HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

Academiejaar 2007 – 2008

Departement Toegepaste Ingenieurswetenschappen
Schoonmeersstraat 52 - 9000 Gent

Digital Signage Management System

Eindwerk voorgedragen tot het behalen van het diploma van
MASTER IN DE INDUSTRIËLE WETENSCHAPPEN : INFORMATICA

Cedric VANDEWOUDE

Promotoren: Leen BROUNS
Jan VAN LEMMENS

HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

Academiejaar 2007 – 2008

Departement Toegepaste Ingenieurswetenschappen
Schoonmeersstraat 52 - 9000 Gent

Digital Signage Management System

Eindwerk voorgedragen tot het behalen van het diploma van
MASTER IN DE INDUSTRIËLE WETENSCHAPPEN : INFORMATICA

Cedric VANDEWOUDE

Promotoren: Leen BROUNS
Jan VAN LEMMENS

Woord vooraf

Op 26 september 2006 begon ik nogal onzeker aan deze opleiding. Nu, twee zware maar heel leerrijke jaren later, kan ik hier trots mijn eindwerk voorstellen. Dit was nooit gelukt zonder de steun van een aantal mensen die ik graag oprecht wil bedanken.

Vooreerst mijn oprechte dank aan mijn interne promotor Leen Brouns voor de uitstekende begeleiding. Ik wil ook mijn dank betuigen aan mijn externe promotors: Jan Van Lemmens en Sien D'Hooghe. Zij hebben me steeds met raad en daad bijgestaan tijdens mijn stage. Ook aan Gunther Govaerts en Martijn Ringoot een woord van dank voor de vele tips en de feedback. Ook alle andere werknemers van SDP die hebben bijgedragen tot een leuke werksfeer verdienen een woord van dank.

Daarnaast wil ik graag alle docenten van de opleiding Industrieel Ingenieur Informatica bedanken voor de kwalitatieve opleiding. Een speciaal woord van dank aan alle ex-schakelaars voor de morele steun en de leuke sfeer. Ten slotte wil ik graag familie en vrienden bedanken voor de steun en hun luisterend oor.

Cedric Vandewoude
Sint-Niklaas, juni 2008

Samenvatting

DIGITAL SIGNAGE MANAGEMENT SYSTEM

Nederlandse versie:

Grote tv-schermen, u komt ze steeds vaker tegen in winkels van allerlei sectoren. Ze worden gebruikt voor een nieuwe vorm van reclame: digital signage. Dit eindwerk handelt dan ook over de analyse, het ontwerp en de realisatie van een zogenaamd digital signage management system. Een beheerder kan vanuit een centrale bepalen wat er verschijnt op de schermen van zijn winkelketen. Hij kan zelf presentaties aanmaken en bepalen op welke schermen ze getoond worden. Het systeem werkt ook probleemloos samen met een kassasysteem. Zo is het geen enkel probleem om het nummer van de volgende klant te tonen naast een presentatie. Tijdens dit project werd ook aandacht besteed aan enkele nieuwe technologieën waaronder Adobe Flex en AMFPHP.

English version:

Big TV screens, you can see them in almost every shop nowadays. They bring a new kind of publicity: digital signage. This paper is about the analysis, design and realization of a digital signage management system. An administrator, located at a central location, can choose the content that is shown on the displays in the shops. He can create presentations and determine on which screens it is displayed. The system also works smoothly with a point of sale system. It’s no problem to show the number of the next client next to a presentation. During this project a couple of new technologies like Adobe Flex and AMFPHP have been researched.

Inhoudsopgave

Lijst met figuren

Lijst met tabellen

Inleiding

Webcare, een dochteronderneming van de SDP-groep, situeert zich in de ontwikkeling van allerlei webprojecten.
Hun applicaties omvatten zowel statische sites als complexere databasegekoppelde B2B-omgevingen.
Verder verzorgt Webcare nog een heel gamma aan intranettoepassingen.

Sinds een aantal jaar ontwikkelt Webcare ook software voor de steeds populairder wordende mediadisplays. Dit zijn grote tv-schermen of displays op weegschalen die gebruikt worden om informatie te verstrekken aan klanten maar ze zijn natuurlijk ook het middel bij uitstek om reclame te maken. Deze nieuwe vorm van reclame wordt ook wel digital signage genoemd.

Digital Signage is een relatief jong medium, maar niet zo nieuw als de meeste mensen denken. Tv-schermen met een videorecorder die in een winkel staan, bestaan immers al jaren. De ontwikkelingen van de afgelopen jaren laten een enorme verbetering zien: schermen worden platter en goedkoper en de komst van het internet biedt kansen voor een nieuwe manier van planning en distributie van de presentaties.

Digital signage is zeer effectief omdat de schermen zich op bezoekers van de winkel richten. Deze bezoekers zijn waarschijnlijk tot aankoop bereid, anders waren ze er niet. De mogelijkheid om de aankoop te doen is ook aanwezig, omdat de schermen en de bezoekers zich al in de zaak bevinden.

Tijdens mijn stage heb ik een digital signage management systeem ontwikkeld. Het systeem bestaat uit drie luiken. Enerzijds is er het programma om schermen en presentaties te beheren en anderzijds een programma dat verantwoordelijk is voor het afspelen van de presentaties. Daartussen zit een laag die ervoor zorgt dat de presentaties vanuit de centrale naar de winkels verspreid worden.

1. Analyse

Webcare ontwikkelde reeds twee versies van Media Display, een digital signage management systeem. Dit systeem was niet erg flexibel en daarom werd er besloten om met een schone lei te beginnen en daarbij nieuwe technologieën te gebruiken.
Voor het echte programmeerwerk kon beginnen, heb ik eerst een grondige analyse gemaakt in twee delen. Het eerste deel, de functieanalyse handelt over de omgeving waarin de mediadisplays terecht zullen komen. In het tweede deel, de definitieanalyse, kan u lezen wat het nieuwe systeem allemaal moet kunnen.

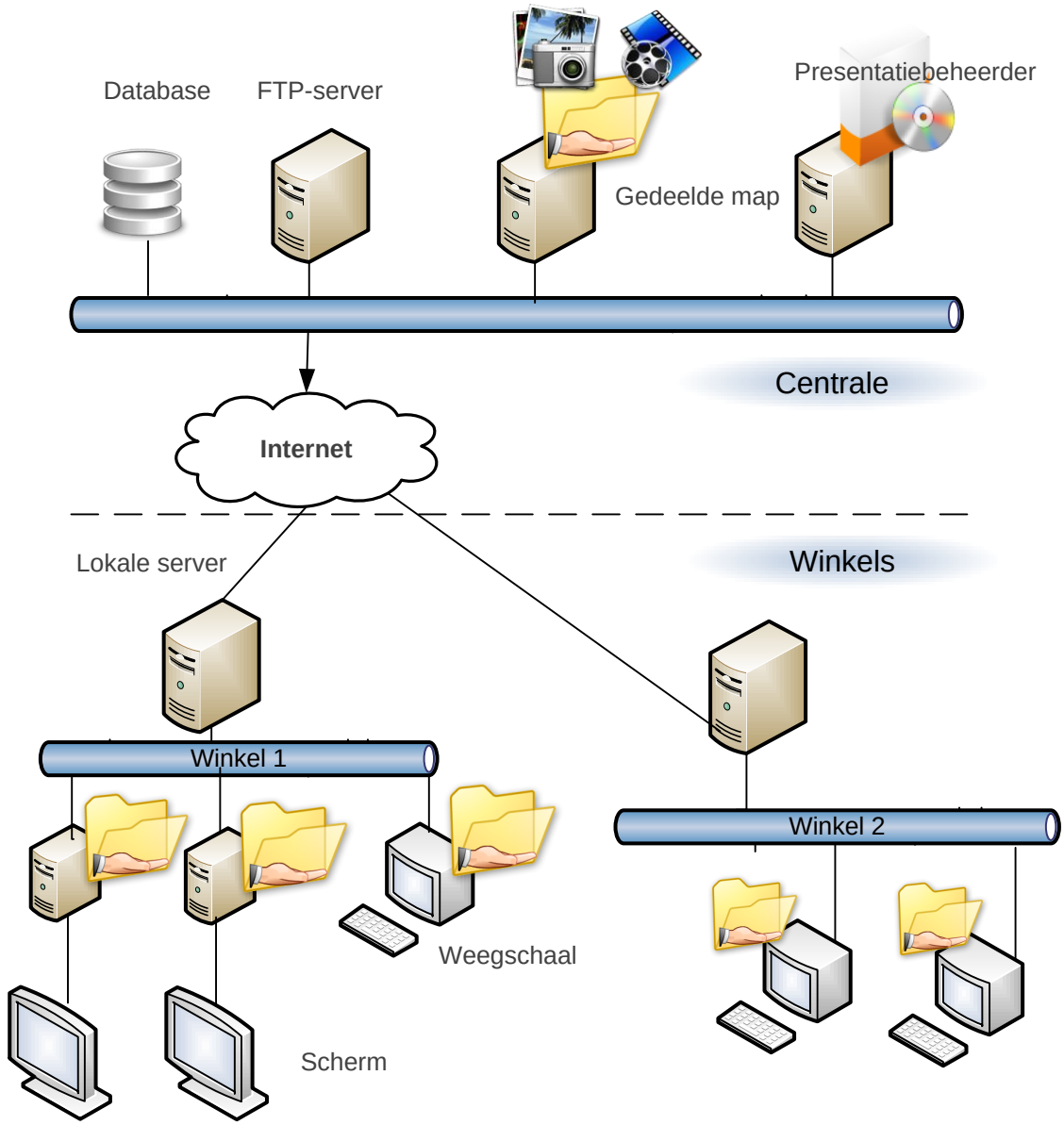
1.1. Functieanalyse

In de functieanalyse die nu volgt, zal ik proberen schetsen in welke (netwerk)omgeving de mediadisplays gebruikt zullen worden.

Er is één centrale server en één of meerdere winkels met elk een lokale server.
Een centrale server vertegenwoordigt meestal de hoofdzetel van een bedrijf en de winkels zijn de verschillende filialen.

De lokale servers in de winkels zijn via het internet verbonden met de centrale server. In elke winkel zijn er verschillende display units. Dit zijn compacte computers die ervoor zorgen dat de presentaties correct worden weergegeven op een of meerdere schermen.

De display units zijn via het lokale netwerk met de lokale server verbonden.

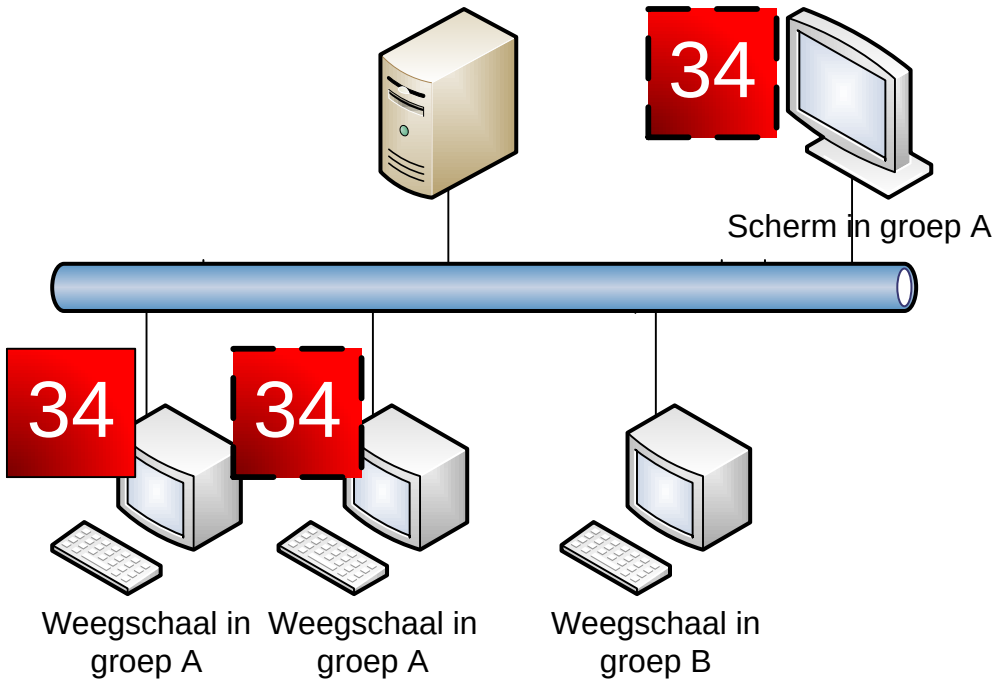


Figuur 1 Netwerkstructuur

Zowel de centrale als de lokale server moeten geïntegreerd kunnen worden in het bestaande systeem van de klant. SDP verkoopt altijd totaalpakketten, dus zowel soft- en hardware. Men weet dus steeds op voorhand welke software en servers reeds geïnstalleerd zijn en hoe deze werken.

Het programma zal drie servers gebruiken. Een databaseserver, een fileserver en een FTP-server. Het is niet noodzakelijk dat deze servers elk op een verschillende computer draaien. Er kan dus gebruik gemaakt worden van de servers die reeds aanwezig zijn.

Er moet ook rekening gehouden worden met het POS-systeem van SDP. Klanten zoals Buurtslagers gebruiken dit systeem op hun weegschalen om automatisch de prijs te berekenen aan de hand van het gewogen product. Bovendien kunnen ze aangeven wat het nummer is van de volgende klant. Een POS-server wordt op de hoogte gebracht telkens het nummer van de volgende klant verandert. Elke weegschaal in de betreffende groep controleert om de zoveel tijd of het nummer aangepast werd. Het programma moet dus in staat zijn om op een of andere manier gegevens uit te wisselen met dit POS-systeem om bijvoorbeeld het nummer van de volgende klant te kunnen weergeven. In onderstaande figuur kan u zien hoe het POS-systeem juist werkt. Op de weegschaal linksonder geeft men aan dat het nummer van de volgende klant 34 is. Alle weegschalen en display units in dezelfde groep worden op de hoogte gebracht en tonen hetzelfde nummer.



Figuur 2 Nummer van de volgende klant doorgeven

De weegschalen hebben twee schermen, één gericht naar het personeel en het andere gericht naar de klanten. De winkelbediende kan op zijn scherm aangeven welk product gewogen wordt. De klanten zien op hun scherm informatie over het product. Afhankelijk van de instellingen van de weegschaal krijgt de klant een aantal modules te zien. Eén van die modules kan het programma zijn dat presentaties toont. Op de display units waaraan een tv-scherm gekoppeld is, draait enkel het programma om de presentaties te tonen. Het moment waarop presentatie X getoond wordt op scherm Y van winkel Z wordt volledig bepaald in de centrale.

1.2. Definitieanalyse

Dit deel van de analyse is net zoals het systeem opgesplitst in drie delen: het eerste deel handelt over de presentatiebeheerder, het volgende deel gaat dieper in op de service die in de winkels draait en in het laatste deel komt het programma dat de presentaties weergeeft op het scherm aan bod.

1.2.1. Presentatiebeheerder

De presentatiebeheerder is het programma dat beheerders zullen gebruiken om presentaties en schermen te beheren. Het beheer kan gebeuren via verschillende computers maar alles moet wel centraal bewaard worden.
Tijdens de installatie van het programma zal een installateur alle schermen moeten kunnen definiëren. Een scherm maakt deel uit van een hiërarchie:

- omgeving
    - regio
        - winkel
            - afdeling
                - categorie
                    - scherm

Een voorbeeldje zal deze structuur duidelijk maken:

Omgeving:	Oost-Vlaanderen
Regio:	Sint-Niklaas
Winkel:	Lamstraat 1
Afdeling:	Slagerij
Categorie:	BBQ
Scherm:	Weegschaal1

Deze indeling is er om een overzicht te kunnen bewaren over alle schermen.
Eenmaal alle schermen gedefinieerd zijn, kan de beheerder bepalen wat er allemaal getoond moet worden op een bepaald scherm. Hij kan hiervoor gebruikmaken van verschillende modules waarvan hij de plaats en de afmetingen kan bepalen.
Elke module kan beschouwd worden als een grafische component die een bepaalde taak vervult. De presentatiemodule is verantwoordelijk voor het afspelen van een aantal presentaties, de logomodule toont het logo van de winkel en de volgende-klant-module toont een animatie telkens het nummer van de volgende klant verandert. De beheerder moet op een eenvoudige manier de plaatsing van de modules kunnen bepalen en ergens kunnen opslaan. Elk schermontwerp kan geëxporteerd worden naar één of meerdere schermen. Het is dus zeker niet verplicht om alle modules te gebruiken. De beheerder kan bijvoorbeeld op kleine schermen enkel de volgende-klant-module plaatsen.

De presentatiemodule is de meest ingewikkelde module. Ze is verantwoordelijk voor het afspelen van de presentaties die de beheerder kan samenstellen. Het moet dus mogelijk zijn om nieuwe presentaties aan te maken en bestaande presentaties te bewerken. Elke presentatie bestaat uit een aantal dia's. Als de beheerder een dia toevoegt, kan hij kiezen uit een aantal soorten: een videodia, een promotiedia en een fotodia.
Alle dia's hebben volgende eigenschappen: een titel, een speelduur en een volgnummer. Bij een videodia kan de beheerder het pad van een filmpje opgeven.

Naast de titel kan de beheerder bij een fotodia ook de omschrijving en het pad van een foto opgeven.
De promotiedia geeft de beheerder de mogelijkheid om maximum 4 artikels in te geven. Het beheer van de artikels zelf moet onafhankelijk zijn van de presentaties. De beheerder kan dus bijvoorbeeld eerst zijn artikels invoeren en ze nadien aan bod laten komen in een aantal presentaties.
Als een presentatie volledig is afgewerkt kan de beheerder ze exporteren naar één of meerdere schermen. Hierbij kan de beheerder ook aangeven wanneer de presentatie juist afgespeeld mag worden. Het moet ook eenvoudig zijn om de presentatie te exporteren naar alle schermen in een bepaalde regio, categorie of afdeling.

Om een goed overzicht te krijgen van alle functionaliteiten van de presentatiebeheerder heb ik een aantal scenario's uitgeschreven.

Basisscenario: Nieuwe presentatie aanmaken	
Aanleiding	
Actor	Presentatiebeheerder (vanaf nu beheerder genoemd)
Frequentie	
Aandachtspunten	
Preconditie	
Scenario beschrijving	1. De beheerder geeft aan dat hij een nieuwe presentatie wil maken 2. De beheerder vult de naam en de speelduur van de presentatie in 3. Het systeem valideert de gegevens en maakt een nieuwe presentatie aan
Postconditie	Er is een nieuwe presentatie aangemaakt.

Basisscenario: Presentatie bewaren	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De presentatie is gewijzigd
Scenario beschrijving	1. De beheerder geeft aan dat hij de presentatie wil bewaren 2. Het systeem bewaart de presentatie
Postconditie	De presentatie is bewaard

Basisscenario: Presentatie exporteren	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De installateur heeft alle schermen gedefinieerd
Scenario beschrijving	1. De beheerder geeft aan dat hij de presentatie wil exporteren 2. De beheerder selecteert de schermen waarop de presentatie getoond moet worden 3. Het systeem zet de presentatie en de mediabestanden klaar zodat de winkels deze kunnen ophalen

Postconditie	De presentatie staat klaar om opgehaald te worden door de winkels

Basisscenario: Presentaties van de centrale ophalen	
Aanleiding	Het systeem op de winkel heeft een tijdstip bereikt waarop het zelf moet controleren of er nieuwe presentaties zijn. (er kan ingesteld worden om de hoeveel minuten het systeem moet controleren op nieuwe versies) Of de centrale geeft aan dat het systeem van de winkel onmiddellijk gewijzigde presentaties moet ophalen
Actor	Systeem
Frequentie	
Aandachtspunten	Deze taak wordt uitgevoerd door het systeem dat in de winkel geïnstalleerd is.
Preconditie	De netwerkconfiguratie voor de verbinding met de centrale is juist ingesteld door de installateur De winkel en de schermen van de winkel zijn gedefinieerd in de centrale
Scenario beschrijving	1. Het systeem controleert om de zoveel tijd of er nieuwe presentaties aanwezig zijn voor de winkel 2. Het systeem maakt verbinding met de centrale en haalt alle nieuwe presentaties op 3. Het systeem plaatst de presentaties op de display units van schermen waarvoor de presentatie bedoeld is 4. Het systeem laat de centrale weten of er problemen waren tijdens de overdracht
Postconditie	De presentaties zijn beschikbaar op de display units

Basisscenario: Dia toevoegen aan een presentatie	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	Er is een bestaande presentatie geopend of een nieuwe presentatie aangemaakt
Scenario beschrijving	1. De beheerder geeft aan dat hij een nieuwe dia wil toevoegen 2. Het systeem geeft een overzicht van de verschillende mogelijke soorten dia's 3. De beheerder geeft aan welk soort dia hij wil toevoegen 4. Het systeem geeft een overzicht de verschillende eigenschappen (bij elke dia sowieso de titel en de speelduur 5. De beheerder vult de eigenschappen in 6. Het systeem valideert de gegevens en voegt de nieuwe dia toe
Postconditie	Er is een dia van de gewenste soort toegevoegd

Subscenario: Een fotodia toevoegen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De beheerder heeft gekozen om een fotodia toe te voegen
Scenario beschrijving	1. De beheerder kiest een foto 2. Het systeem kopieert de foto naar de map met mediabestanden
Postconditie	Er is een nieuwe fotodia toegevoegd aan de presentatie
Alternatieve wegen	2.b. 1. De foto is niet in 16:9 formaat 2. De beheerder selecteert een deel van de foto dat wel in het 16:9 formaat is.

