dataobject naar .NET dataobject				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
3.1	DataGrid is correct gevuld	Opvragen aantal records uit alle tabellen	# records "tt_zbm_Menu": 43 # records "tt_zbm_MenuDescr": 43 # records "tt_zbm_MenuItem": 225 # records "tt_zbm_Program": 184 # records "tt_zbm_ProgramDescr": 184	OK
3.2	Relaties tussen tabellen zijn correct gedefinieerd	Gebruik maken van "browse" mogelijkheid tussen tabellen in DataGrid	Relaties zijn gelijk aan de relaties zoals die zijn gedefinieerd in het Progress ProDataSet-object (zie testcases 2.2 t/m 2.5)	OK

7.6 Testen bouweenheid 4

Bouweenheid 4 heeft een .NET UserControl opgeleverd waarin het menu wordt geïnitialiseerd op basis van het eerder geïmplementeerde dataobject.
Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 4: Menu initialiseren op basis van menu-dataobject				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
4.1	Het menu wordt correct geïnitialiseerd	Compileren en uitvoeren UserControl	Overzicht van het hoofdmenu van XL-Enz	OK
4.2	Menu-items zijn correct geordend	Geïnitialiseerd menu	Menu-items zijn geordend op basis van het veld "MenuItemSeq" uit "tt_zbm_MenuItem"	OK

7.7 Testen bouweenheid 5

In bouweenheid 5 zijn de navigatiemogelijkheden van het menu geïmplementeerd. Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 5: Ontwikkelen navigatiemogelijkheid				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
5.1	Werking navigatie in TreeView	Klik op menu-item	Inhoud van geselecteerd menu-item is zichtbaar in ListView	OK
5.2		Dubbelklik op menu-item in TreeView waarvan inhoud niet zichtbaar is	Submenu('s) van geselecteerd item is/zijn zichtbaar in TreeView en submenu('s) en programma('s) is/zijn zichtbaar in ListView	OK
5.3		Dubbelklik op menu-item in TreeView waarvan inhoud zichtbaar is	Submenu('s) van geselecteerd item is/zijn onzichtbaar in TreeView en submenu('s) en programma('s) is/zijn zichtbaar in ListView	OK
5.4		Klik op een "-" (collapse) icoon voor menu-item	Inhoud van betreffende menu-item is onzichtbaar	OK
5.5		Klik op een "+" (expand) icoon voor menu-item	Inhoud van betreffende menu-item is zichtbaar	OK
5.6	Werking navigatie in ListView	Klik op menu-item in ListView	Geen uitvoer	OK
5.7		Dubbelklik op menu-item in ListView	Inhoud geselecteerd menu-item is zichtbaar in ListView, geselecteerd menu-item is ook in TreeView zichtbaar en is geselecteerd	OK
5.8		Dubbelklik op programmanaam in ListView	Geen uitvoer	OK

7.8 Testen bouweenheid 6

Bouweenheid 6 had als resultaat de mogelijkheid tot het afsluiten van het menu. Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 6: Ontwikkelen mogelijkheid tot afsluiten menu				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
6.1	Gebruiker is uitgelogd	Opvragen "Status" van Progress AppServer	Aantal "Active Clients" is één lager dan wanneer de gebruiker is ingelogd (wanneer het menu wordt getoond)	OK
6.2	Menu afsluiten	Klik op knop dat menu doet afsluiten	Menu wordt niet meer getoond	OK

7.9 Testen bouweenheid 7

In bouweenheid 7 is het menu zodanig gecompileerd en geregistreerd in de Windows Registry dat het systeem het menu herkent als een ActiveX object. Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 7: Menu compileren en registreren als ActiveX object				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
7.1	Menu is gecompileerd als ActiveX object	Controle op uitvoer compilatie	De map *Projectnaam*\bin\debug bevat het menu in de vorm van een DLL bestand	OK
7.2	DLL bestand is "ondertekend" (signed)	Controle op eigenschappen DLL bestand	DLL bestand bevat "Version Information" en kan in de "Global Assembly Cache" worden geplaatst	OK
7.3	ActiveX object is correct geregistreerd in de Windows Registry	Controle op correcte registratie	Zie paragraaf 5.3.7 van dit document	OK

7.10 Testen bouweenheid 8

In bouweenheid 8 is getracht het menu als ActiveX object in te voegen in een Progress applicatie. Hieronder is een overzicht opgenomen van de resultaten van de tests voor deze bouweenheid.