Subscenario: Een videodia toevoegen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De beheerder heeft gekozen om een videodia toe te voegen
Scenario beschrijving	1. De beheerder kiest een video 2. Het systeem maakt een voorbeeldafbeelding van de video en kopieert de video naar de map met mediabestanden
Postconditie	Er is een nieuwe videodia toegevoegd aan de presentatie

Subscenario: Een promotiedia toevoegen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De beheerder heeft gekozen om een promotiedia toe te voegen
Scenario beschrijving	1. De beheerder selecteert de artikels 2. De beheerder vult eventueel een nieuwe prijs en omschrijving in
Postconditie	Er is een nieuwe promotiedia toegevoegd aan de presentatie
Alternatieve wegen	1.b. 1. De beheerder maakt eerst nieuwe artikels aan via Artikelbeheer

Subscenario: Een artikel toevoegen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	Het artikelbeheer staat los van een promotiedia en kan op elk moment gedaan worden.
Preconditie	
Scenario beschrijving	1. De beheerder vult alle eigenschappen van het artikel in 2. Het systeem valideert de gegevens, kopieert de foto van het artikel naar de map met mediabestanden.
Postconditie	Er is een nieuw artikel toegevoegd.
Alternatieve wegen	1.b. 1. De beheerder maakt eerst nieuwe artikels aan via Artikelbeheer 2.b. 1. De foto van het artikel bestaat reeds, gebruiker vragen of de foto overschreven mag worden

Subscenario: Een artikel wijzigen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De beheerder heeft minstens 1 artikel aangemaakt
Scenario beschrijving	1. De beheerder selecteert het artikel dat hij wil wijzigen 2. Het systeem valideert de gegevens, kopieert de foto van het artikel naar de map met mediabestanden.
Postconditie	Het artikel werd gewijzigd.

Basisscenario: Dia's van een presentatie in de juiste volgorde zetten	
Aanleiding	De volgorde van de dia's in de presentatie is niet juist
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	Er is een presentatie geopend Er zijn minstens 2 dia's in de presentatie
Scenario beschrijving	1. Zolang de volgorde van de dia's niet naar wens is 1. De beheerder geeft aan dat hij een bepaalde dia voor of na een andere dia wil plaatsen 2. Het systeem verplaatst de dia
Postconditie	De dia's van een presentatie zitten in de juiste volgorde

Basisscenario: Dia uit een presentatie verwijderen	
Aanleiding	Er zitten overbodige dia's in de presentatie
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	Er is een presentatie geopend Er zit minstens 1 dia in de presentatie
Scenario beschrijving	1. De beheerder geeft aan welke dia hij wil verwijderen 2. Het systeem verwijdert de dia
Postconditie	De overbodige dia's zitten niet meer in de presentatie

Basisscenario: Geldigheidsduur van een presentatie aanpassen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	Er is een presentatie geopend
Scenario beschrijving	1. De beheerder geeft aan dat hij de periode dat de presentatie getoond zal worden, wil aanpassen 2. De beheerder vult 'van' en 'tot' datum in 3. Het systeem valideert de gegevens en past de geldigheidsduur van de presentatie aan
Postconditie	De geldigheidsduur van de presentatie is aangepast

Basisscenario: Scherm toevoegen	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	Een scherm is een verwijzing naar een fysiek beeldscherm. Dat kan een tv-scherm zijn, een scherm op een weegschaal, een projector, … De computer die aan dit scherm gekoppeld is wordt een display unit genoemd. De configuratie van de schermen van de verschillende winkels gebeurt in de centrale.
Preconditie	Er is een 'display unit' in het netwerk opgenomen die van de juiste software voorzien is.
Scenario beschrijving	1. De beheerder geeft aan dat hij een scherm wil toevoegen 2. De beheerder geeft het IP of de naam van de display unit in (fysieke plaats) 3. De beheerder geeft aan in welke omgeving, regio, winkel, afdeling en categorie het scherm zicht bevindt (logische plaats) 4. Het systeem valideert de gegevens en voegt het scherm toe
Postconditie	Het scherm is toegevoegd
Opmerking	De omgevingen, regio's, winkels, afdelingen en categorieën worden op een gelijkaardige manier aangemaakt.

Basisscenario: Schermontwerp maken	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	Het virtuele scherm is een verkleinde weergave van het echte scherm en geeft de beheerder een beeld hoe het schermontwerp er zal uitzien.
Preconditie	
Scenario beschrijving	1. De beheerder geeft aan dat hij een nieuw schermontwerp wil maken. 2. De beheerder geeft de resolutie van het scherm in. 3. Het systeem toont alle beschikbare modules. 4. De beheerder plaatst een module op het virtuele scherm. 5. Het systeem toont alle eigenschappen van de geselecteerde module 6. De beheerder past de eigenschappen aan 7. De beheerder plaatst modules op het virtuele scherm tot hij tevreden is (terug naar stap 4) 8. De beheerder vult een naam in voor het schermontwerp. 9. Het systeem bewaart het schermontwerp.
Postconditie	Er is een nieuw schermontwerp

Basisscenario: Schermontwerp exporteren	
Aanleiding	
Actor	Beheerder
Frequentie	
Aandachtspunten	
Preconditie	De schermen zijn gedefinieerd. Er is een schermontwerp geopend.
Scenario beschrijving	1. De beheerder kiest op welke schermen het schermontwerp van toepassing is. 2. Het systeem past de koppeling tussen het scherm en schermontwerp aan en zet het schermontwerp klaar zodat de winkel het kan ophalen.
Postconditie	Het schermontwerp staat klaar om opgehaald te worden door de winkel.

1.2.2. Winkelservice

Eenmaal de presentaties door de beheerder zijn geëxporteerd, kunnen deze opgehaald worden door de winkels.
Hiertoe draait in elke winkel een service die geconfigureerd wordt door een beperkt aantal parameters. Het is de bedoeling dat de configuratie zoveel mogelijk door de centrale gebeurt en dat elke winkel deze configuratie ophaalt.
De volgende parameters moeten ingesteld worden:

- FTP-gebruikersnaam
- FTP-wachtwoord
  - Als de winkel met deze gegevens aanmeldt, komt hij terecht in zijn persoonlijke map op de centrale.
  - De winkel zal in deze map ook een logboek achterlaten.

Deze parameters moeten ingesteld kunnen worden via een afzonderlijk programma dat

toegankelijk is vanuit de taakbalk.

Als de service in een winkel gestart wordt dan zal de configuratie voor die winkel opgehaald worden. De service opent een FTP-verbinding, meldt zich aan, komt terecht in de persoonlijke map van de winkel en downloadt de configuratie.
In dit configuratiebestand zitten de waarden van de overige parameters die de centrale bepaald heeft voor de service in deze winkel.

- FTP-gebruikersnaam en FTP-wachtwoord:
  - Gegevens die nodig zijn om toegang te krijgen tot map met de mediabestanden.
- Startuur:
  - Vanaf dit uur mag de service controleren of er updates zijn.
- Einduur:
  - Tot en met dit uur mag de service controleren of er updates zijn.
- Tijdsinterval:
  - Het aantal minuten dat de service moet wachten tussen de opeenvolgende controles.
- Random afwijking:
  - Bepaalt hoeveel minuten er afgeweken mag worden van het tijdsinterval om overbelasting te vermijden.

Om de zoveel tijd moet de service controleren of er nieuwe of gewijzigde presentaties aanwezig zijn. Alle nieuwe en gewijzigde presentaties moeten gedownload worden samen met alle mediabestanden die nog niet in de winkel aanwezig zijn.
Vervolgens moeten de presentaties gekopieerd worden naar de display units en weegschalen. Nadien moeten alle overbodige bestanden (van presentaties die nooit meer actief zullen zijn) verwijderd worden.
Per scherm moet de status van deze overdracht bewaard worden zodat men in de centrale kan zien welke schermen problemen opleveren.

1.2.3. Presentatiespeler

Eenmaal de presentaties op de juiste display unit staan, moeten ze nog getoond worden op het scherm. Het programma dat hiervoor verantwoordelijk is, moet een aantal configuratiebestanden inlezen. Eerst moet bepaald worden welke modules waar getoond moeten worden. Vervolgens wordt een overzicht ingelezen dat aangeeft wanneer welke presentatie getoond moet worden. Deze lijst met presentaties wordt overlopen en als de presentatie op dat moment geldig is dan wordt ze afgespeeld. Dat wil zeggen dat het overzicht van dia's wordt ingelezen en dat er per dia een beeld gecreëerd wordt dat te zien is op het scherm.

Basisscenario: Presentatieprogramma initialiseren	
Aanleiding	
Actor	Systeem
Frequentie	
Aandachtspunten	
Preconditie	Er is een configuratiebestand aanwezig dat de naam bevat van de display unit.
Scenario beschrijving	1. Het systeem leest het bestand in met de naam van de display unit. 2. Het systeem leest het bestand in met het schermontwerp. 3. Het systeem voert scenario 'Modules op het scherm plaatsen' uit.
Postconditie	Het systeem is klaar om presentaties af te spelen

Subscenario: Modules op het scherm plaatsen	
Aanleiding	
Actor	Systeem
Frequentie	
Aandachtspunten	Momenteel zijn er 3 modules. Een logomodule, presentatiemodule en een volgende-klant-module. (De taken die ze moeten uitvoeren staan beschreven in de volgende scenario's).
Preconditie	Er is een configuratiebestand aanwezig dat aangeeft waar welke module moet komen op het scherm. (Dit configuratiebestand definieert het schermontwerp)
Scenario beschrijving	1. Het systeem leest het bestand in met het schermontwerp. 2. Het systeem overloopt alle modules in het schermontwerp en plaatst ze (op de opgegeven plaats en met de opgegeven afmetingen) op het scherm.
Postconditie	Alle modules staan op het scherm.

Subscenario: Logomodule tonen	
Aanleiding	
Actor	Systeem
Frequentie	
Aandachtspunten	
Preconditie	Er is een logomodule opgenomen in het schermontwerp.
Scenario beschrijving	1. Het systeem laadt het logo. 2. Het systeem bepaalt de grootte en de positie van het logo.
Postconditie	Het logo wordt weergegeven.

Subscenario: Nummer van de volgende klant tonen	
Aanleiding	Een winkelbediende drukt op een knop om de volgende klant te bedienen.
Actor	Externe interface
Frequentie	Telkens de interface het nummer van de volgende klant ontvangt.
Aandachtspunten	
Preconditie	De initialisatie is zonder problemen verlopen. Er is een volgende-klant-module opgenomen in het schermontwerp.
Scenario beschrijving	1. De interface wordt op de hoogte gebracht dat het nummer van de volgende klant veranderd is. 2. De interface geeft het nummer door aan het systeem. 3. Het systeem toont het nummer.
Postconditie	Het systeem toont het nummer van de volgende klant

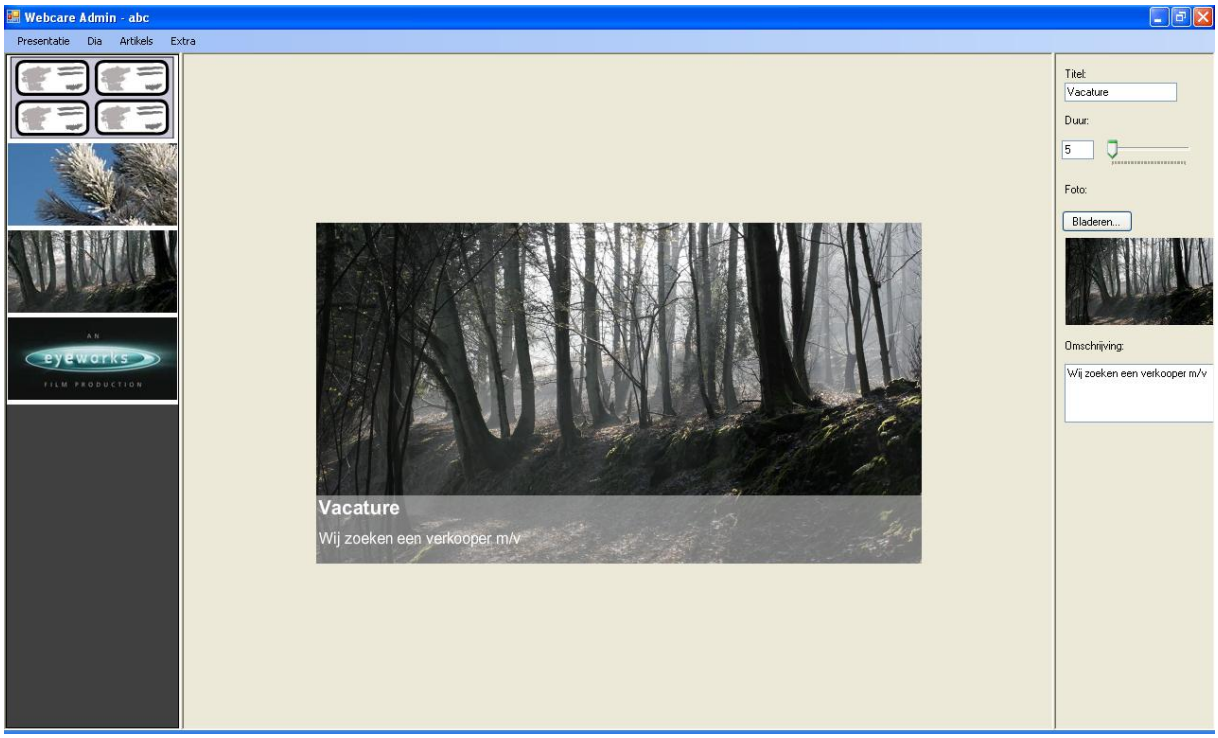
Basisscenario: Geldige presentaties afspelen	
Aanleiding	
Actor	Systeem
Frequentie	
Aandachtspunten	
Preconditie	De initialisatie is zonder problemen verlopen. Er is een presentatiemodule opgenomen in het schermontwerp. Er is een bestand aanwezig dat een overzicht geeft van de verschillende presentaties per scherm.
Scenario beschrijving	1. Het systeem leest het overzichtsbestand. 2. Het systeem overloopt de presentaties die voor zijn scherm bedoeld zijn 3. Het systeem speelt de presentaties af die op dat moment voldoen (datum en tijdstip moeten overeenstemmen) Scenario 'Dia's van een presentatie afspelen wordt uitgevoerd 4. Naar stap 2 als alle presentaties getoond zijn.
Postconditie	Het systeem is klaar om presentaties af te spelen
Alternatieve wegen	2.b. 1. Er zijn nog geen presentaties toegewezen aan het scherm waarop het systeem draait 2. Het systeem toont niks

Subscenario: Dia's van een presentatie afspelen	
Aanleiding	
Actor	Systeem
Frequentie	
Aandachtspunten	
Preconditie	Het presentatiebestand is aanwezig
Scenario beschrijving	1. Het systeem leest het presentatiebestand. 2. Het systeem overloopt alle dia's in het bestand. 3. Het systeem toont elke dia het opgegeven aantal seconden
Postconditie	Alle dia's van een presentatie werden getoond.

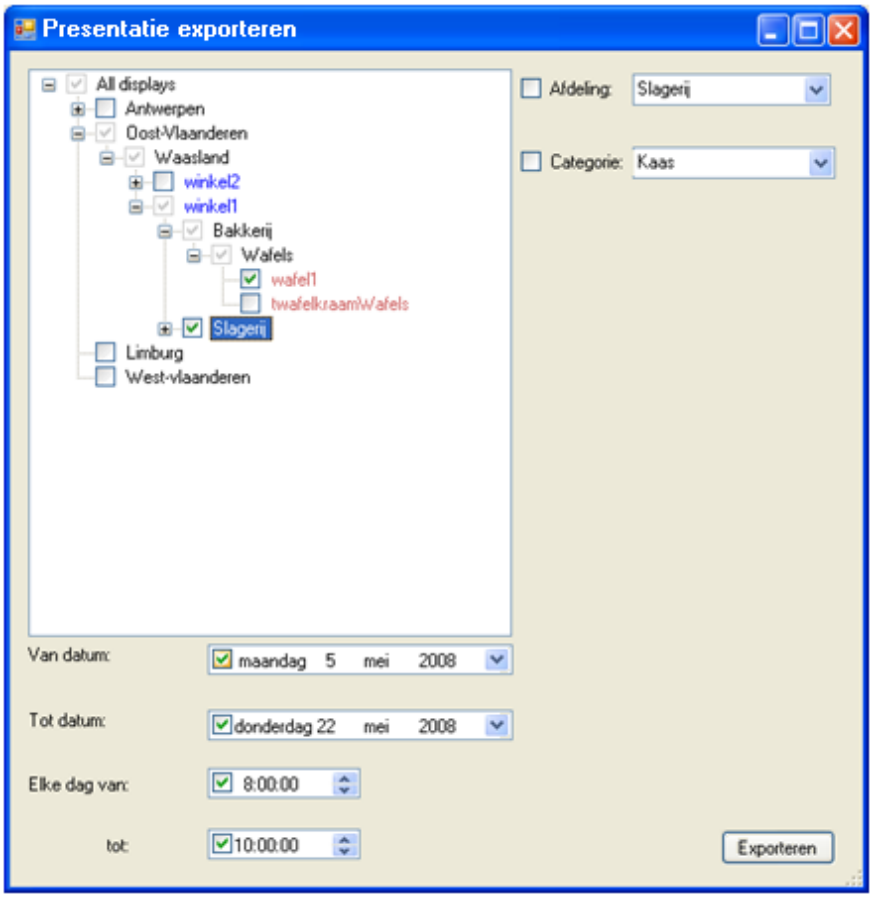
Basisscenario: Een update verwerken	
Aanleiding	
Actor	Externe interface
Frequentie	Telkens de externe interface op de hoogte wordt gebracht van een update.
Aandachtspunten	Met de externe interface bedoel ik het systeem dat het presentatieprogramma als component gebruikt.
Preconditie	De initialisatie is zonder problemen verlopen.
Scenario beschrijving	1. De interface wordt op de hoogte gebracht dat er zodadelijk een update zal beginnen 2. De interface brengt het systeem op de hoogte 3. Het systeem stopt presentatie die op dat moment aan het spelen is en toont een volledig zwart scherm. 4. De interface wordt op de hoogte gebracht dat de update voltooid is 5. Het systeem voert het scenario ' Presentatieprogramma initialiseren' uit
Postconditie	Het systeem toont de modules zoals aangegeven in een eventueel nieuw schermontwerp. Het speelt nieuwe / gewijzigde presentaties af.

2. Vooruitblik

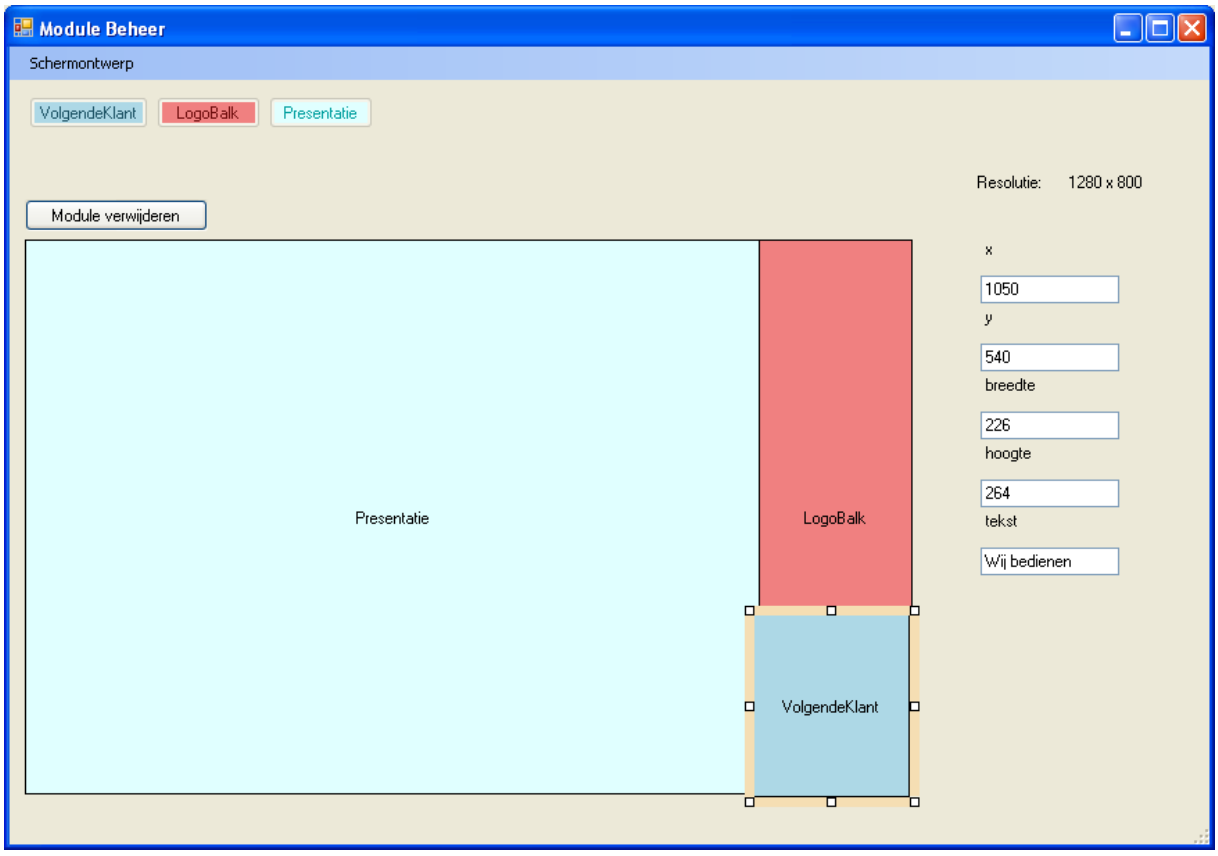
In de vorige hoofdstukken heb ik beschreven hoe het digital signage management systeem eruit zou moeten zien. Een aantal schermafbeeldingen zeggen natuurlijk veel meer dan een doorlopende tekst. Bovendien zal u na deze vooruitblik de realisatiefase van het systeem beter kunnen volgen.
Op de bijgeleverde cd-rom kan u ook een aantal demonstratiefilmpjes terugvinden.



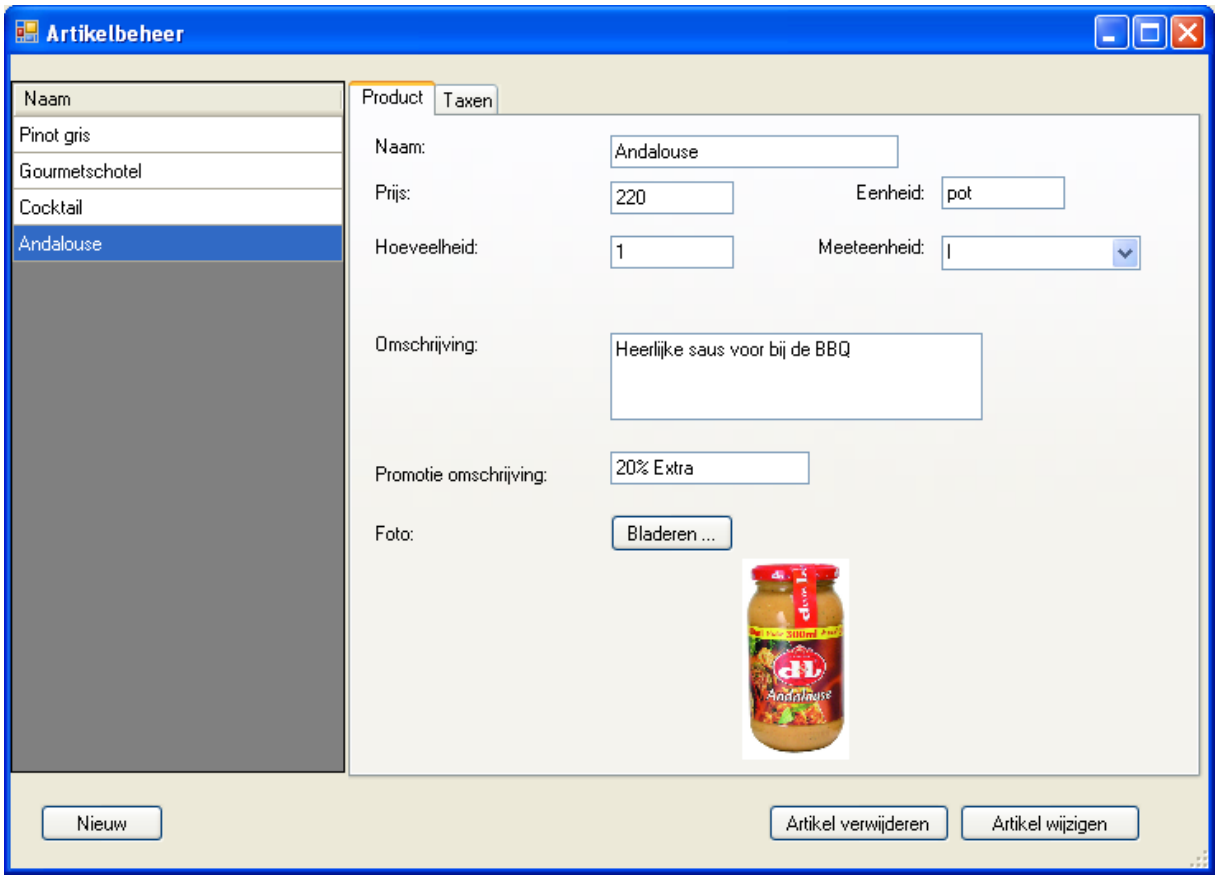
Figuur 3 Presentatie aanmaken



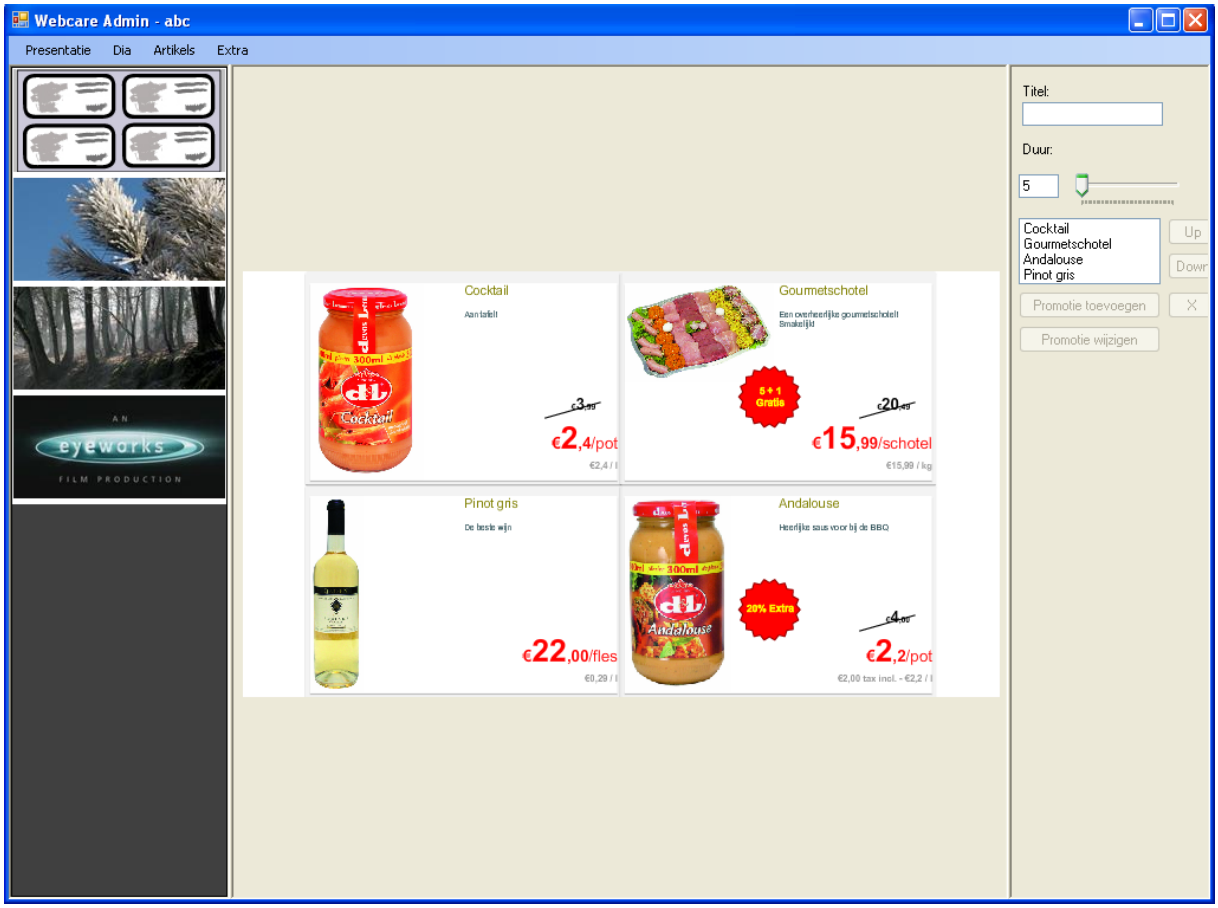
Figuur 4 Presentatie exporteren



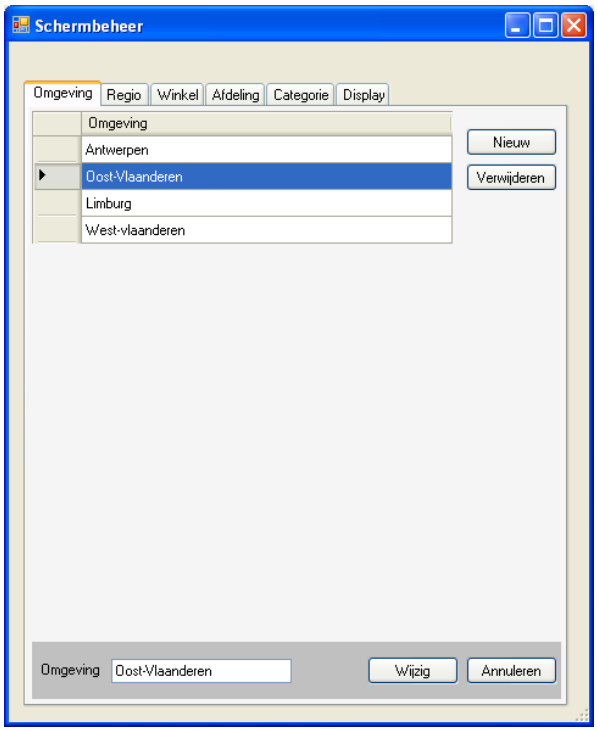
Figuur 5 Schermontwerp aanmaken



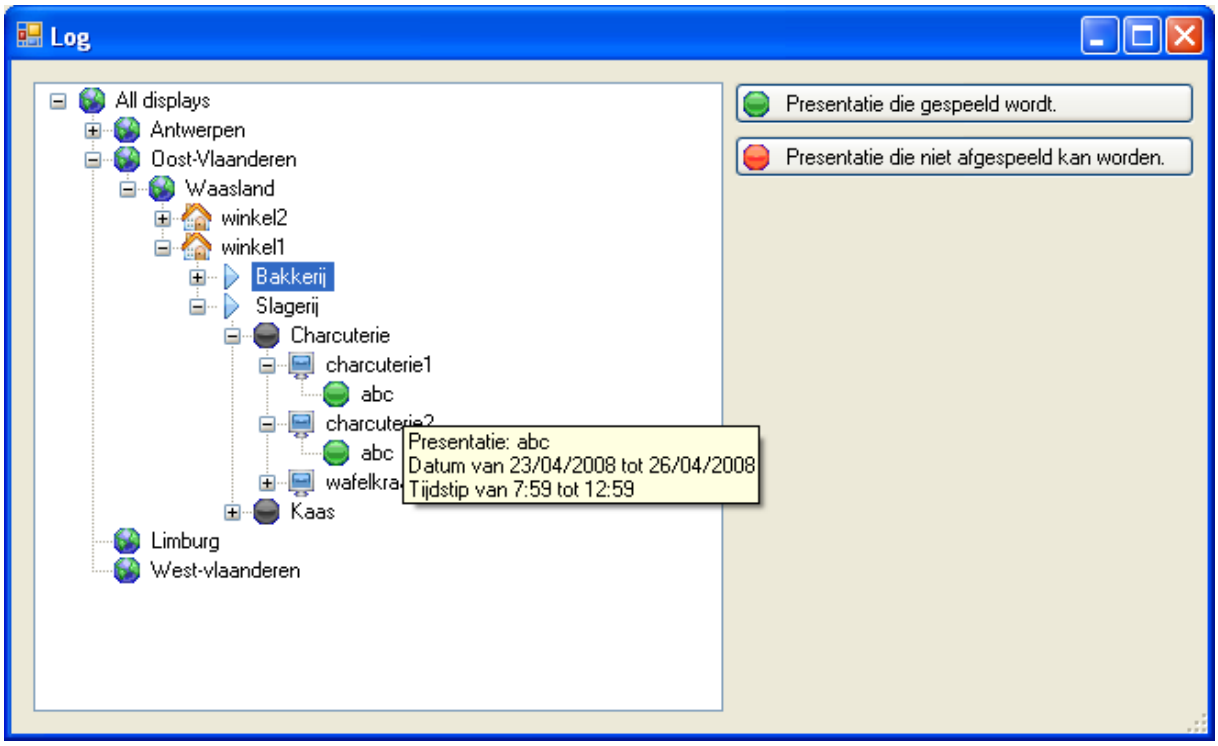
Figuur 6 Artikelbeheer



Figuur 7 Promotiedia



Figuur 8 Schermbeheer

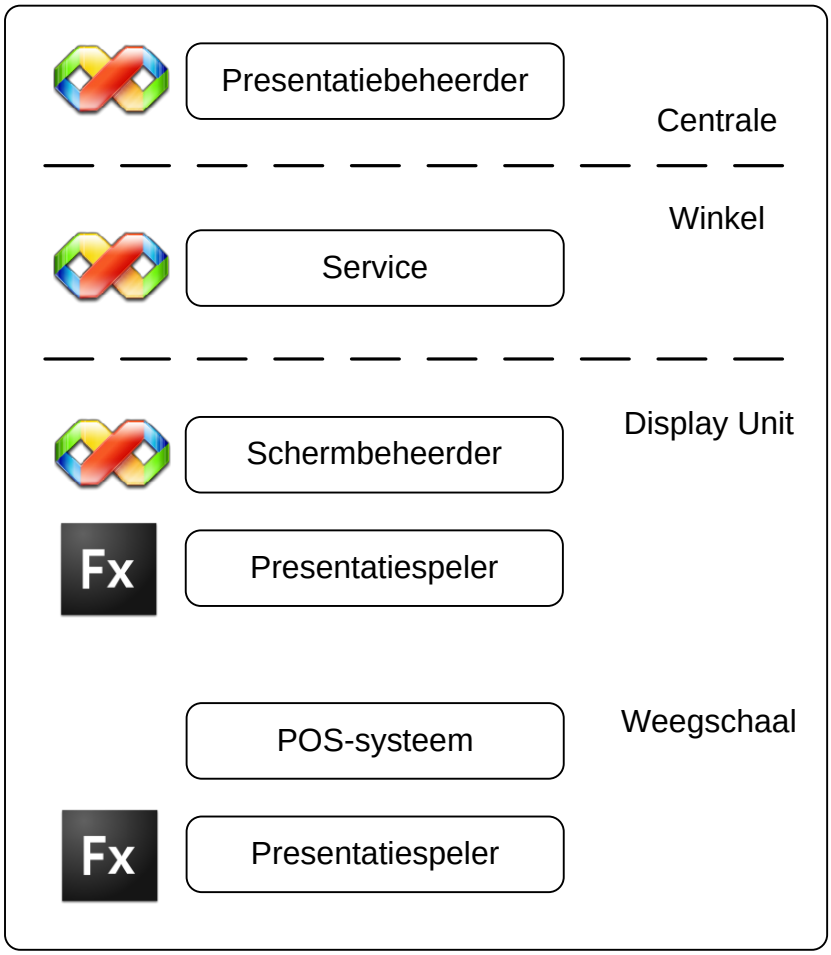


Figuur 9 Overzicht: welke presentaties worden correct getoond en welke niet

Figuur 10 De presentatie die de klanten te zien krijgen op een tv-scherm

3. Ontwerp en realisatie

Na het lezen van vorige hoofdstukken weet u wat het digital signage management systeem juist moet kunnen en hoe het eruit ziet. In dit hoofdstuk zal ik het ontwerp en de realisatie van het systeem bespreken. Het volgende schema geeft u alvast een beeld van de verschillende softwareonderdelen:



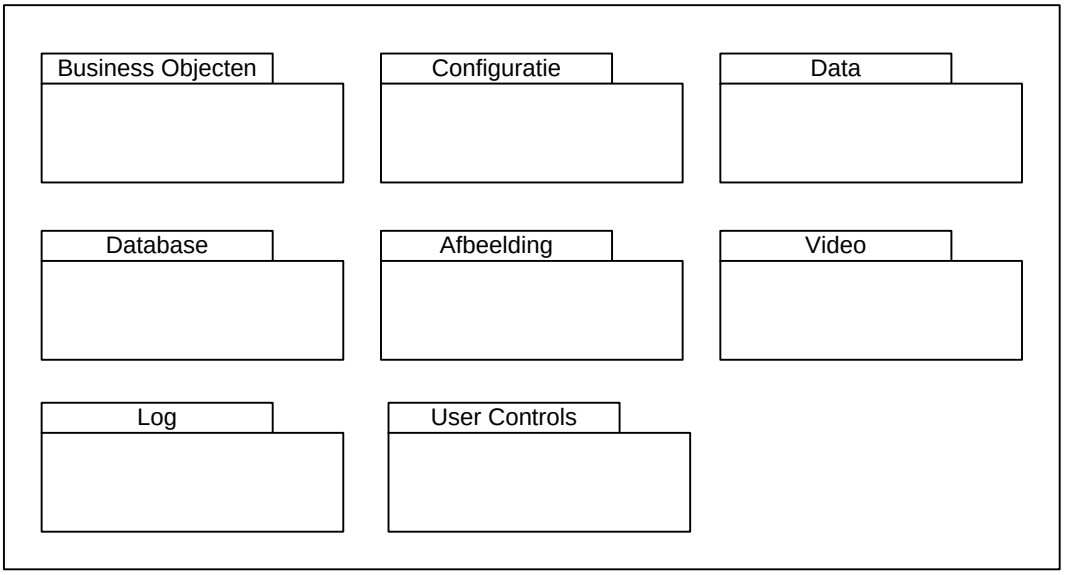
Figuur 11 De verschillende softwareonderdelen

3.1. De presentatiebeheerder

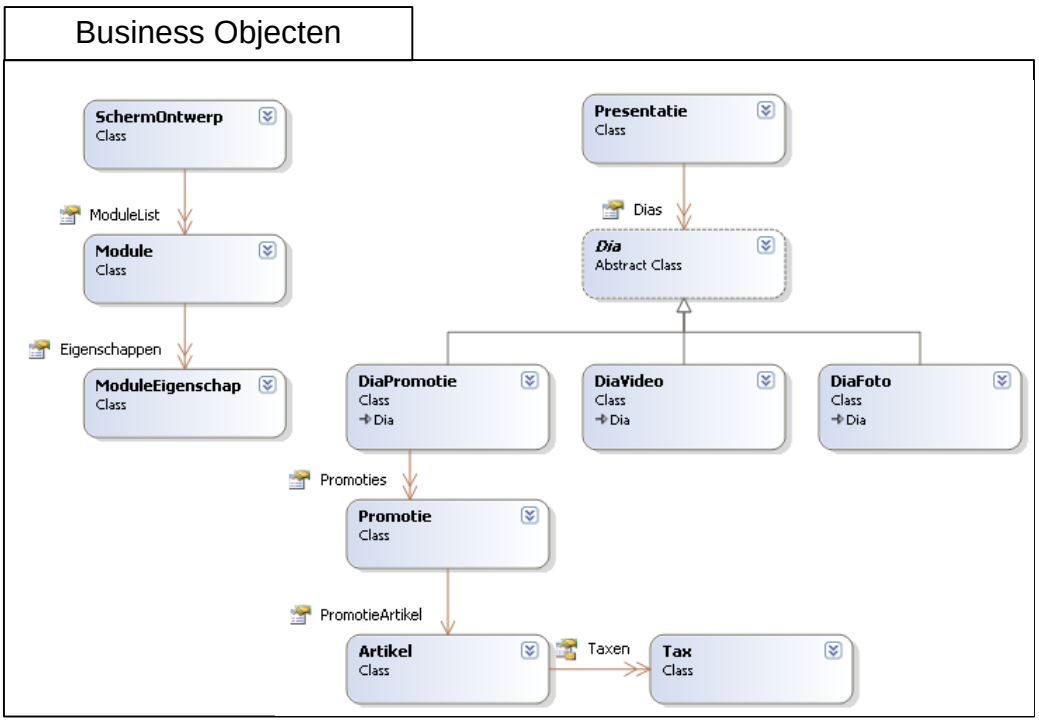
Zoals ik reeds beschreven heb in de analyse is er één centrale waar beheerders presentaties kunnen aanmaken en bewerken. In het netwerk van de centrale is een databaseserver, FTP-server en gedeelde map aanwezig.
De database bewaart de artikels, de hiërarchische indeling van de verschillende schermen en allerhande parameters.
Presentaties en de mediabestanden worden bewaard in een gedeelde map. Zo kunnen verschillende beheerders presentaties aanmaken en wordt alles centraal bewaard. Omdat alles centraal bewaard wordt is er ook geen redundantie van grote mediabestanden die in verschillende presentaties gebruikt kunnen worden.

Eenmaal een presentatie is afgewerkt, kan deze geëxporteerd worden. Dit wil zeggen dat de beheerder kan kiezen op welke schermen de presentatie getoond moet worden. De beheerder krijgt de relatie te zien tussen de verschillende omgevingen, regio's, winkels, categorieën, afdelingen en schermen. Hij kan ook kiezen om bijvoorbeeld alle schermen in een bepaalde regio of in een bepaalde afdeling aan te duiden.