Testspecificaties bouweenheid 8: Menu als ActiveX object invoegen in Progress				
Case #	**Test**	**Invoer**	**Verwachte uitvoer**	**Uitvoer**
8.1	Menu komt als ActiveX object voor in de lijst met beschikbare ActiveX objecten	Open "OCX" dialoog box in AppBuilder	Menu staat in de lijst beschikbare ActiveX objecten	OK
8.2	Menu kan worden ingevoegd in Progress applicatie	Selecteer menu uit lijst beschikbare ActiveX objecten en plaats menu op Progress Window	Menu is als ActiveX object in de Progress applicatie opgenomen	Test mislukt: Progress applicatie loopt vast of crasht
8.3	Menu wordt correct geïnitialiseerd	Uitvoeren Progress applicatie waarin menu is opgenomen	Menu is geïnitialiseerd	Test niet uitgevoerd
8.4	Functionaliteiten menu werken correct	Navigatie door menu	Navigatie werkt zoals geïmplementeerd in bouweenheid 5	Test niet uitgevoerd

Mislukte tests:

- Testcase 8.2: Invoegen menu in Progress applicatie
  Wanneer in de lijst met beschikbare ActiveX objecten het menu wordt geselecteerd om in te voegen in een Progress applicatie, loopt de Progress ontwikkelomgeving vast of wordt deze afgesloten (zonder een foutmelding). Op dit moment is hiervoor nog geen reden en/of oplossing voor gevonden.

- Testcases 8.3 en 8.4 zijn niet uitgevoerd als gevolg van het falen van testcase 8.2.

7.11 Testen bouweenheid 9

Als gevolg van het falen van testcase 8.2 is bouweenheid 9 niet geïmplementeerd en zijn er dus ook geen tests uitgevoerd voor deze bouweenheid.

7.12 Beoordeling pilot

In deze paragraaf wordt beoordeeld of de pilot (na de eerste iteratie) voldoet aan de "Must Have" eisen die zijn opgesteld in het rapport definitiestudie.
In het onderstaande overzicht is te zien aan welke eisen is voldaan.

ID	Omschrijving	Voldaan
MF1	Een gebruiker heeft de beschikking over een grafisch menu waarmee programma's van XL-Enz kunnen worden opgestart	Nee, mogelijkheid programma's opstarten is nog niet geïmplementeerd
MF2	Een gebruiker kan op gelijke wijze door het menu navigeren als door de Windows Explorer	Ja
MF3	Een gebruiker kan het menu afsluiten	Ja
MN1	Het menu-dataobject kan in Progress een dynamisch opgebouwde menustructuur inlezen uit een bestaande Progress database	Ja
MN2	Het menu-dataobject is te converteren van een Progress dataobject naar een ADO.NET dataobject en vice versa	Ja
MN3	Het menu van XL-Enz heeft een vergelijkbaar uiterlijk als de Windows Explorer: aan de linkerkant van het scherm staan de menu-items in een boomstructuur en aan de rechterkant staat de inhoud van het geselecteerde menu-item weergegeven	Ja
MN4	Het menu van XL-Enz wordt in Progress als een ActiveX object ingevoegd	Nee, invoegen mislukt (zie paragraaf 7.10)
MN5	Het menu-dataobject wordt opgebouwd op de server als een Progress dataobject en vervolgens doorgegeven aan de .NET client, waar het object wordt geconverteerd naar een ADO.NET dataobject	Nee, dataobject wordt lokaal opgebouwd

Op dit moment is het mogelijk om in een .NET omgeving het menu te initialiseren en te navigeren. De functionaliteit voor het opstarten van programma's kan echter alleen in de Progress omgeving worden geïmplementeerd.
Deze functionaliteit kan echter nog niet worden geïmplementeerd vanwege het feit dat het invoegen van het menu in een Progress applicatie mislukt. Als gevolg hiervan kan de pilot niet als succesvol afgerond worden beschouwd.