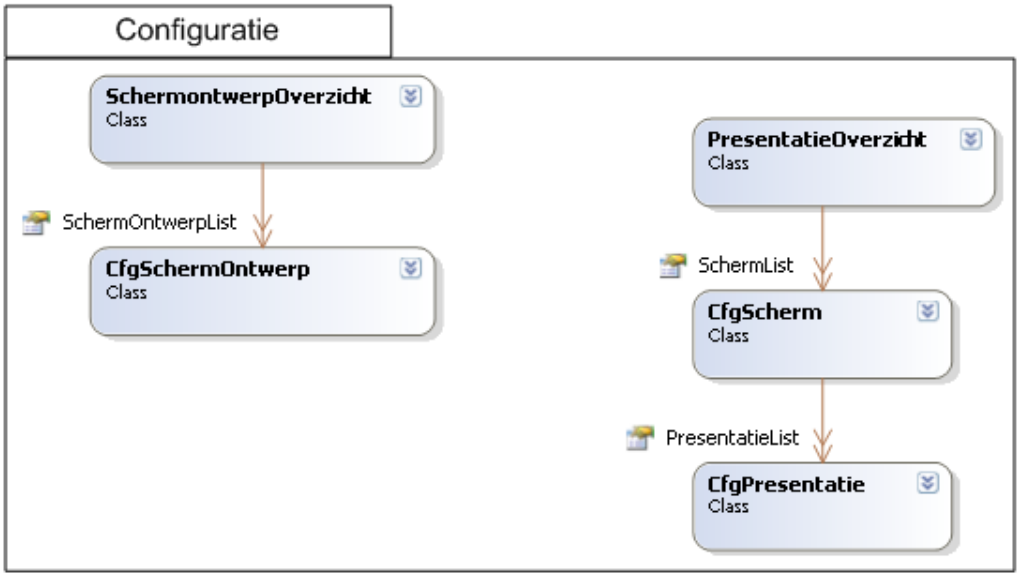
In volgende paragrafen zal ik het ontwerp en de functionaliteit van de presentatiebeheerder bespreken. Om te zorgen dat u niet verdwaalt in de wirwar van klassen geef ik eerst een aantal algemene klassendiagrammen:



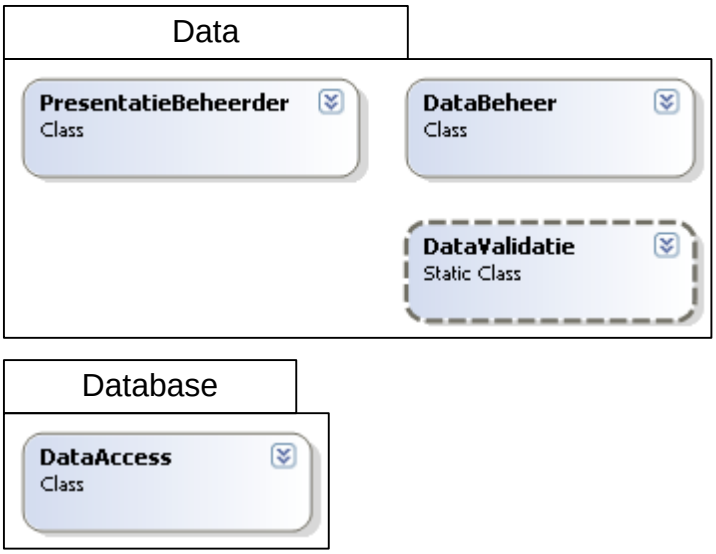
Figuur 12 De verschillende pakketten in de presentatiebeheerder



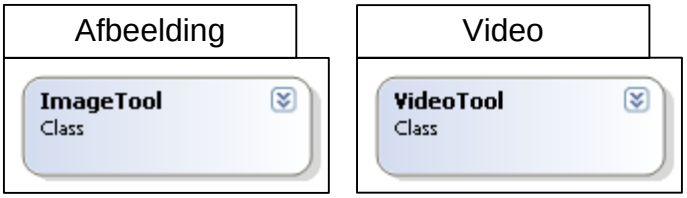
Figuur 13 Klassendiagram: Business Objecten



Figuur 14 Klassendiagram: Configuratie

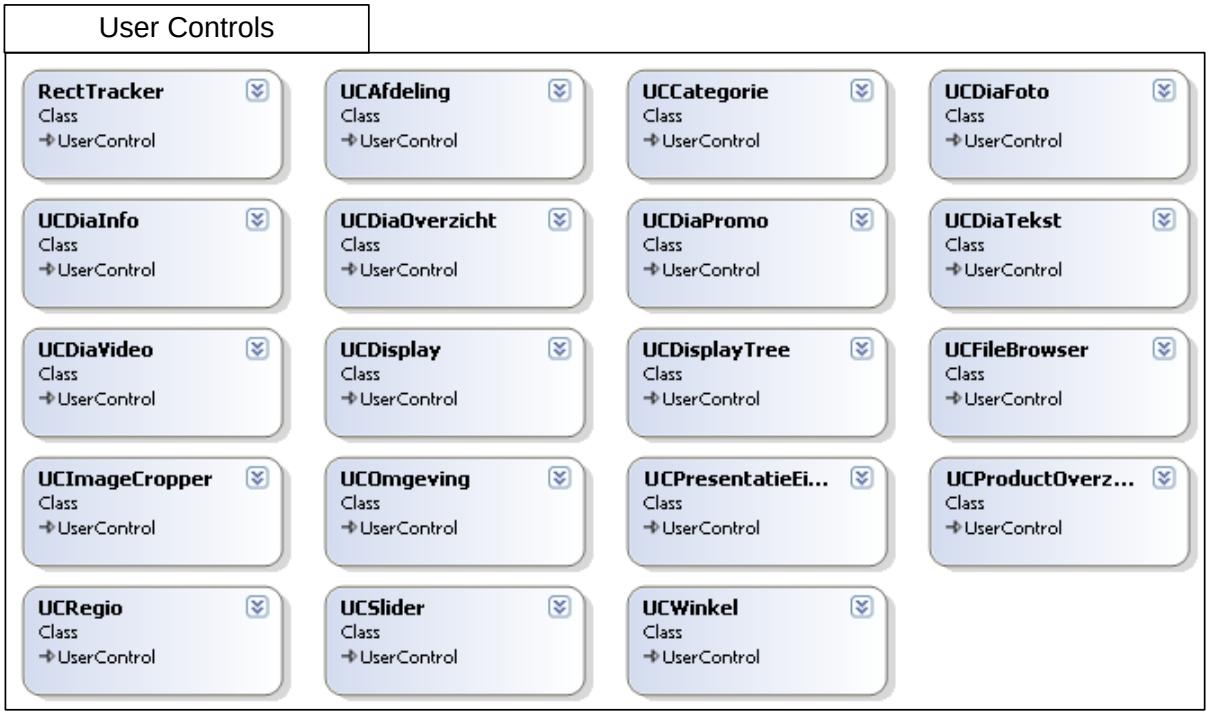


Figuur 15 Klassendiagram: Data en Database



Figuur 16 Klassendiagram: Afbeelding en Video

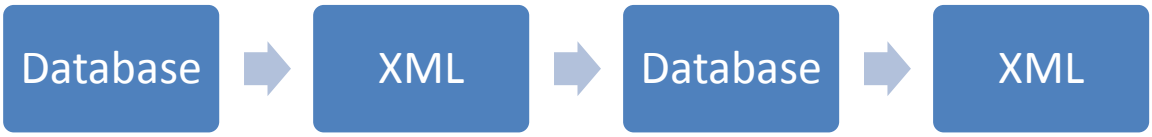
Figuur 17 Klassendiagram: Log



Figuur 18 Klassendiagram: User Controls

3.1.1. Presentaties openen en bewaren

In de vorige versie van het systeem werden presentaties bewaard in een database. Als de beheerder besliste om de presentaties te exporteren dan werd de inhoud van de database omgezet naar XML die dan op zijn beurt weer werd ingelezen door de databaseserver in de winkel. In de winkels werden de presentaties opgehaald uit de database en omgezet naar XML-bestanden die op de display units terecht kwamen.

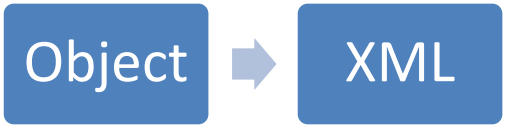


Figuur 19: Omzetting van database naar XML wordt onnodig herhaald

De omzetting van XML-bestanden naar databaserecords gebeurde dus eigenlijk 2 keer en er was heel veel code voor nodig. Deze manier van werken heeft ook als nadeel dat een presentatie niet echt 'draagbaar' is. Bij de ontwikkeling van het nieuwe systeem heb ik ervoor gekozen om een presentatie object te serialiseren[1] naar XML.
Nu wordt enkel de naam van de presentatie nog bewaard in de database. Dit heeft een aantal voordelen. Omdat men in C# objectgeoriënteerd kan programmeren en er ingebouwde methodes gebruikt kunnen worden om objecten om te zetten naar XML, is

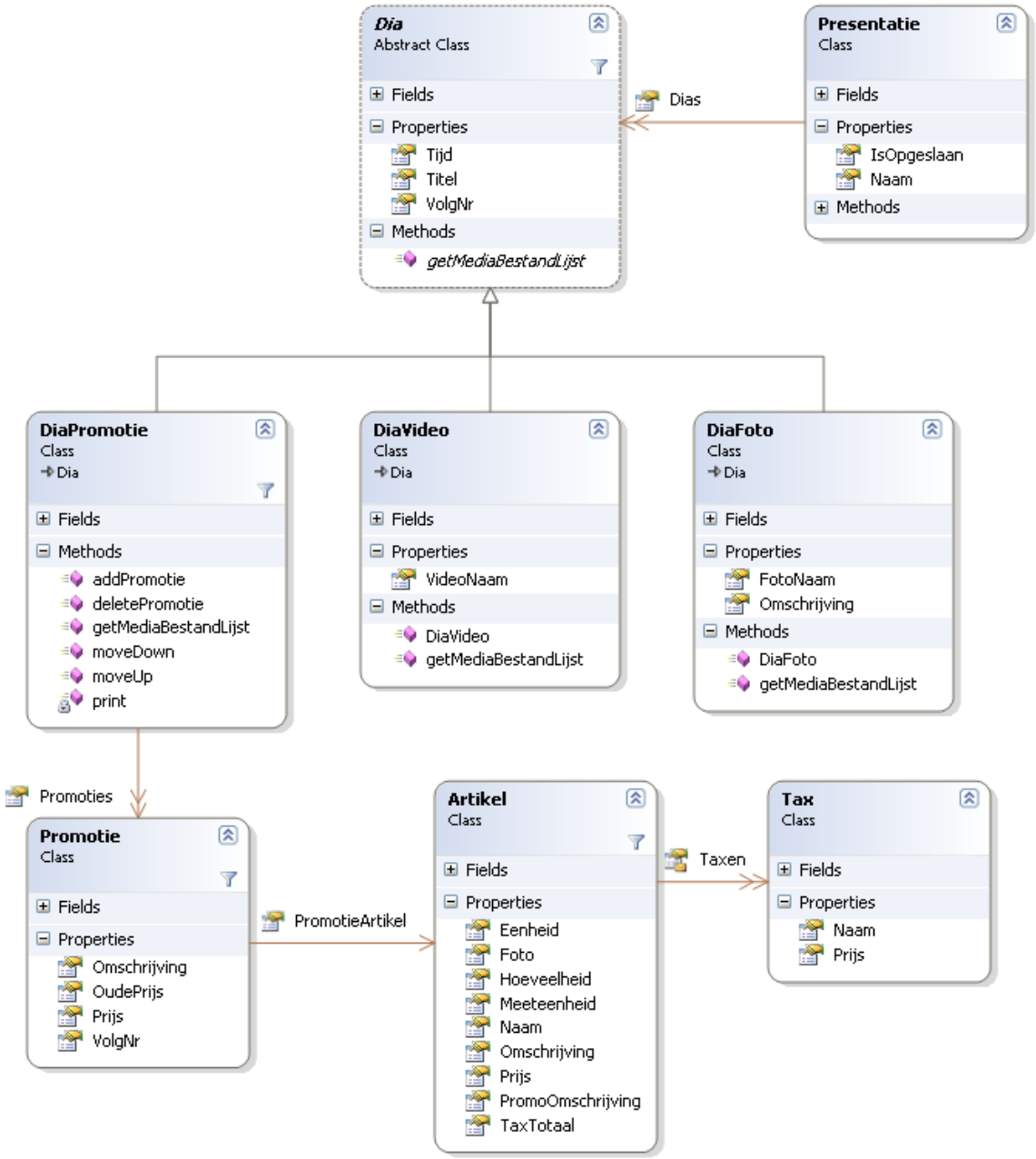
[1] U kan meer algemene informatie over dit onderwerp vinden in hoofdstuk 5.2.1.

deze manier van werken veel eenvoudiger en bovendien is de ingewikkelde databasestructuur verleden tijd. Een presentatie wordt nu bewaard als een bestand en kan ook in dit formaat doorgestuurd worden naar de winkels.



Figuur 20 XML-serialisatie

Een presentatie is opgebouwd uit verschillende dia's met een aantal eigenschappen. Er zijn drie soorten dia's: videodia, promotiedia en fotodia. Elke dia heeft een titel, een speelduur en een volgnummer.
Ik heb een abstracte klasse Dia geschreven die de algemene dia-eigenschappen bevat. De klassen DiaVideo, DiaPromotie en DiaFoto erven van deze abstracte klassen en bevatten de eigenschappen die specifiek zijn voor hun soort. U kan deze structuur ook terugvinden in volgend klassendiagram en bijhorende stukken code.



Figuur 21 Klassendiagram: Een presentatie is opgebouwd uit verschillende dia's

```
[XmlRoot("Presentatie")]
public class Presentatie
{
    …
    #region Properties
    [XmlIgnore]
    public bool IsOpgeslaan
    …

    [XmlAttribute()]
    public string Naam
    …

    [XmlArray("Dias")]
    [XmlArrayItem(typeof(DiaFoto))]
    [XmlArrayItem(typeof(DiaVideo))]
    [XmlArrayItem(typeof(DiaPromotie))]
    public List<Dia> Dias
    …
    #endregion
    …
}
```

Een presentatie heeft een naam en bestaat uit een aantal dia's.

```
public abstract class Dia
{
    …
    [XmlAttribute()]
    public String Titel

    [XmlAttribute()]
    public int Tijd

    [XmlAttribute()]
    public int VolgNr
    …
}
```

Elke dia heeft een titel, speeltijd en een volgnummer.

```
public class DiaFoto : Dia
{
    …
    public String FotoNaam

    public String Omschrijving
    …
}
```

DiaFoto erft van Dia en heeft een extra eigenschap omschrijving. DiaVideo en DiaPromotie definiëren op gelijkaardige manier specifieke eigenschappen.

Op elk moment kan er maar één presentatie geopend zijn. Dit is de actieve presentatie

en wordt bijgehouden (in het geheugen) door de klasse PresentatieBeheerder. Van deze singleton klasse kan er slechts één instantie aangemaakt worden die overal in het programma beschikbaar is.

```
class PresentatieBeheerder
{
    private static PresentatieBeheerder instance;
    private Presentatie actievePresentatie;
    …

    public static PresentatieBeheerder getInstance()
    {
        if (instance == null)
            instance = new PresentatieBeheerder();
        return instance;
    }

    #region properties
    public Presentatie ActievePresentatie
    {
        get { return actievePresentatie; }
        set {
            actievePresentatie = value;
            volgNr = 1;
        }
    }

    public Dia ActieveDia
    {
        get {
            if (actievePresentatie.Dias.Count < volgNr)
            {
                return null;
            }
            else
            {
                return actievePresentatie.Dias[volgNr - 1];
            }
        }

    }
    #endregion
    …
}
```

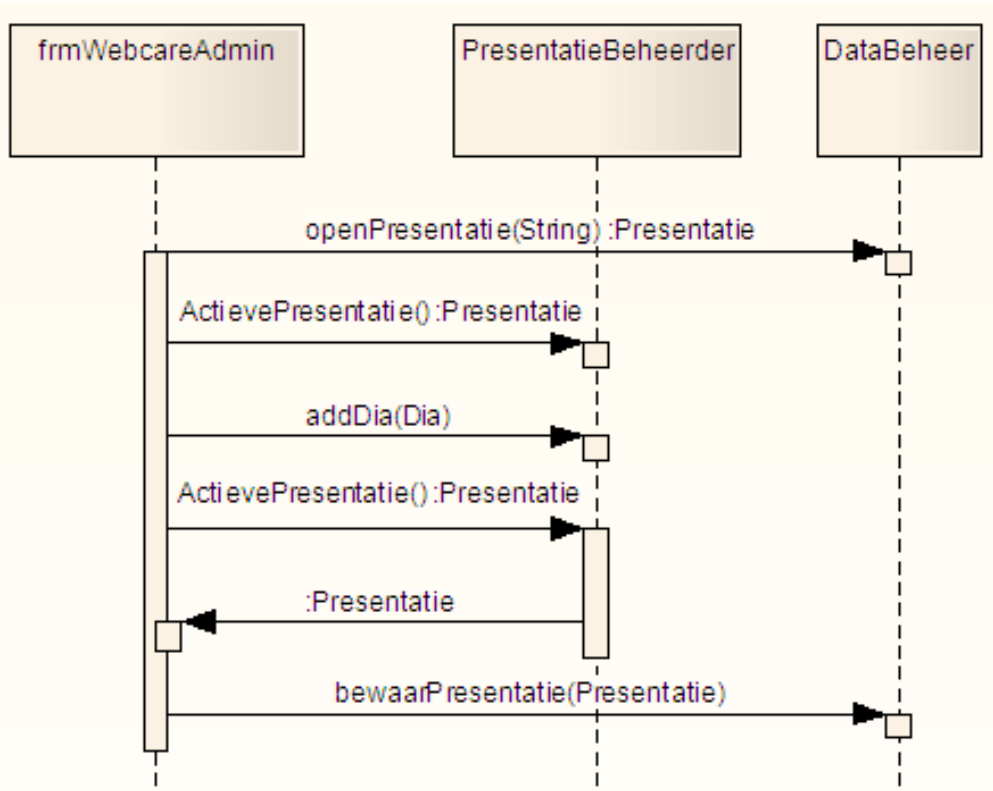
De actieve presentatie en de actieve dia zijn dus op elk moment beschikbaar in het programma zonder dat er telkens een nieuwe instantie aangemaakt moet worden van de PresentatieBeheerder.

Als de beheerder de actieve presentatie wil bewaren, wordt het presentatie object (en automatisch ook alle onderliggende dia's) geserialiseerd naar een XML-bestand.
Alle methodes die data ophalen of schrijven naar de centrale gedeelde map zijn verzameld in de klasse DataBeheer waarvan eveneens maar één instantie kan aangemaakt worden.
De methode bewaarPresentatie van de klasse DataBeheer zet een Presentatieobject om naar een XML-bestand. Het presentatiebeheer is totaal onafhankelijk van het databeheer. De presentatiebeheerder hoeft ook niet te weten hoe de actieve presentatie

geserialiseerd moet worden, al deze richtlijnen zitten in de Presentatieklasse en de diaklassen. Als we bijvoorbeeld de naam van een presentatie niet als een XML-attribuut maar als een XML-element willen serialiseren dan hoeven we enkel in de klasse Presentatie de richtlijn `[XmlAttribute()]` boven de eigenschap Naam weg te halen.

```
DataBeheer.getInstance().bewaarPresentatie(PresentatieBeheerder.getIn
stance().ActievePresentatie);
```

De code en het sequentiediagram verduidelijken hoe dit juist werkt.



Figuur 22 Sequentiediagram: presentatie openen en bewaren

```
public void bewaarPresentatie(Presentatie presentatie)
{
    XmlSerializer s = new XmlSerializer(typeof(Presentatie));
    String pad = DataBeheer.getInstance().getPresentationPath();
    TextWriter w = new StreamWriter(pad + @"\" + presentatie.Naam +
".xml");
    s.Serialize(w, presentatie);
    w.Close();
}
```

Het presentatieobject wordt geserialiseerd naar een XML-bestand. De bestandsnaam is de naam van de presentatie gevolgd door de extensie xml. Als de beheerder de naam van een nieuwe presentatie ingeeft, wordt er gecontroleerd of er geen speciale tekens aanwezig zijn.

De databeheerder voorziet natuurlijk ook een methode om een presentatie te openen. Nu wordt een XML-bestand omgezet naar een presentatieobject. Dit proces wordt ook wel deserialisatie genoemd.

```
public Presentatie openPresentatie(String pad)
{
    XmlSerializer s = new XmlSerializer(typeof(Presentatie));
    TextReader r = new StreamReader(pad);
    Presentatie presentatie = (Presentatie)s.Deserialize(r);
    r.Close();

    return presentatie;

}
```

3.1.2. Presentaties bewerken

Als de beheerder een nieuwe presentatie heeft aangemaakt of een bestaande presentatie heeft geopend, kan hij dia's toevoegen en wijzigen. Er wordt een overzicht getoond van de verschillende soorten dia's. De beheerder kiest een soort en alle eigenschappen ervan worden getoond. Per eigenschap heb ik een User Control ontwikkeld die dynamisch wordt toegevoegd aan de lijst van eigenschappen. Een User Control biedt het voordeel dat andere klassen zich niet bewust moeten zijn van de componenten die erop staan. Er zijn enkel een aantal eigenschappen beschikbaar.

Een voorbeeldje zal deze techniek wellicht verduidelijken.
Elke dia heeft een speelduur die de gebruiker kan aanpassen door een getal in te vullen in een tekstvak of door de positie van een schuifbalk aan te passen. De User Control zorgt ervoor dat de waarde van het tekstvak aangepast wordt als de positie van de schuifbalk aangepast wordt en omgekeerd. Voor andere klassen maakt het niet uit hoe de gebruiker de speelduur kan ingeven zolang ze maar de speelduur kunnen achterhalen. Via de publieke eigenschap Waarde van de User Control kunnen andere klasse de speelduur opvragen.

```
public int Waarde
{
  get { return trackBar1.Value; }
}
```

Naast de speelduur heeft elke dia ook een titel die aangepast kan worden. Dit zijn algemene eigenschappen. De verschillende soorten dia's (videodia: 3.1.2.1, fotodia: 3.1.2.2, promotiedia: 3.1.2.3) hebben ook nog eigenschappen die specifiek zijn voor hun soort.

3.1.2.1. Videodia

Op deze dia wordt het gewenste filmpje afgespeeld. De beheerder kan een filmpje kiezen in het FLV[2]-formaat. Vervolgens wordt er een schermafbeelding gemaakt zodat de beheerder het filmpje later ook nog makkelijk kan herkennen.

Deze videobewerking is totaal onafhankelijk van een dia of presentatie en hoort dus thuis in een aparte klasse. Ik heb dus een klasse VideoTool geschreven met een statische methode makeThumbnail. Deze methode verwacht 4 parameters: het pad, de breedte en

[2] Flash Video

hoogte van het filmpje en het tijdstip waarop een schermafbeelding gemaakt moet worden. Het programma FFmpeg [3] zorgt voor de schermafbeelding en kan uitgevoerd worden als een proces vanuit C#:

```
      public static String makeThumbnail(String flvPath, int breedte, int
hoogte, int sec){

            Process ffmpeg = new Process(); // maak een proces aan
            …
            //argumenten die FFmpeg nodig heeft
            String quotes = "\"";
            String arg = "-i " + quotes + flvPath + quotes + " -vcodec
             mjpeg -vframes 1 -an -f rawvideo -ss " + sec.ToString() + " -s
            " + breedte.ToString() + "x" + hoogte.ToString() + " " + quotes
            + thumbPad + quotes ;
            ffmpeg.StartInfo.Arguments = arg;

            //het scherm verbergen voor de gebruiker
            ffmpeg.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;

            //het programma dat we willen uitvoeren
            ffmpeg.StartInfo.FileName = "ffmpeg.exe";

            ffmpeg.Start(); // maak thumbnail
            ffmpeg.WaitForExit();// wachten tot de operatie voltooid is

            return thumbPad;
      }
```

Eenmaal de schermafbeelding gemaakt is, wordt ze gekopieerd naar de centrale gedeelde map en krijgt de beheerder ze te zien in het paneel met de dia-eigenschappen.

3.1.2.2. Fotodia

Een fotodia toont een foto en eventueel een omschrijving en een titel.
De beheerder vult de titel en omschrijving in en kiest een foto. Deze foto moest zo groot mogelijk in het 16:9 formaat, zonder te vervormen, getoond worden. De foto vult dus steeds heel het scherm.
Als de foto reeds in het 16:9 formaat staat moet hij enkel uitgerekt worden (met behoud van de verhoudingen) zodat hij zeker past op het grootste scherm. Het is natuurlijk niet vanzelfsprekend dat de foto die de beheerder kiest al in het 16:9 formaat staat. Als het formaat van de foto niet correct is dan kan de beheerder een rechthoekig gebied (formaat 16:9) van de foto selecteren. Dit gebied wordt aangegeven door een doorzichtige rechthoek met een gekleurde rand. Van zodra de muiscursor van positie verandert, zal ook de rechthoek van positie veranderen. De beheerder hoeft dus geen grafisch expert te zijn om een foto in het 16:9 formaat te bekomen.

[3] FFmpeg is an audio/video conversion tool. It includes libavcodec, the leading open source codec library. An experimental streaming server for live broadcasts is also included.

Figuur 23 Gebied (formaat 16:9) uit een foto selecteren (1)

Figuur 24 Gebied (formaat 16:9) uit een foto selecteren (2)

Alle methodes die bewerkingen uitvoeren op afbeeldingen heb ik gegroepeerd in de klasse ImageTool. Deze klasse bevat 3 statische methoden: cropAndZoomImage, scaleToFixedSize en needToScale.
De methode cropAndZoomImage snijdt een rechthoekig gebied uit een afbeelding en rekt het uit. Ze heeft 6 parameters: de oorspronkelijke afbeelding, het startpunt: de linkerbovenhoek van het rechthoekig gebied, de hoogte en de breedte van het rechthoekig gebied en de hoogte en de breedte van de resulterende afbeelding.
De nieuwe afbeelding wordt gemaakt met behulp van GDI+, de .NET-versie van GDI[4].

[4] GDI staat voor Graphic Device Interface. Het is een set van C++-klassen die data uitwisselen tussen de applicatielaag en de hardwarelaag. De implementatie in .NET is beter gekend onder de naam GDI+.

```
public static Image cropAndZoomImage(Image img, Point pStart, int
     breedte, int hoogte, int doelBreedte, int doelHoogte) {


    Bitmap bmpAfbeelding = new Bitmap(doelBreedte, doelHoogte,
          img.PixelFormat);
    bmpAfbeelding.SetResolution(img.HorizontalResolution,
          img.VerticalResolution);

    Graphics grAfbeelding = Graphics.FromImage(bmpAfbeelding);
    grAfbeelding.InterpolationMode =
          InterpolationMode.HighQualityBicubic;

    grAfbeelding.DrawImage(img, new Rectangle(0, 0, doelBreedte,
          doelHoogte), new Rectangle(pStart.X, pStart.Y, breedte,
          hoogte), GraphicsUnit.Pixel);
    grAfbeelding.Dispose();

    return bmpAfbeelding;
}
```

Eerst wordt een bitmap met juiste afmetingen en resolutie aangemaakt voor de resulterende afbeelding. Vervolgens maak ik een Graphics object, de oppervlakte waarop je iets tekent. Om te zorgen dat de afbeelding er niet korrelig uitziet als ze vergroot of verkleind wordt zet ik de InterpolationMode op HighQualityBicubic. Ten slotte teken ik het rechthoekig gebied en rek ik het uit zodat het heel de oppervlakte van de bitmap beslaat.

3.1.2.3. Promotiedia

Op deze dia worden een aantal promoties weergegeven. De schikking is afhankelijk van het aantal promoties die de beheerder toevoegt.
Deze promoties zijn eigenlijk een kopie van de artikels. Alle artikels worden centraal bewaard in een database zodat elke beheerder er gebruik van kan maken. Via het artikelbeheer kunnen er artikels aangemaakt en gewijzigd worden. Dit artikelbeheer is onafhankelijk van de presentaties: een beheerder kan op elk moment artikels bewerken en aanmaken.
Elk artikel heeft volgende eigenschappen:

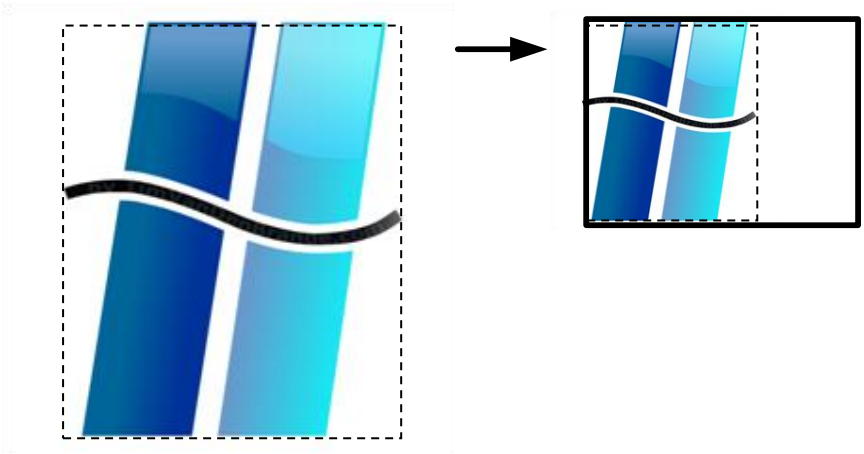
- Naam
- Prijs
- Eenheid: beschrijving van de verpakking (voorbeeld: stuk, fles, doos, …)
- Hoeveelheid: hoeveel, gemeten in de meeteenheid, de verpakking bevat
- Meeteenheid: (voorbeeld: kg, l, m², …)
- Omschrijving
- Promotieomschrijving: (voorbeeld: 2 + 1 gratis)
- Foto
- Taksen: lijst van taksen die in de prijs verrekend zijn

De foto van het artikel wordt verkleind (met behoud van de verhoudingen) tot de breedte maximaal 580 pixels en de hoogte maximaal 500 pixels is.
In de klasse ImageTool heb ik een methode scaleToFixedSize geschreven om deze taak uit te voeren.

```
        public static Image scaleToFixedSize(Image img, int doelBreedte,
int doelHoogte) {
            float percentBreedte;
            float percentHoogte;
            float factor;

            percentBreedte = ((float)doelBreedte/ (float)img.Width);
            percentHoogte = ((float)doelHoogte / (float)img.Height);


            if (percentHoogte < percentBreedte)
            {
                factor = percentHoogte;

            }
            else
            {
                factor = percentBreedte;

            }

            int breedte = (int)(img.Width * factor);
            int hoogte = (int)(img.Height * factor);
            …
            return bmpAfbeelding;
          }
```

Er worden 3 argumenten verwacht, de foto die verkleind moet worden, de maximale breedte en de maximale hoogte. Eerst wordt berekend met welke factor de oorspronkelijke breedte en hoogte vermenigvuldigd moeten worden om te zorgen dat de foto zo weinig mogelijk verkleind wordt maar toch past in de gewenste rechthoek (met een breedte gelijk aan doelbreedte en een hoogte gelijk aan doelhoogte). Vervolgens wordt de afbeelding aangemaakt met behulp van GDI+.

Op onderstaande figuur kan u zien hoe dit juist in zijn werk gaat. Het logo van de hogeschool wordt verkleind en de verhoudingen blijven behouden. Hoeveel de afbeelding verkleind moet worden is afhankelijk van de doelrechthoek. We verkleinen de afbeelding tot ze volledig binnen de doelrechthoek past.



Figuur 25 Een afbeelding zo weinig mogelijk verkleinen

We hebben een foto van onze promotie met de gewenste afmetingen. Nu moeten we de andere eigenschappen nog achterhalen. Een promotie kan beschouwd worden als een

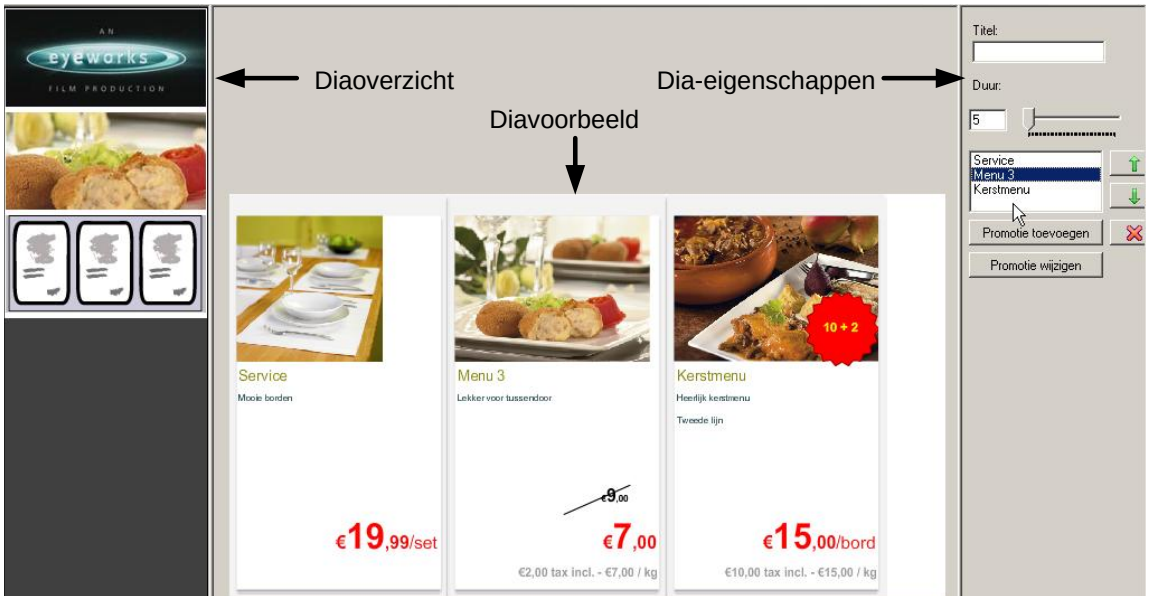
kopie van een artikel en een aantal extra eigenschappen. Artikeleigenschappen zoals de naam, omschrijving, eenheid en foto kunnen in een promotie niet meer gewijzigd worden. De prijs en de promotieomschrijving worden overgenomen van het artikel maar kunnen wel nog gewijzigd worden. Naast een 'gewone' prijs heeft een promotie ook een oude (doorstreepte) prijs die eveneens gewijzigd kan worden.

Een promotiedia kan één, twee, drie of vier promoties bevatten. De eigenschappen van elke promotie kunnen op elk moment gewijzigd worden en ook de volgorde van de promoties onderling kan aangepast worden.

Figuur 26 Promotiedia

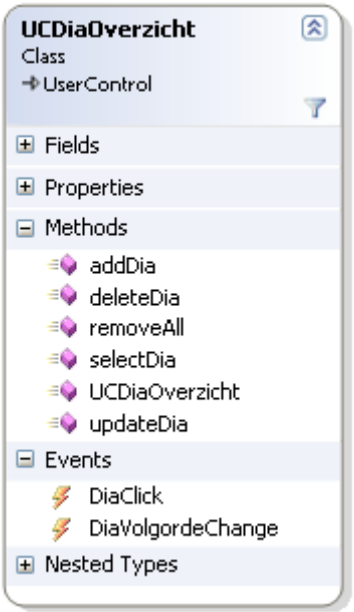
3.1.2.4. Dia's ordenen

De beheerder kan drie verschillende soorten dia's toevoegen aan een presentatie. Om ervoor te zorgen dat hij het overzicht niet verliest, worden alle dia's van de actieve presentatie weergegeven in een lijst. Voor elke dia in de presentatie wordt een voorbeeldafbeelding getoond die aangeeft wat er op de dia staat. Daarnaast ziet de beheerder een live voorbeeld van de actieve (geselecteerde) dia.



Figuur 27 Overzichtelijk diabeheer

De werking van de lijst met de verschillende dia's is nogal complex, daarom heb ik een User Control (UCDiaOverzicht) ontwikkeld die fungeert als façade.



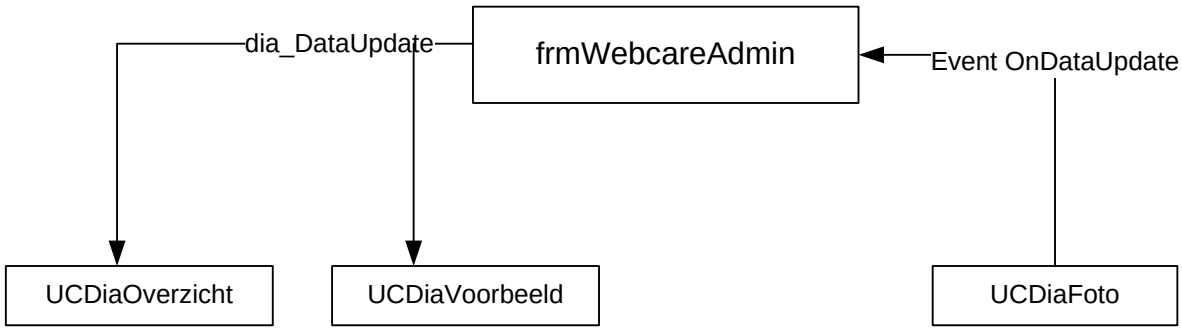
Figuur 28 De User Control verbergt de complexe werking van het diaoverzicht

De ingewikkelde operaties worden afgehandeld door de User Control terwijl de andere klassen het diaoverzicht kunnen raadplegen en aanpassen met een beperkt aantal methodes.

Laten we eerst even kijken hoe de verschillende soorten dia's voorgesteld worden.
Een fotodia wordt voorgesteld door een verkleinde afbeelding van de foto.
Voor videodia's wordt een schermafbeelding van het filmpje gebruikt.
En bij promotiedia's is de afbeelding afhankelijk van het aantal promoties die op de dia staan.

Als de beheerder een nieuwe fotodia toevoegt, is er nog geen foto geselecteerd. Om dit op te vangen is er een standaard afbeelding voorzien. Dit principe wordt ook toegepast bij de videodia en de promotiedia.
Als de beheerder de foto, het filmpje, het aantal promoties of de volgorde van de promoties wijzigt, wordt de afbeelding ook aangepast. Het venster moet dus verwittigd worden als de beheerder de eigenschappen van de actieve dia aanpast. Om dit te realiseren heb ik in elke User Control een gebeurtenis OnDataUpdate gedefinieerd die opgroepen wordt als de beheerder de eigenschappen van de dia verandert. Het venster wordt verwittigd en de voorbeeldafbeelding kan ververst worden.

```
public partial class UCDiaFoto : UserControl
{
    private DiaFoto diaFoto;
    public delegate void DataUpdateHandler(object sender, EventArgs
e,    object d);
    public event DataUpdateHandler DataUpdate;

    //Wordt getriggered als de eigenschappen van de dia gewijzigd
worden
    protected virtual void OnDataUpdate(EventArgs e)
    {
        if (DataUpdate != null)
        {

            DataUpdate(this, e, diaFoto);
        }
    }
    //Beheerder heeft foto gewijzigd -> venster verwittigen
    private void ucFileBrowser_DataUpdate(object sender, EventArgs e)
    {
        diaFoto.FotoNaam = Path.GetFileName(ucFileBrowser.Pad);
        OnDataUpdate(e);
    }
```



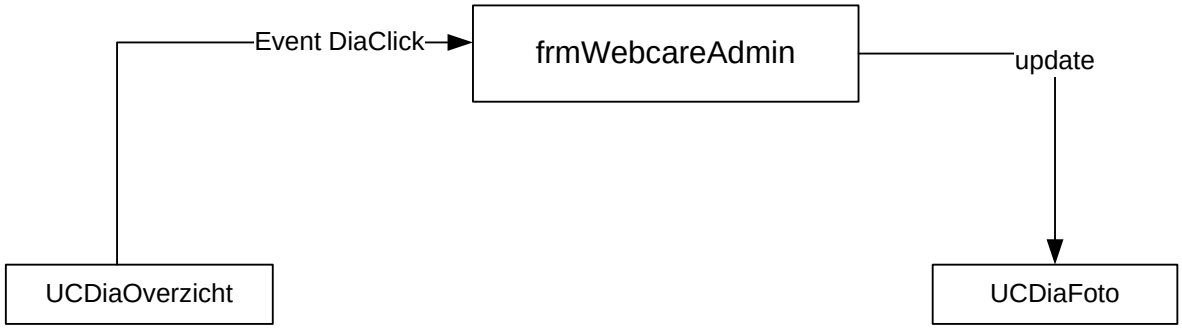
Figuur 29 Diavoorbeeld verversen als de eigenschappen van de dia gewijzigd worden

Bovenstaande code definieert de gebeurtenis OnDataUpdate in de User Control UCDiaFoto. Als de beheerder de foto wijzigt, wordt enerzijds de eigenschap FotoNaam van het diaobject aangepast en anderzijds wordt de gebeurtenis DataUpdate opgroepen. Een klasse die luistert naar deze gebeurtenis krijgt naast het object dat de gebeurtenis veroorzaakt heeft en de gebeurtenisargumenten ook een diaobject mee.

```
public frmWebcareAdmin()
{
  ucDiaFoto = new UCDiaFoto();
  ucDiaFoto.DataUpdate +=new
UCDiaFoto.DataUpdateHandler(dia_DataUpdate);

  ucDiaVideo = new UCDiaVideo();
  ucDiaVideo.DataUpdate +=new
UCDiaVideo.DataUpdateHandler(dia_DataUpdate);
    …
}
private void dia_DataUpdate(object sender, EventArgs e, object dia) {
    ucDiaOverzicht.updateDia(getPreviewAfbeeldingPad((Dia)dia));
    toonDiaPreview(ucDiaOverzicht.HuidigVolgNr);
}
```

Zoals u kunt zien in bovenstaande code wordt er één methode (dia_DataUpdate) gebruikt om de wijzigingen van alle diasoorten op te vangen.

We kunnen de voorbeeldafbeelding van de dia verversen als de beheerder de eigenschappen van een dia wijzigt. Nu moeten we zorgen dat eigenschappen van de dia ververst worden als de beheerder een andere dia selecteert. De werking is gelijkaardig, het venster wordt verwittigd als de beheerder op een dia klikt (DiaClick gebeurtenis) en zal de eigenschappen van de dia verversen.



Figuur 30 Dia-eigenschappen verversen als er een andere dia geselecteerd wordt

Naast de gebeurtenis DiaClick heeft het diaoverzicht nog een andere gebeurtenis: DiaVolgordeChange. De beheerder kan een dia slepen en neerzetten om de volgorde van de dia's te wijzigen. De volgorde van de dia's is dan zichtbaar gewijzigd maar de wijziging is nog niet doorgegeven aan de actieve presentatie. Het venster moet dus verwittigd worden wanneer de volgorde van de dia's wijzigt om vervolgens de volgorde van de dia's aan te passen in het presentatieobject.

```
private void ucDiaOverzicht_DiaVolgordeChange(object sender,
EventArgs e)
{
PresentatieBeheerder.getInstance().ActievePresentatie.changeDiaVolgor
de(ucDiaOverzicht.OudVolgNr, ucDiaOverzicht.NieuwVolgNr);
}
```

We hebben gezien dat het venster wordt verwittigd telkens de beheerder eigenschappen van een dia aanpast, een dia selecteert of de volgorde van de dia's wijzigt. Vervolgens zal het venster bepalen welke User Controls ververst moeten worden en of het actieve presentatieobject gewijzigd moet worden.

3.1.3. Presentaties exporteren

Alle presentaties worden bewaard in de centrale gedeelde map. Ze kunnen geopend en gewijzigd worden op de centrale maar zijn nog niet zichtbaar op de schermen in de winkels. Er moet immers bepaald kunnen worden waar welke presentaties getoond moeten worden.

Door een presentatie te exporteren kan de beheerder aangeven op welke schermen en gedurende welke periode een presentatie getoond mag worden.
Het zou niet erg gebruiksvriendelijk zijn als de beheerder enkel een lijst zou zien van alle schermen met een checkbox ernaast. De beheerder zou dan moeten weten welke schermen in welke winkels staan. Bovendien zou het dan niet eenvoudig zijn om schermen te selecteren die aan bepaalde criteria voldoen.
Ik heb ervoor gekozen om de hiërarchische indeling van de schermen voor te stellen met een boom waarvan elke knoop een vakje is dat zich in 3 staten kan bevinden: aangevinkt, uitgevinkt of tussenin. Als de beheerder een vakje aanvinkt (uitvinkt) op een bepaald niveau worden ook alle kinderen automatisch aangevinkt (uitgevinkt). De derde status wordt gebruikt als niet alle kinderen van een knoop zijn aangevinkt (uitgevinkt).