Het is nog onbekend wat de reden is waardoor de Progress applicatie vastloopt of wordt afgesloten bij het invoegen van het menu. Op dit moment wordt hiernaar onderzoek verricht.
Wanneer de reden bekend is en een oplossing beschikbaar is, kan de ontwikkeling van bouweenheid 8 worden voortgezet door de oplossing te implementeren. Wanneer de geboden oplossing dit vereist, kan de fase pilotontwikkeling in zijn geheel opnieuw worden doorlopen, waarbij de eisen en het ontwerp worden aangepast aan de nieuwe inzichten.
Wanneer er geen oplossing mogelijk blijkt te zijn voor het genoemde probleem, dient met de begeleider en/of opdrachtgever van dit project overleg te worden gepleegd over de verdere ontwikkeling van het menu.

Conclusie

In dit document is een ontwerp gemaakt dat heeft gediend ter ondersteuning bij de implementatie van de tweede pilot uit het project .NET interfacing, waarin de "Must Have" functionaliteiten van het menu van XL-Enz zijn ontwikkeld.
Na implementatie van elke bouweenheid zijn de opgestelde testspecificaties uitgevoerd. De resultaten van deze tests hebben uitgewezen dat de pilot na de eerste iteratie nog niet voldoet aan alle "Must Have" eisen.

Wanneer het menu in een Progress omgeving wordt ingevoegd als een ActiveX object, loopt de Progress ontwikkelomgeving vast of wordt deze afgesloten. Op dit moment is hiervoor nog geen reden en/of oplossing voor gevonden.
Dit probleem heeft tot gevolg dat de functionaliteit voor het opstarten van programma's nog niet is ontwikkeld, omdat deze functionaliteit alleen vanuit een Progress omgeving kan worden geïmplementeerd.

Wanneer de reden bekend is en een oplossing beschikbaar is, kan de ontwikkeling van bouweenheid 8 worden voortgezet door de oplossing te implementeren. Wanneer de geboden oplossing dit vereist, kan de fase pilotontwikkeling in zijn geheel opnieuw worden doorlopen, waarbij de eisen en het ontwerp worden aangepast aan de nieuwe inzichten.
Wanneer er geen oplossing mogelijk blijkt te zijn voor het genoemde probleem, dient met de begeleider en/of opdrachtgever van dit project overleg te worden gepleegd over de verdere ontwikkeling van het menu.

Bijlage A: Plan van aanpak ontwikkeling pilot 2

Inleiding
Deze bijlage bevat het plan van aanpak voor de ontwikkeling van de tweede pilot, waarin de "Must Have" functionaliteiten van het menu van XL-Enz worden geïmplementeerd. Dit plan van aanpak is een detaillering van het eerder gemaakte plan van aanpak voor het gehele project.
Als eerste worden de resultaten beschreven die deze pilot oplevert. Vervolgens wordt een planning gemaakt voor de uitvoering van de pilot, waarbij de planning uit het eerder gemaakte plan van aanpak als uitgangspunt fungeert.
Voor informatie over kwaliteitsborging en de projectorganisatie wordt verwezen naar het plan van aanpak voor het gehele project.

Producten en activiteiten pilotontwikkeling
Hieronder volgt een overzicht van de activiteiten die worden uitgevoerd tijdens de ontwikkeling van de tweede pilot. Per activiteit worden deelactiviteiten beschreven en er wordt aangegeven wat het resultaat is van deze activiteit.