Na een zoektocht op het internet vond ik een zogenaamde Tri-State Treeview Control die bruikbaar was in .NET. Omdat deze boom ook gebruikt kan worden om schermontwerpen te exporteren, heb ik een herbruikbare User Control (UCDisplayTree) ontwikkeld. Laten we even kijken hoe deze User Control werkt.
Eerst wordt alle data (omgevingen, regio's, winkels, afdelingen en categorieën en hun onderlinge relatie) opgehaald uit de database. Vervolgens worden alle knopen aangemaakt. Voor omgevingen, regio's en winkels is dit vrij eenvoudig omdat een winkel maar in één regio kan zitten en een regio maar in één omgeving. Maar voor afdelingen en categorieën is iets meer werk nodig.
Een categorie zit in een bepaalde afdeling maar een afdeling kan voorkomen in meerdere winkels. We moeten deze veel op veel relatie dus op een of andere manier omzetten naar een hiërarchische structuur.
In .NET is het vrij eenvoudig om deze relaties voor te stellen in een boomstructuur. Ik haal eerst alle gegevens op uit de database en bewaar ze in DataSets. Daarna overloop ik niveau per niveau (eerst omgevingen dan regio's, …) de gegevens en maak de nodige boomknopen aan. En met de methode Select van de klasse DataTable kunnen de gegevens gefilterd worden zonder dat ze opnieuw opgehaald moeten worden uit de database.

Een voorbeeldje om dit te verduidelijken. Het bovenste niveau bestaat uit de omgevingen. In elke omgeving zitten een aantal regio's. Ik zou dus per omgeving telkens een databaseaanvraag kunnen doen om alle regio's van die omgeving op te vragen. Omdat ik sowieso alle regio's nodig heb, haal ik ze allemaal op voorhand op. Als ik de omgevingen overloop selecteer ik enkel de regio's die tot die omgeving behoren.

```
DataSet dsOmgevingen = DataAccess.getInstance().getOmgevingen();
DataSet dsRegios = DataAccess.getInstance().getRegios();
DataSet dsWinkels = DataAccess.getInstance().getWinkelRelatie();
…
```

```
            foreach (DataRow rowRegio in
            dsRegios.Tables[0].Select("regOmgevingID =" +
            rowOmgeving["omgID"])){
            …
            }
}
```

De boomstructuur geeft de beheerder de mogelijkheid om heel snel bijvoorbeeld alle schermen in een bepaalde winkel, regio of omgeving te selecteren. Daarnaast heb ik ook een mogelijkheid voorzien om alle schermen in een bepaalde afdeling of categorie te selecteren. Hiervoor zijn twee vervolgkeuzelijsten voorzien met elk een checkbox ervoor.

De hiërarchische indeling van de schermen is enkel van belang voor de beheerder. Om de presentaties te exporteren is enkel een lijst van winkels nodig met per winkel een lijst van schermen die geselecteerd werden:

```
public Dictionary<String,Winkel> getSelectedWinkels()
```

Figuur 31 Klassendiagram: presentaties exporteren

We hebben dus een lijst met alle schermen waarop de presentatie getoond moet worden. Nu moet nog aangegeven worden wanneer de presentatie juist getoond moet worden. De beheerder vult hiertoe vier velden in:

- Datum van: de presentatie is geldig vanaf deze dag.
- Datum tot: de presentatie is geldig tot en met deze dag.
- Uur van: de presentatie is geldig tussen de opgegeven datums vanaf dit uur.
- Uur tot: de presentatie is geldig tussen de opgegeven datums tot en met dit uur.

We beschikken nu over alle informatie: alle schermen waarop de presentatie gedurende een bepaalde periode getoond moet worden. Het overzicht van de presentaties wordt bijgewerkt en de benodigde bestanden worden gekopieerd naar de FTP-server. Het is nu de taak van de service in de winkels om de nieuwe presentatie op te halen en te verspreiden naar de juiste schermen.

3.1.4. Schermontwerp

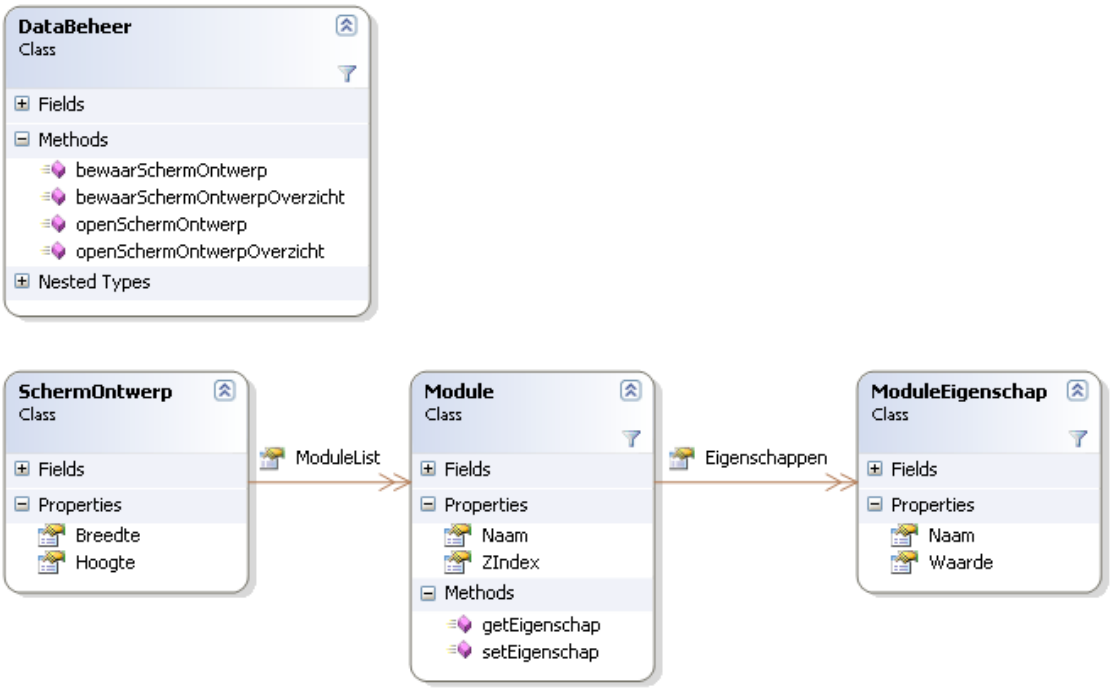
We hebben al gezien dat de beheerder presentaties kan exporteren naar één of meerdere schermen. Naast presentaties kunnen er ook nog andere modules op het scherm getoond worden.

In het oude systeem werd een statische positionering gebruikt voor alle elementen die op het scherm getoond werden. Zo werd er rechts altijd een balk getoond (met vaste grootte) met het logo van de centrale, links een presentatie en rechtsonderdaan eventueel het nummer van de volgende klant.

Deze vaste lay-out is niet erg handig, zeker als er verschillende formaten van schermen gebruikt worden. Daarom heb ik ervoor gekozen om in het nieuwe systeem te werken met modules. Zo'n module is een grafisch element dat gelijk waar op het scherm geplaatst kan worden en niet gebonden is aan een vaste grootte. Dit geeft beheerders de mogelijkheid om op bepaalde schermen bijvoorbeeld enkel het nummer van de volgende klant te tonen. Het is niet alleen belangrijk dat men exact kan bepalen welke module waar moet komen maar ook dat er later eenvoudig nieuwe modules gecreëerd kunnen worden.
Om dit te realiseren heb ik een overzicht van de modules bijgehouden in een XML-bestand. Wanneer men beslist om later een nieuwe module te creëren, hoeft men deze enkel op te nemen in dit XML-bestand. In de presentatiebeheerder wordt dit bestand ingelezen en krijgt de beheerder alle beschikbare modules te zien.

```
<?xml version="1.0" encoding="utf-8" ?>
- <SchermOntwerp
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    Breedte="1280" Hoogte="800">
  - <Modules>
    - <Module>
        <Naam>VolgendeKlant</Naam>
      - <Eigenschappen>
          <Eigenschap naam="x" waarde="0" />
          <Eigenschap naam="y" waarde="0" />
          <Eigenschap naam="breedte" waarde="100" />
          <Eigenschap naam="hoogte" waarde="100" />
          <Eigenschap naam="tekst" waarde="Wij bedienen" />
        </Eigenschappen>
      </Module>
    + <Module>
    + <Module>
    </Modules>
  </SchermOntwerp>
```

Figuur 32 De beschikbare modules



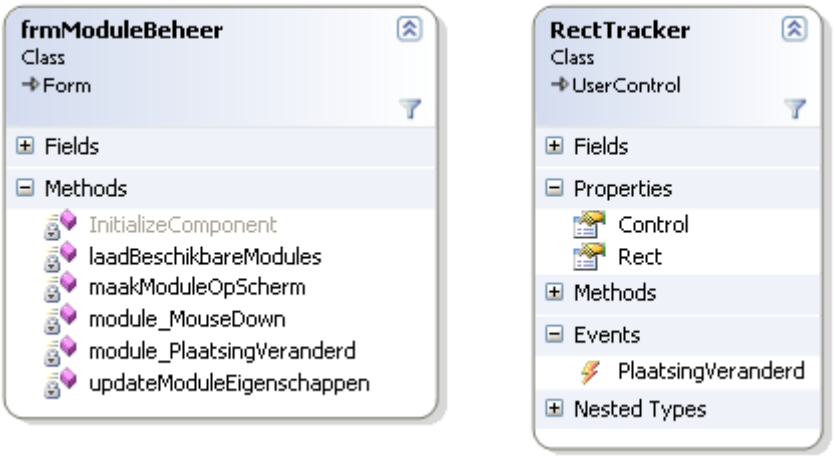
Figuur 33 Klassendiagram: Schermontwerp

In de presentatiebeheerder kan de beheerder een scherm 'ontwerpen'. Dit wil zeggen dat hij kan bepalen waar en hoe groot een bepaalde module getoond moet worden. Alle beschikbare modules worden bijgehouden in een XML-bestand (zie Figuur 32). Wanneer een beheerder kiest om een nieuw schermontwerp te maken of een bestaand schermontwerp te openen, krijgt hij alle beschikbare modules te zien. Zo'n module bestaat uit een lijst met eigenschappen. Een aantal van deze eigenschappen bepalen waar de module getoond wordt en hoe groot ze wordt weergegeven (x, y, breedte en hoogte). Bijkomende eigenschappen kunnen vrij gekozen worden. Zo heeft de volgende-klant-module de eigenschap 'tekst' die bepaalt welke boodschap (bv. 'Wij bedienen:') wordt weergegeven boven het nummer van de volgende klant.
Later kan men dus heel eenvoudig een nieuwe module toevoegen die bijvoorbeeld het laatste nieuws over het scherm laat schuiven. Er kan dan bijvoorbeeld een bijkomende eigenschap 'RSS-feed' gedefinieerd worden.

De beheerder krijgt alle beschikbare modules te zien in de vorm van een knop met een bepaalde kleur (die ook dynamisch bepaald wordt). Als hij op zo'n knop klikt, wordt de module toegevoegd aan het virtuele scherm. De module kan versleept worden en ook de grootte kan gewijzigd worden door de muis naar een hoek van de module te verplaatsen en de linkermuisknop ingedrukt te houden en vervolgens de muis te verslepen.
Dit systeem kan het best vergeleken worden met de manier waarop men in Visual Studio formulieren ontwerpt. Als de beheerder een module toevoegt, wordt er een label aangemaakt met als tekst de naam van de module. Dit label stelt de module voor op het virtuele scherm. Vervolgens maak ik een instantie van de User Control RectTracker[5]. Deze User Control voert alle acties uit die nodig zijn om de module te verplaatsen en uit te rekken.

Het klassendiagram en bijhorende code verduidelijken hoe dit juist werkt:

[5] http://www.codeproject.com/KB/miscctrl/CSharpRectTracker.aspx



Figuur 34 Klassendiagram: dynamische plaatsing van modules

```
//Module aanmaken op het virtuele scherm
private void maakModuleOpScherm(Module module, Color kleur) {
      Label lblModule = new Label();
      lblModule.Text = module.Naam;
      lblModule.BackColor = kleur;
…
}
//Als de beheerder op een module klikt
private void module_MouseDown(object sender, MouseEventArgs e) {
      CSharpTracker = new RectTracker((Control)sender);
      CSharpTracker.PlaatsingVeranderd += new
RectTracker.PlaatsingVeranderdHandler(module_PlaatsingVeranderd);
      this.Controls.Add(CSharpTracker);
      CSharpTracker.BringToFront();
      CSharpTracker.Draw();
…
}
```

Als de beheerder op een module klikt op het virtueel scherm, krijgt hij ook alle eigenschappen van de module te zien. Dit is mogelijk omdat het venster verwittigd wordt door de RectTracker (gebeurtenis: PlaatsingVeranderd).

Elke eigenschap wordt weergegeven in een tekstveld. Er is dus een paneel dat afhankelijk van de geselecteerde module een aantal tekstvelden bevat met de waarden van de eigenschappen. De wijzigingen van de eigenschappen zijn ook direct merkbaar. Als de beheerder bijvoorbeeld de breedte van de module aanpast door de waarde te veranderen van het bijhorende tekstvak dan wordt de breedte van de module op het virtueel scherm ook onmiddellijk gewijzigd.

Waneer de beheerder tevreden is over de plaatsing, de grootte en de eigenschappen van de verschillende modules, kan hij het schermontwerp bewaren. Nadien kan het schermontwerp toegewezen worden aan één of meerdere schermen. Het toewijzen van een schermontwerp is volledig analoog aan het toewijzen van een presentatie aan een scherm (zie 3.1.3). Een schermontwerp is in tegenstelling tot een presentatie wel altijd geldig.

3.2. Schermbeheerder

We hebben alle ingrediënten om iets aantrekkelijk te tonen op de schermen: presentaties, mediabestanden en het schermontwerp. Onder schermen verstaan we enerzijds display units waaraan een tv-scherm gekoppeld is en anderzijds weegschaalschermen.
Op elke display unit staat een .NET-applicatie die de presentaties weergeeft en de beheerder de mogelijkheid geeft de configuratie van het scherm aan te passen. Dit programma wordt opgestart als de computer wordt opgestart. Eerst wordt een configuratiebestand ingelezen dat de naam van het scherm bevat en aangeeft of de presentaties op het eerste of het tweede scherm getoond moeten worden. Vervolgens wordt een venster getoond met daarin een ActiveX-component die de presentaties afspeelt. Als de beheerder op de Esc-toets drukt, verdwijnt het venster met de presentaties en krijgt hij het configuratievenster te zien.

Het Flexprogramma (dat ingeladen wordt door de ActiveX-component) toont de verschillende modules en speelt de presentaties af. Dit programma is beschikbaar als een SWF[6]-bestand. Met dit bestand op zich zijn we in principe niet veel, we hebben nog een ander programma nodig om het SWF-bestand af te spelen. De eerste vereiste is dat de Flash Player geïnstalleerd is op de computer van de klant. Eenmaal de Flash Player aanwezig is, hebben we een aantal mogelijkheden om het SWF-bestand af te spelen: in een webbrowser, in de standalone Flash Player of in een ActiveX-component. Een webbrowser of een standalone Flash Player volstaan als je enkel een SWF-bestand wilt afspelen. Dit is echter niet voldoende als je vanuit een programma methodes (bijvoorbeeld een methode om het nummer van de volgende klant te tonen) in het SWF-bestand wenst op te roepen. Een ActiveX-component die ingebed is in een .NET-applicatie (of een Delphi-applicatie op weegschalen) ondersteunt deze communicatie wel.

Voor we berichten kunnen uitwisselen tussen de .NET-applicatie en het SWF-bestand moet de ActiveX-component juist geconfigureerd worden. Volgende code toont hoe dit kan gebeuren:

```
//De locatie van de ActiveX-component
_axShockwaveFlash = new AxShockwaveFlash();
_axShockwaveFlash.Location = new System.Drawing.Point(0, 0);
this.Controls.Add(_axShockwaveFlash);
_axShockwaveFlash.Size = new
  System.Drawing.Size(_screen.Bounds.Size.Width,
_screen.Bounds.Size.Height);

//De component configureren en het Flexprogramma afspelen
_axShockwaveFlash.AllowScriptAccess = "sameDomain";
_axShockwaveFlash.LoadMovie(0, "MediaDisplay.swf");
_axShockwaveFlash.Loop = false;
_axShockwaveFlash.Play();
```

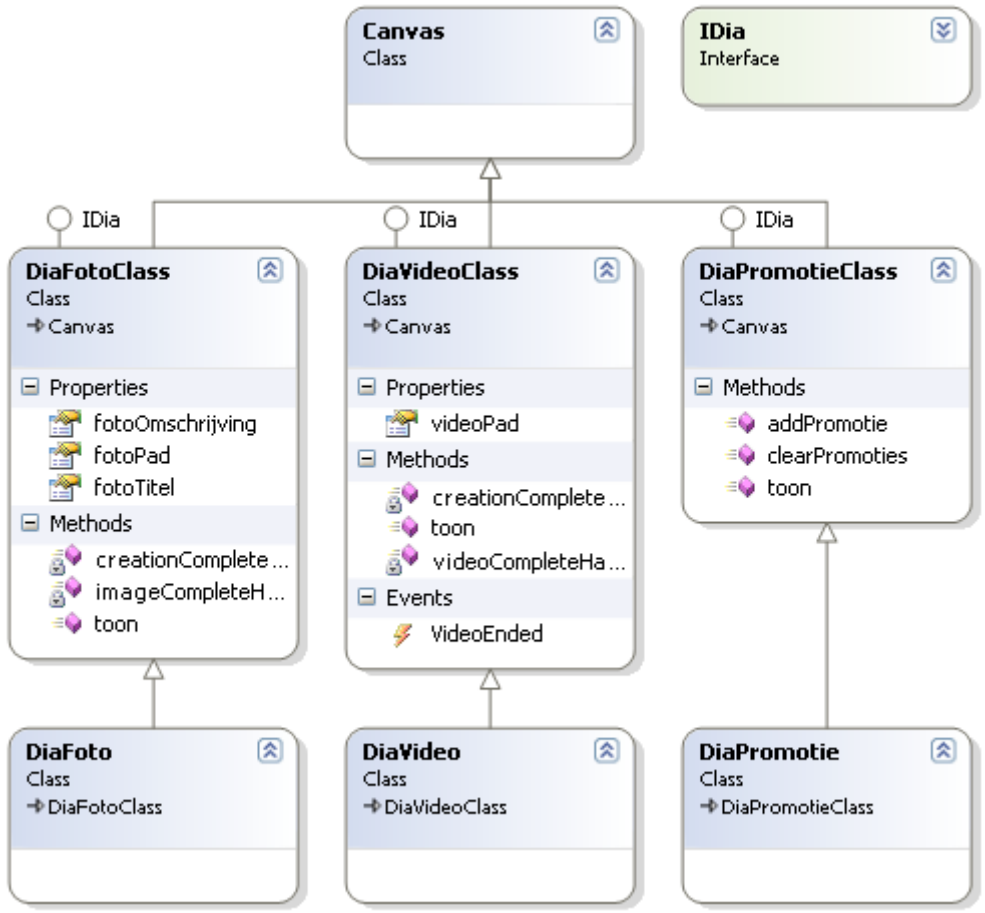
De ActiveX-component wordt aangemaakt en onmiddellijk toegevoegd aan het venster. Het is belangrijk dat de ActiveX-component wordt toegevoegd aan een container die zichtbaar is voordat er een filmpje ingeladen wordt. Als je deze volgorde niet respecteert krijg je nadien foutmeldingen.
Nadien worden de breedte en de hoogte van de component zo ingesteld dat hij heel het scherm vult.

[6] Shockwave Flash: een bestandsformaat voor multimedia, uitgesproken als swif.

Om te zorgen dat we methodes in het SWF-bestand kunnen oproepen, moeten we dit expliciet aangeven door de eigenschap *AllowScriptAccess* van de component in te stellen op *sameDomain*.
De configuratie van de ActiveX-component is voltooid. Het enige wat we nu nog moeten doen, is het SWF-bestand laden en afspelen door de methodes *LoadMovie* en *Play* op te roepen.