- Specificeren globaal-functionele structuur pilot
  - Bepalen functionele reikwijdte pilot;
  - Detailleren scenario's d.m.v. UML diagram(men);
  - Ontwerpen prototype gebruikersinterface;
  - Resultaat: ontwerp van functionele structuur.
- Specificeren globaal-technische structuur pilot
  - Bepalen fysieke allocatie;
  - Specificeren technisch opslagmodel;
  - Resultaat: ontwerp van technische structuur.
- Opstellen pilotontwikkelplan
  - Definiëren pilotdelen;
  - Definiëren bouweenheden;
  - Synchroniseren bouweenheden;
  - Voorbereiden beoordeling en testen pilotdelen;
  - Resultaat: ontwikkelplan voor pilot.
- Ontwerpen software-bouweenheden
  - Specificeren integratie en testen bouweenheden;
  - Opstellen detailspecificaties;
  - Opstellen testspecificaties;
  - Resultaat: gedetailleerd ontwerp pilotdelen.
- Bouwen software-bouweenheden
  - Coderen pilotdelen;
  - Preparen component-testgegevens;
  - Uitvoeren component-test ;
  - Corrigeren component;
  - Resultaat: implementatie van pilot.

- Opstellen invoeringsprocedure pilot
  - Ontwerpen invoeringsprocedure;
  - Beschrijven verantwoordelijkheden rond invoering;
  - Specificeren tests invoeringsprocedures;
  - Testen invoeringsprocedures;
  - Corrigeren invoeringsprocedures;
  - Resultaat: beschrijving invoeringsprocedure.
- Integreren bouweenheden
  - Integreren bouweenheden;
  - Opzetten integratietestomgeving;
  - Testen integratie bouweenheden;
  - Corrigeren integratie bouweenheden;
  - Resultaat: bouweenheden zijn samengevoegd tot één geheel.
- Beoordelen en testen pilot
  - Opzetten testomgeving;
  - Beoordelen en testen pilot;
  - Analyseren uitkomsten;
  - Corrigeren fouten;
  - Rapporteren over beoordelen en testen.

De volgende activiteiten die IAD voorschrijft worden tijdens dit project niet uitgevoerd:

- Voorbereiden pilotontwerp-workshop
  Reden: De pilot is te klein om een workshop zinvol te laten zijn.
- Specificeren globaal-organisatorische inrichting
  Reden: De pilot heeft geen invloed op organisatorische aspecten.
- Aanpassen externe componenten
  Reden: Er is geen noodzaak tot aanpassing van externe componenten.
- Wijzigen andere informatiesystemen
  Reden: Er is geen noodzaak tot aanpassing van andere informatiesystemen.
- Bouwen conversie-tools
  Reden: Er zijn geen conversie-tools benodigd voor deze pilot.
- Ontwerpen handmatige procedures
  Reden: Er is geen sprake van handmatige procedures bij deze pilot.
- Samenstellen opleidingsmateriaal pilot
  Reden: Er is geen behoefte aan opleidingsmateriaal voor deze pilot.
- Samenstellen handleiding pilot
  Reden: Deze activiteit is gepland tijdens de fase invoering

Iteratiestrategie
Na het uitvoeren van bovenstaande activiteiten zal de pilot worden getoetst of deze voldoet aan de gestelde eisen en normen die in de definitiestudie zijn vastgesteld voor de geïmplementeerde functionaliteiten. Mocht dit niet het geval zijn, dan kan de fase pilotontwikkeling nogmaals worden doorlopen. Deze beslissing wordt genomen in overleg met de begeleider en/of opdrachtgever.
Tijdens een volgende iteratie wordt het ontwerp aangepast of verfijnd om deze aanpassingen vervolgens te implementeren en opnieuw te beoordelen.
Hoewel de fase definitiestudie eenmalig is uitgevoerd, zal het pilotplan eventueel worden bijgesteld aan nieuwe ontwikkelingen.

Faseplanning pilot
Hieronder wordt een gedetailleerde planning van de ontwikkeling van de tweede pilot opgesteld. Hierbij wordt de planning die is opgesteld in het plan van aanpak voor het gehele project als uitgangspunt gebruikt en waar nodig bijgesteld.
De doorlooptijd van deze pilot is gepland op drie weken, van 9 februari 2004 tot en met 27 februari 2004.