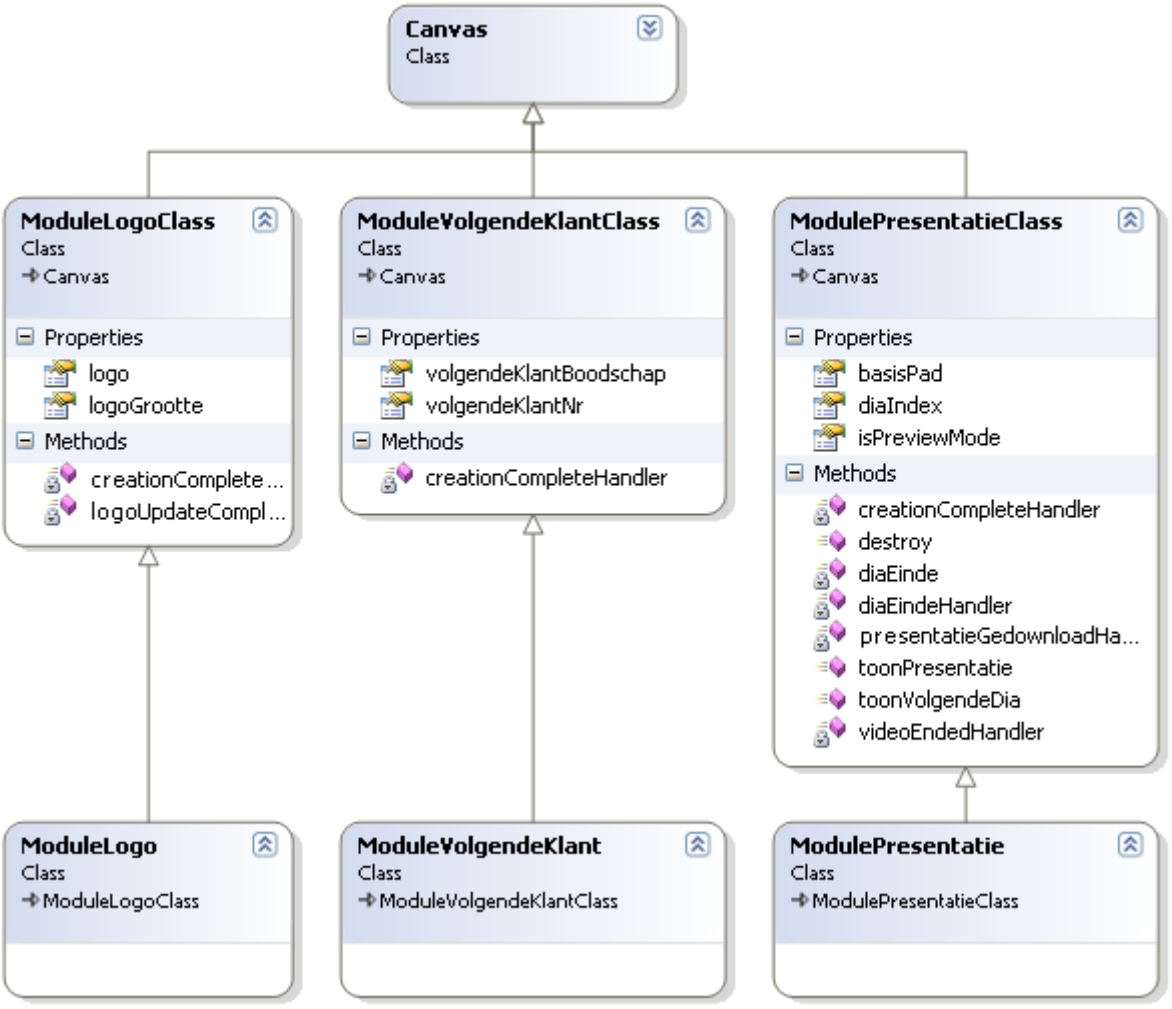
3.3. Presentatiespeler

Het programma dat de verschillende modules op het scherm toont en de presentaties afspeelt, is ontwikkeld in Flex[7]. De volgende diagrammen geven een algemeen beeld van de structuur van deze Flexapplicatie.



Figuur 35 Klassendiagram: de verschillende soorten dia's in Flex

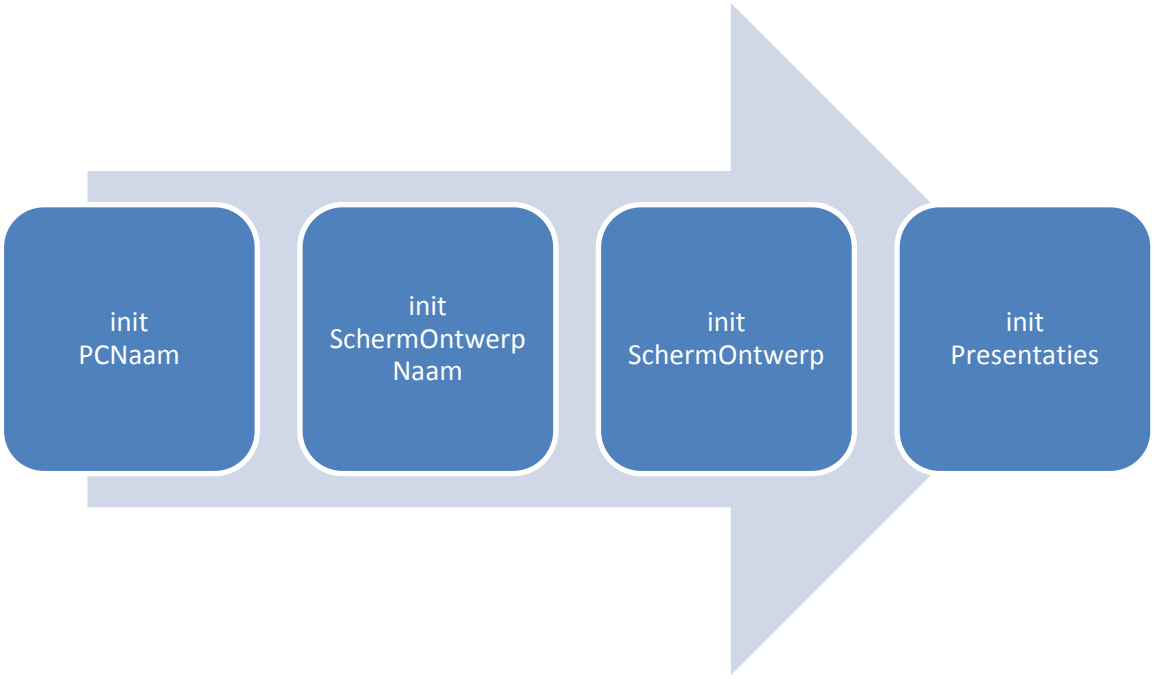
[7] Een Richt Internet Framework. Meer informatie over Flex kan u terugvinden in hoofdstuk 5.1.



Figuur 36 Klassendiagram: de verschillende modules in Flex

De modules en de dia's zijn ontwikkeld volgens de *code-behind* (zie 5.1.2.3) techniek. Klassennamen die eindigen op *class* zijn ActionScriptklassen en erven van de Canvasklasse. De andere klassen zijn MXML-componenten die erven van de ActionScriptklassen.

Voor we presentaties kunnen tonen, moeten er een aantal zaken geïnitialiseerd worden. Eerst moeten we weten op welk scherm de applicatie draait zodat we het juiste schermontwerp kunnen laden en de juiste presentaties kunnen weergeven. Dit initialisatieproces bestaat uit een aantal stappen:



Figuur 37 De initialisatiefase van de presentatiespeler

Om een beter beeld te krijgen van wat er juist gebeurt, zal ik een ingekorte versie van de code bespreken.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
        xmlns:mx="http://www.adobe.com/2006/mxml"
        layout="absolute"
        creationComplete="init()">
         <mx:HTTPService
                id="srvPCNaam"
                url="parameters.xml"
                resultFormat="e4x"
                result="initPCNaam(event);"/>

        <mx:HTTPService
                id="srvSchermOntwerpNaam"
                url="module-summary.xml"
                resultFormat="e4x"
                result="initSchermOntwerpNaam(event);"/>

        <mx:HTTPService
                id="srvSchermOntwerp"
                resultFormat="e4x"
                result="initSchermOntwerp(event);"/>

        <mx:HTTPService
                id="srvPresentaties"
                url="pres-summary.xml"
                resultFormat="e4x"
                result="initPresentaties(event);"/>
```

Eerst worden er een aantal HTTPService objecten aangemaakt die verantwoordelijk zijn voor het ophalen van configuratiebestanden.

```
private function init():void{
        ...
        srvPCNaam.send();
}
```

```
/**
* Naam of ip van de computer ophalen
*/
private function initPCNaam(event:ResultEvent):void{

	var xmlData:XML = event.result as XML;
	this._PCNaam = String(xmlData.Schermnaam);

	//naam van het XML-bestand ophalen dat het SchermOntwerp bevat

	srvSchermOntwerpNaam.send();
}

/**
* Naam van het XML-bestand ophalen dat het SchermOntwerp bevat
*/
private function initSchermOntwerpNaam(event:ResultEvent):void{
	var xmlData:XML = event.result as XML;
	this._SchermOntwerpNaam = String(xmlData.Scherm.(@Naam == this._PCNaam).@Ontwerp);


	srvSchermOntwerp.url = _pad + "modules\\" +  this._SchermOntwerpNaam;
	srvSchermOntwerp.send();
}
```

We hebben de schermnaam en kunnen nu het bijhorende schermontwerp opvragen. Via een asynchrone aanvraag zal de HTTPService *srvSchermOntwerpNaam* de naam van het schermontwerp ophalen. Als we de naam van het schermontwerp hebben kunnen we het schermontwerp zelf ophalen met behulp van de HTTPService *srvSchermOntwerp*.

```
/**
* Het XML-bestand met het schermontwerp overlopen en de modules op het scherm plaatsen
*/
private function initSchermOntwerp(event:ResultEvent):void{

	var xmlData:XML = event.result as XML;

	for each (var module:XML in xmlData.Modules.Module){

		//algemene eigenschappen van de module ophalen
		...

		//de initialisatie is anders per module
		switch(moduleNaam){

			case "VolgendeKlant":
				...
			break;

			case "Presentatie":
				this._ModulePresentatie = new ModulePresentatie();

				//event als presentatie afgelopen is
				_ModulePresentatie.addEventListener("presentatieEnded",
presentatieEndedHandler);

				//event als de component geladen is
				_ModulePresentatie.addEventListener("presentatieModuleLoaded",
presentatieModuleLoadedHandler);
				...
			break;

			case "LogoBalk":
				...
			break;
		}//END switch
```

```
        }//END foreach
}
```

In het schermontwerp staat beschreven welke modules er getoond moeten worden op het scherm. We overlopen al de modules en initialiseren ze.
De eerste presentatie kan pas getoond worden wanneer de presentatiemodule volledig geladen is. We worden hiervan op de hoogte gebracht door de methode *presentatieModuleLoadedHandler*.
Als een presentatie eindigt, worden we op een gelijkaardige manier verwittigd door de methode *presentatieEndedHandler*.

```
/**
* Wordt opgeroepen als de presentatiemodule geladen is
*/
private function presentatieModuleLoadedHandler(event:flash.events.Event):void{
        //overzicht van presentaties ophalen
        srvPresentaties.send();
}
```

We weten nu dat de presentatiemodule geladen is en kunnen het presentatieoverzicht ophalen.

```
/**
* Bewaart het overzicht van presentaties in een object en zorgt dat de eerste presentatie
getoond wordt
*/
private function initPresentaties(event:ResultEvent):void{
        //presentaties voor dit scherm uitlezen
        var xmlData:XML = event.result as XML;
        var xmlPresentatieOverzicht:XMLList = xmlData..Scherm.(@PCNaam ==
this._PCNaam).Presentaties;

        //overzicht bewaren in een object
        _PresentatieOverzicht = new PresentatieOverzicht();
        _PresentatieOverzicht.xmlData = xmlPresentatieOverzicht;

        //presentatiemodule en het presentatieoverzicht is geladen, toon de eerste presentatie
        toonPresentatie( PresentatieOverzicht.getNextPresentation());


}
```

Het presentatieoverzicht wordt uitgelezen en we bewaren enkel een lijst van presentaties die getoond moeten worden op dit scherm. We zijn nu klaar om de eerste presentatie die geldig is te tonen.

```
/**
* Toont een presentatie
*/
private function toonPresentatie(naam:String):void{
        if (this._ModulePresentatie != null)
                _ModulePresentatie.toonPresentatie(_pad + "presentations\\" + naam + ".xml");

}
```

De presentatie afspelen, is de taak van de presentatiemodule. Deze module zal alle dia's overlopen en ze één voor één weergeven.

```
/**
* Wordt opgeroepen als de presentatie is afgelopen.
* De volgende presentatie wordt getoond
*/
private function presentatieEndedHandler(event:flash.events.Event):void{
	//Er is een presentatie afgelopen, toon de volgende
	//Niets doen als we aan een update bezig zijn
	if (this._ModulePresentatie != null)
		toonPresentatie( PresentatieOverzicht.getNextPresentation());
}
```

Telkens een presentatie afgelopen is, worden we verwittigd en kunnen we de volgende presentatie afspelen.

```
/**
* Controleert om de 5 sec of er een update is
*/
private function checkForUpdate(event:TimerEvent):void{...}

/**
* Verwijdert alle modules als er een update start.
* een update start als er een bestand update.txt gevonden wordt in de huidige directory
*/
private function updateStartHandler(event:flash.events.Event):void{...}

/**
* 'Herstart' het programma als een update compleet is.
* een update is 'complete' als het bestand update.txt verdwenen is
*/
private function updateCompleteHandler(event:flash.events.IOErrorEvent):void{
	if(_updateInProgress){
		_updateInProgress = false;
		srvPCNaam.send();
	}
}
```

Als de service in de winkel nieuwe of gewijzigde presentaties kopieert dan wordt er een updatebestand achtergelaten op de display unit. Tijdens een update mogen we geen presentaties afspelen omdat bepaalde bestanden dan in gebruik zouden kunnen zijn. Na een tijdje verdwijnt het updatebestand en halen we opnieuw de computernaam op. Zo wordt de Flexapplicatie opnieuw geïnitialiseerd. Dit is noodzakelijk want het schermontwerp en het presentatieoverzicht kunnen gewijzigd zijn.

3.3.1. Presentatiemodule

De presentatiemodule toont een presentatie. Dat wil zeggen dat een XML-bestand wordt ingelezen en dat aan de hand daarvan de verschillende dia's gecreëerd en weergegeven worden. De dia's worden niet elke keer opnieuw gecreëerd, van elk type dia is een object aanwezig dat opgevuld wordt met de juiste data.

Tijdens het testen van deze module steeg het geheugenverbruik en daalde het pas na een aantal minuten. Dit is te wijten aan de garbagecollector. Het gedrag van deze garbagecollector is ingewikkeld en niet eenvoudig om te beschrijven. Voorspellen wanneer de garbagecollector zal starten is dus niet vanzelfsprekend. Het enige wat we

weten is dat de garbagecollector actief wordt als we geheugen toewijzen. Dit betekent dat het geheugenverbruik dus ook niet zal veranderen als er geen objecten aangemaakt worden.
Na een zoektocht op het internet vond ik volgende code die verzekert dat de garbagecollector geactiveerd wordt:

```
try{

	var lc1:LocalConnection = new LocalConnection();
	var lc2:LocalConnection = new LocalConnection();

	lc1.connect( "gcConnection" );
	lc2.connect( "gcConnection" );
}
catch (e:Error) {}
```

Na het vertonen van een dia wordt dit stukje code opgeroepen zodat het geheugenverbruik niet blijft stijgen.

Om de algemene werking van deze module beter te begrijpen, heb ik hieronder een heel sterk ingekorte versie geplaatst:

```
public class ModulePresentatieClass extends Canvas
{
	public function toonPresentatie(pad:String):void{...}
	private function toonVolgendeDia():void{...}

	//De speelduur van de dia is afgelopen -> volgende dia
	private function diaEindeHandler(event:TimerEvent):void{
		diaEinde();
	}

	//Het filmpje op de videodia is afgelopen -> volgende dia
	private function videoEndedHandler(event:flash.events.Event):void{
		diaEinde();
	}

	private function diaEinde():void{
		//START GARBAGE COLLECTION ...

		if (_diaIndex > _diaMaxIndex)
			dispatchEvent(new Event("presentatieEnded"));
		else
			this.toonVolgendeDia();
	}
}
```

3.3.1.1. Videodia

In de presentatiebeheerder kan de beheerder verschillende soorten dia's toevoegen aan een presentatie. De presentatiespeler moet dus voor elke diasoort een grafische component hebben.
De videodia in Flex bevat een *VideoDisplay* om de FLV-bestanden af te spelen.
Eenmaal de creatie van de component voltooid is, wordt het filmpje geladen. Als het filmpje eindigt, willen we graag de volgende dia tonen. We koppelen dus de gebeurtenis

VideoEvent.COMPLETE aan de *VideoDisplay*. Op het moment dat het filmpje eindigt, geven we de gebeurtenis door aan de presentatiemodule en kan de volgende dia getoond worden.

```
public class DiaVideoClass extends Canvas implements IDia
{
	[Event(name="videoEnded", type="flash.events.Event")]

	public var videoPlayer:VideoDisplay;
	private var _videoPad:String;
	…

	private function creationCompleteHandler(event:FlexEvent):void{
		videoPlayer.source = this. videoPad;
		videoPlayer.addEventListener(VideoEvent.COMPLETE, videoCompleteHandler);
	}

	private function videoCompleteHandler(event:VideoEvent):void{
			dispatchEvent(new Event("videoEnded"));
	}
}
```

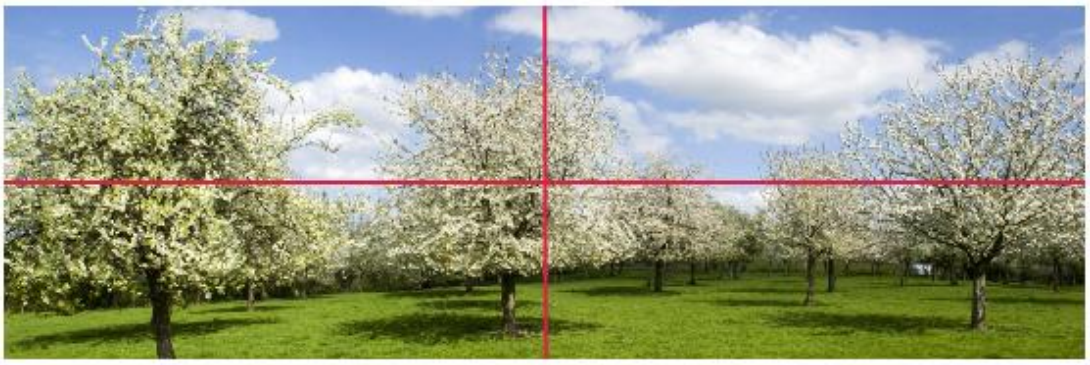
3.3.1.2. Fotodia

De fotodia toont een foto en eventueel een titel en een omschrijving.
Het is de bedoeling dat de foto zo groot mogelijk getoond wordt en dat er geen witte randen zichtbaar zijn. Er moet dus ingezoomd worden op de foto zodat deze de hele presentatiemodule beslaat.
We zouden de foto ook kunnen uitrekken maar dat is visueel niet aantrekkelijk en bovendien daalt de beeldkwaliteit.
De beheerder kan de presentatiemodule elke mogelijke grootte geven dus moeten we een algemene oplossing zoeken.

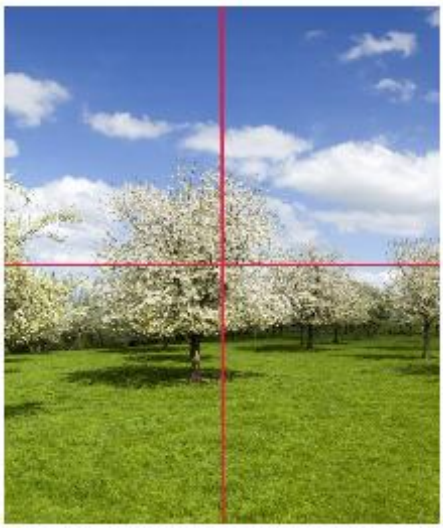
```
public class DiaFotoClass extends Canvas implements IDia{
…
	private function imageCompleteHandler(event:Event):void{
		//afbeelding is geladen, juist positioneren en effecten spelen

		var afbVerhouding:Number = 1366/768; //vaste verhouding
		var frmVerhouding:Number = this.width/this.height;//module
		var f:Number;

		//if (afbVerhouding == frmVerhouding)
		//formaat afbeelding en formaat module zijn gelijk
		//de afbeelding kan in dit formaat getoond worden

		if(frmVerhouding < afbVerhouding){
			f = this.height/768;
			imgFoto.height = this.height;
			imgFoto.width = 1366*f;
			imgFoto.x = (this.width - imgFoto.width)/2;
		}else{
			f = this.width/1366;
			imgFoto.width = this.width;
			imgFoto.height = 768*f;
			imgFoto.y = (this.height - imgFoto.height)/2;
		}
```

Eenmaal de foto geladen is, wordt de verhouding van de foto berekend.
Er zijn nu twee mogelijkheden. Ofwel is de module te breed t.o.v. de hoogte: de verhouding van de module is dan groter dan de verhouding van de afbeelding. We zoomen dus in zodat de breedte van de foto overeenkomt met de breedte van de module. Een stuk van de onderkant en een stuk van de bovenkant van de foto vallen buiten de module en zijn niet meer zichtbaar.
Is de module te hoog dan volgen we de omgekeerde redenering. Nu zal er links en rechts een stuk van de foto verdwijnen.
Zoals u kan zien op volgende figuren wordt er steeds ingezoomd op het midden van de foto.

Figuur 38 De oorspronkelijke foto



Figuur 39 De foto weergegeven in een module die te breed is

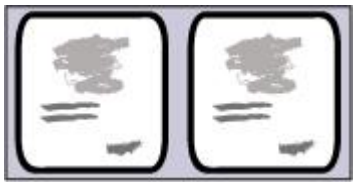


Figuur 40 De foto weergegeven in een module die te hoog is

3.3.1.3. Promotiedia

De promotiedia kan een variabel aantal (maximum vier) promoties bevatten. Het was dus geen eenvoudige opgave om een dynamische lay-out te ontwerpen.
De lay-out van de dia is afhankelijk van het aantal promoties:

één horizontaal artikel



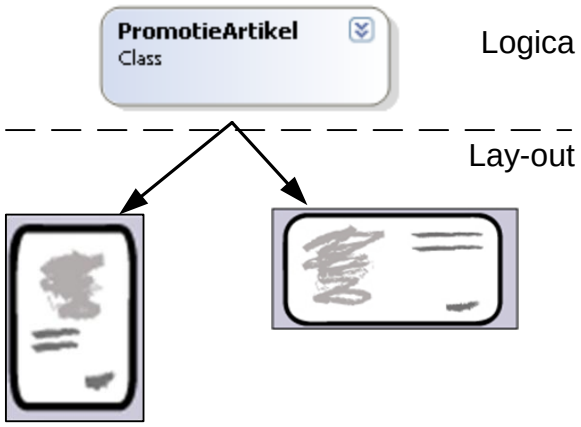
horizontale lay-out:
twee verticale artikels

horizontale lay-out:
drie verticale artikels

verticale lay-out binnen horizontale lay-out:
vier horizontale artikels

Figuur 41 De lay-out van een promotiedia

De promoties worden bijgehouden in een lijst en afhankelijk van hun aantal worden twee soorten lay-out bepaald. Enerzijds de lay-out van de promoties op het scherm. En anderzijds de lay-out van de promotie zelf.
Ik heb dus twee grafische componenten ontwikkeld: een horizontaal artikel en een verticaal artikel. De plaatsing van de promotie-eigenschappen is verschillend maar de componenten gebruiken wel dezelfde klasse om hun gegevens op te halen.



Figuur 42 de artikelklasse met twee verschillende grafische voorstellingen

Zoals u kunt zien in volgende code wordt een casestructuur gebruikt om te bepalen wat de lay-out is van de promoties op het scherm.

```
public class DiaPromotieClass extends Canvas implements IDia      {

        public function toon():void{
                switch(this._promotieList.length){
                        case 1: … break;

                        case 2: …
}
```

3.3.2. Volgende-klant-module

In de analyse heb ik reeds beschreven hoe een winkelbediende het nummer van de volgende klant kan aanpassen op een weegschaal.
De POS-server wordt ingelicht telkens dit nummer wordt aangepast en moet dit doorgeven aan weegschalen in dezelfde groep en aan alle andere schermen waarop het nummertje weergegeven moet worden. Tijdens het overleg met de programmeurs van het POS-systeem werden een aantal mogelijke uitwerkingen voorgesteld.

De POS-server kopieert een bestandje met daarin het nummer van de volgende klant naar elke weegschaal in de groep. Dit systeem is wel vrij belastend voor zowel het netwerk als de weegschalen. Bovendien kunnen er grote vertragingen optreden als één van de weegschalen buiten werking is. Als de server een bestand probeert te kopiëren naar een weegschaal die om een of andere reden niet meer is aangesloten op het netwerk zal er pas een fout optreden na een welbepaalde tijd (het gaat hier over enkele seconden per weegschaal). Dit is niet aanvaardbaar, het nummertje moet zo snel mogelijk op alle schermen weergegeven worden.

Een andere mogelijkheid is dat de software op de weegschalen rechtstreeks leest in de database van de POS-server. De weegschaal moet dan om de seconde het nummertje opvragen en controleren of het gewijzigd is. Deze oplossing lijkt al veel beter dan de vorige maar er is nog één ding over het hoofd gezien. Wat met de andere schermen die het nummertje moeten weergeven? Ze hebben geen toegang tot de database op de POS-server omdat ze in een ander domein zitten. Het nummertje kan dus enkel bezorgd worden door een bestand naar de display units te kopiëren. Maar dit leidt weer tot dezelfde problemen als in het eerste voorstel. Als alternatief kan men op de display units een service draaien die op een bepaalde poort luistert naar het nummertje van de volgende klant.

Het POS-systeem was nog in volle ontwikkeling tijdens mijn stage en de programmeurs hadden nog geen idee op welke manier ze het nummertje zouden doorgeven.

Ik heb er dus voor gekozen om een heel algemene oplossing uit te werken die onafhankelijk is van de manier waarop het nummer van de volgende klant wordt doorgegeven. Een deel van de oplossing werd reeds behandeld bij de bespreking van de applicatie dat het Flexprogramma laadt. We wisselen berichten uit tussen twee verschillende applicaties. Op display units worden berichten uitgewisseld tussen een .NET-applicatie en het Flexprogramma en op weegschalen tussen een Delphi-applicatie en het Flexprogramma.

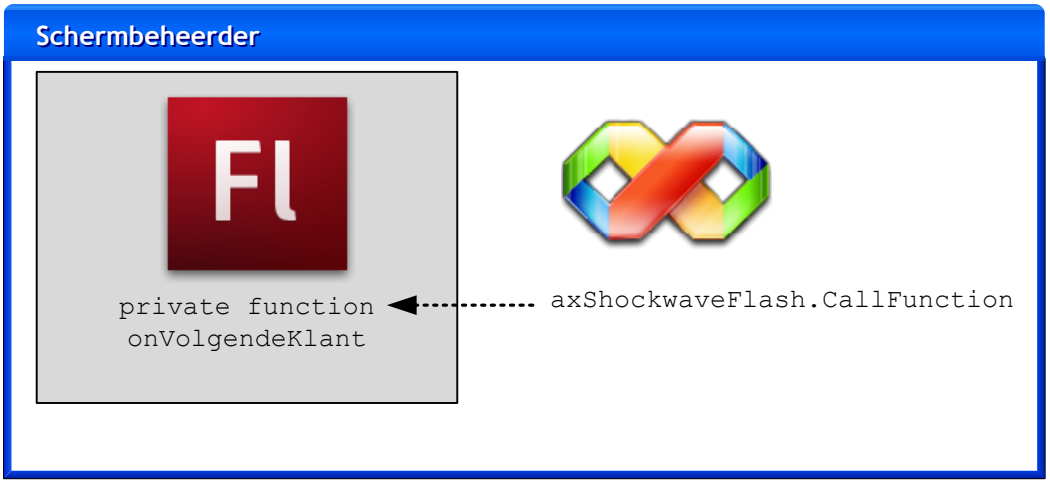
De ExternalInterfaceklasse maakt rechtstreekse communicatie mogelijk tussen ActionScript en de Flash Player. Op weegschalen zal het presentatieprogramma als een component ingebed worden in SPLAN (onderdeel van het POS-systeem dat in Delhpi

geschreven is) en op display units in een stukje .NET-software. In beide talen kan men de Flash Player gebruiken in de vorm van een ActiveX-component.
In Flex heb ik een methode geschreven die opgeroepen moet worden telkens het nummer van de volgende klant verandert. De volgende stap is Flex wijsmaken dat deze methode opgeroepen mag worden door de Flash Player. Dit kan gebeuren tijdens de initialisatie van de applicatie:

```
import flash.external.ExternalInterface;

private function init():void{
	ExternalInterface.addCallback("onVolgendeKlant", onVolgendeKlant);

}

private function onVolgendeKlant(nr:Number):void{
	if (this._ModuleVolgendeKlant != null)
		this._ModuleVolgendeKlant.VolgendeKlantNr = nr;

}
```

De methode *onVolgendeKlant* kan nu door de Flash Player opgeroepen worden. De ActiveX-component heeft een methode *CallFunction* die een String als argument verwacht. Het argument beschrijft (met behulp van XML) de methode die je in Flex wil oproepen: de naam van de methode, het type van elke parameter en een waarde voor elke parameter.

```
_axShockwaveFlash.CallFunction("<invoke
name=\"onVolgendeKlant\"><arguments><number>" + nr.ToString() +
"</number></arguments></invoke>");
```



Figuur 43 Communicatie tussen Flash en .NET

De omgekeerde manier van werken is trouwens ook perfect mogelijk, Flex kan ook methodes oproepen van de software waarin het is ingebed.

Ik denk dat deze oplossing de meest algemene en dynamische is. Dit is niet onbelangrijk aangezien het POS-systeem nog in ontwikkeling is en men nog niet goed weet hoe het nummertje doorgegeven zal worden. Hoe dit uiteindelijk zal gebeuren, doet er niet echt toe. Er zit altijd een interface tussen die de methode *onVolgendeKlant* kan aanroepen.

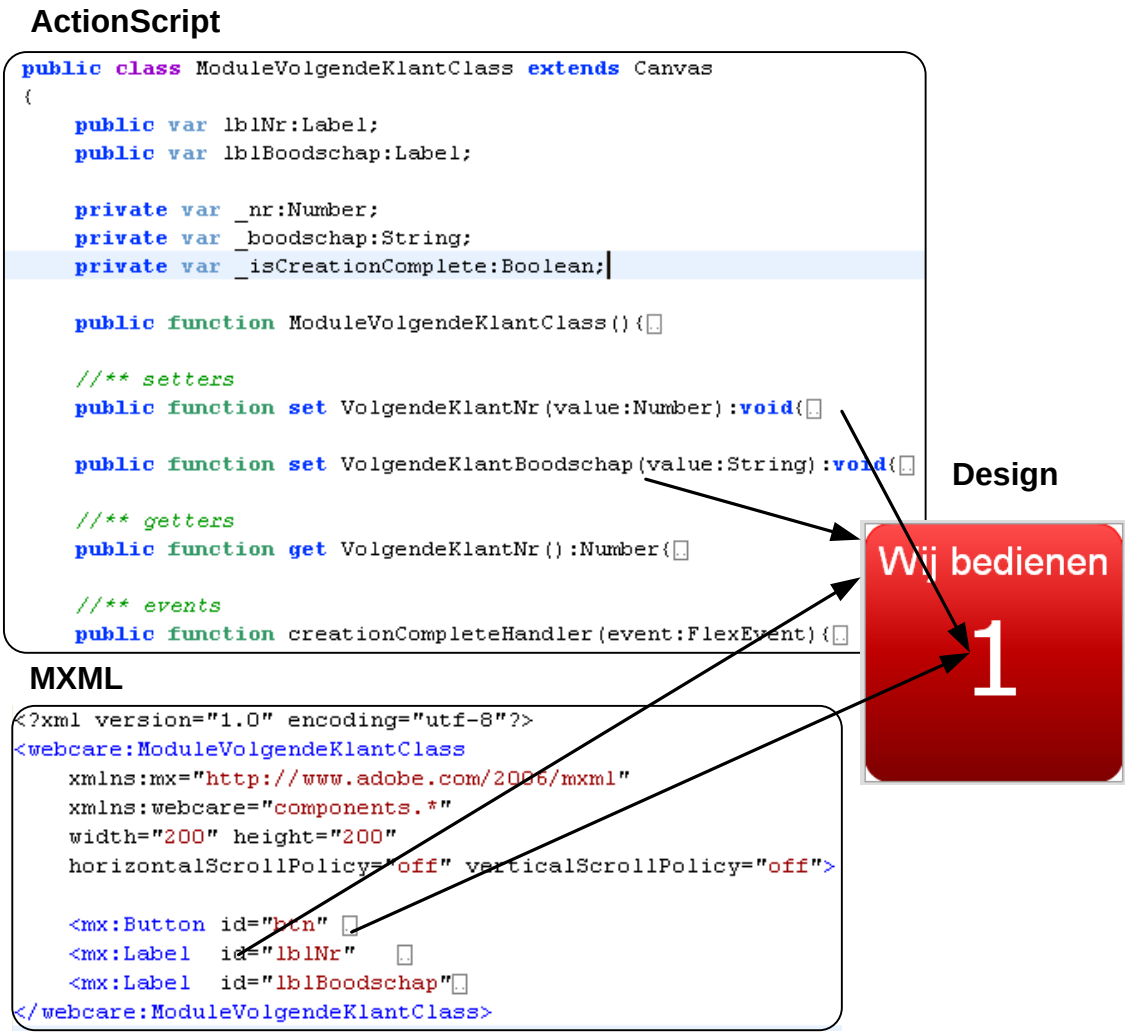
Op dit moment wordt het nummertje doorgegeven op twee manieren afhankelijk van het type scherm waarop het moet verschijnen. Op de weegschalen zal SPLAN het nummertje uit de database lezen en de methode *onVolgendeKlant* oproepen als er een wijziging is. Om het nummer door te geven aan de .NET-applicatie op de display units wordt er een bestand met daarin het nummer naar de display unit gekopieerd. De .NET-applicatie moet dus verwittigd worden als dit bestand overschreven of aangemaakt wordt. Met de klasse *FileSystemWatcher* kan je luisteren naar deze gebeurtenissen en de gewenste acties uitvoeren als ze optreden:

```
_fileSystemWatcher = new FileSystemWatcher();

//enkel sernr.ini bekijken
_fileSystemWatcher.Path = AppDomain.CurrentDomain.BaseDirectory;
_fileSystemWatcher.IncludeSubdirectories = false;
_fileSystemWatcher.Filter = "sernr.ini";

//wakker worden als sernr.ini aangemaakt of overschreven wordt
_fileSystemWatcher.NotifyFilter =
      NotifyFilters.LastWrite | NotifyFilters.CreationTime;

//als ik wakker ben -> _fileSystemWatcher_Changed uitvoeren
_fileSystemWatcher.Changed += new
FileSystemEventHandler(_fileSystemWatcher_Changed);

_fileSystemWatcher.EnableRaisingEvents = true;
```

Er wordt gecontroleerd of het bestand met de naam sernr.ini aangemaakt of overschreven wordt. Als dit gebeurt, wordt de methode *_fileSystemWatcher_Changed* opgeroepen en kan de methode *onVolgendeklant* opgeroepen worden.

```
//het bestand met het nummer van de volgende klant werd aangemaakt
//of overschreven
void _fileSystemWatcher_Changed(object sender, FileSystemEventArgs e)
{
      //code om sernr.ini in te lezen en het nummer uit het bestand
      //te bewaren in de variabele nr

      toonVolgendeKlant(nr);
}

//nummmer van de volgende klant doorgeven
private void toonVolgendeKlant(int nr) {
      _axShockwaveFlash.CallFunction(
            "<invoke name=\"onVolgendeKlant\"><arguments><number>" +
      nr.ToString() + "</number></arguments></invoke>");
}
```

In Flex is het nummer van de volgende klant een module die de beheerder al dan niet kan weergeven op het scherm. Voor elke module heb ik twee klassen geschreven: ModuleVolgendeKlant.mxml en ModuleVolgendeKlantClass.as.
Zoals u kunt zien op volgende figuur wordt MXML gebruikt om de lay-out te bepalen en ActionScript om de applicatielogica vast te leggen.



Figuur 44 Volgende-klant-module is opgebouwd uit ActionScript en MXML

De opbouw van deze Flexcomponent is vrij eenvoudig. De ActionScriptklasse (ModuleVolgendeKlantClass.as) is een grafische component en erft daarom over van de klasse Canvas. Op de component staat een label met een boodschap (bv. 'Wij bedienen') en een label met het nummer van de volgende klant.
In de constructor wordt de waarde van het nummer en de boodschap geïnitialiseerd. Nadien wordt aangegeven dat de methode *creationCompleteHandler* moet uitgevoerd worden als de component gecreëerd en geïnitialiseerd is.

```
public function ModuleVolgendeKlantClass(){
        this._isCreationComplete = false;
        this._boodschap = "";
        this._nr = 0;
        addEventListener(FlexEvent.CREATION_COMPLETE,
creationCompleteHandler);
}
```

Als de methode *creationCompleteHandler* opgeroepen wordt dan ben ik zeker dat de twee labels gecreëerd zijn en kan ik de labels voorzien van tekst. Verder wordt hier ook de grootte van de tekst bepaald. Dit is noodzakelijk omdat Flex de fontgrootte niet zelf aanpast als de grootte van het label verandert. Het was niet eenvoudig om een oplossing

te vinden voor dit probleem maar het is uiteindelijk toch gelukt.
Met de attributen *scaleX*[8] en *scaleY* kan de tekst groter of kleiner gemaakt worden. Eerst heb ik de volgende-klant-module een standaard grootte gegeven van 200 op 200 pixels. In Flex Builder heb ik de twee labels van willekeurige tekst voorzien en de fontgrootte aangepast zodat de tekst niet te groot of niet te klein was. Stel dat deze fontgrootte 12 is en de module de standaardgrootte krijgt van 200 op 200 pixels dan zullen de attributen *scaleX* en *scaleY* beide de waarde 1 hebben. Als de beheerder de module een grootte geeft van 400 op 400 pixels dan hebben zowel *scaleX* als *scaleY* de waarde 2. De tekst zal dus dubbel zo groot getoond worden.

Op het einde van de constructor krijgt de variabele *_isCreationComplete* de waarde true zodat ik in de setters weet of ik de labels van tekst mag voorzien. Het is namelijk mogelijk dat een setter uitgevoerd wordt voordat de creatie van de component voltooid is.

```
public function creationCompleteHandler(event:FlexEvent){
        lblNr.scaleX = this.height / 200;
        lblNr.scaleY = this.height / 200;
        lblBoodschap.scaleX = this.height/200;
        lblBoodschap.scaleY = this.height/200;

        if(isNaN(this._nr)){
                lblNr.text = "";
        }else{
                lblNr.text = this._nr.toString();
        }
        lblBoodschap.text = this._boodschap;
        this._isCreationComplete = true;
}

public function set VolgendeKlantNr(value:Number):void{
        this._nr = value;
        if(this._isCreationComplete)
                lblNr.text = this._nr.toString();
}
```

We hebben gezien hoe de volgende-klant-module geïnitialiseerd wordt. Nu moeten we zorgen dat het nummer ook gewijzigd kan worden nadat de module geladen is. De ActionScriptklasse heeft een eigenschap *VolgendeKlantNr* die gedefinieerd wordt door twee methodes: *set VolgendeKlantNr* (ook wel setter genoemd) en *get VolgendeKlantNr* (ook wel getter genoemd). Het is niet nodig om zowel een setter en een getter te definiëren voor een eigenschap. De eigenschap *VolgendeKlantBoodschap* wordt bijvoorbeeld enkel gedefinieerd door een getter. Men gebruikt vaak de term read-only property voor deze soort eigenschapen waarvan men enkel de waarde kan opvragen.

De volgende-klant-module bestaat uit een MXML-component die erft van een ActionScriptklasse. De lay-out en de applicatielogica van de module kunnen dus onafhankelijk van elkaar aangepast worden. Dit principe wordt dus ook bij elke module toegepast.

[8] The horizontal scaling factor for this object. The value is a Number where 1.0 means the object isn't scaled horziontally, 2.0 means that it is stretched to twice its normal width, and 0.5 means that it is compressed to half its normal width. (Flex Language Reference)

3.3.3. Logomodule

We hebben reeds twee verschillende modules gezien: de presentatiemodule en de volgende-klant-module. In dit systeem heb ik nog een derde module voorzien die de statische logomodule van de vorige versie vervangt.

De beheerder kan in het schermontwerp aangeven dat het logo van de winkel altijd zichtbaar moet zijn. Hij kan ook aangeven hoe groot de balk moet zijn waarin het logo getoond wordt. Afhankelijk van de afmetingen kan de status van de balk beschreven worden als staand of liggend. Als de balk liggend is dan wordt het logo in de linkerhoek getoond anders in de bovenhoek.
Naast de afmetingen van de logobalk kan de beheerder de logogrootte bepalen: hoeveel procent van de grootte van de balk het logo in beslag mag nemen. De module zorgt er dan voor dat het logo in het midden van de balk getoond wordt. Er is dus evenveel witruimte links, boven en onder als de status liggend is en evenveel witruimte boven, links en rechts als de status staand is.

Figuur 45 De logomodule

In bovenstaande figuur zien we het logo van de hogeschool in een liggende balk. Het logo wordt dus links uitgelijnd en er is evenveel witruimte links, boven en onder. Het moment waarop de afmetingen en de positie van het logo bepaald worden is cruciaal. We moeten wachten tot Flex de afbeelding volledig geladen heeft. Dit weten we pas zeker als de gebeurtenis *COMPLETE* wordt opgeroepen. Indien we de eigenschappen van de afbeelding opvragen voor ze geladen is, krijgen we geen fout maar wel de verkeerde waarden.

```
private function creationCompleteHandler(event:FlexEvent):void{
        imgLogo.source = this._logoPad;

        if (this.width > this.height)
                this._landscape = true;

        //wachten tot de afbeelding geladen is

        imgLogo.addEventListener(Event.COMPLETE,  logoUpdateCompleteHandler);
}

private function logoUpdateCompleteHandler(event:Event){

        //code om afmeting en positie van logo te bepalen
}
```

3.4. Netwerkstructuur

De beheerder kan in de centrale presentaties aanmaken en exporteren. Op de display units en weegschalen in de winkels draait een programma dat deze presentaties kan weergeven. Alle data moet dus op een of andere manier in de eerste plaats naar de juiste winkel en in de tweede plaats naar het juiste scherm gekopieerd worden.

In de presentatiebeheerder worden de schermen aangeduid met een logische naam en niet met de naam of het IP-adres van de display unit die gekoppeld is aan het scherm. Er is natuurlijk wel een koppeling nodig tussen de logische naam en het adres van de display unit. Deze koppeling wordt ingesteld door een installateur, hij is de enige die het adres van de display unit te zien krijgt.
Als de presentatie toegewezen is aan een scherm moet alle data gekopieerd worden naar een FTP-server. Elke winkel heeft een private map op deze FTP-server waarin alle presentaties en schermontwerpen voor één bepaalde winkel staan. Er bevindt zich ook een configuratiebestand in deze map die ingelezen wordt door de service in de winkel. Naast de private map is er ook één map die toegankelijk is voor alle winkels. Deze bevat alle mediabestanden die gebruikt worden in presentaties die reeds geëxporteerd zijn.

Het proces om de presentaties (die aangemaakt zijn op de centrale) te tonen op de schermen kan als volgt beschreven worden:

- De beheerder maakt presentaties aan.
- De presentaties en mediabestanden worden bewaard in een centrale gedeelde map.
- De beheerder exporteert presentaties.
- De presentaties en mediabestanden worden gekopieerd naar de FTP-server (deze is bereikbaar via een gedeelde map).
- De winkel meldt zich aan op de FTP-server en komt terecht in zijn private map.
- De winkel downloadt het configuratiebestand en actualiseert de lokale versie.
- De winkel downloadt het overzicht van de schermontwerpen.
- De winkel downloadt alle schermontwerpen die voorkomen in het overzicht.
- De winkel downloadt het overzicht van de presentaties.
- De winkel downloadt alle presentaties die voorkomen in het overzicht.
- De winkel meldt zich aan op de FTP-server met de gegevens die in het configuratiebestand staan en komt terecht in de map met de mediabestanden.
- De winkel downloadt alle mediabestanden die in de presentaties gebruikt worden. Enkel nieuwe of gewijzigde bestanden worden gedownload.
- De winkel brengt de schermen op de hoogte dat er een update zal plaatsvinden.
- De winkel kopieert de presentaties en mediabestanden naar de schermen.
  Enkel de presentaties die bestemd zijn voor het scherm worden gekopieerd.
  Oude bestanden worden verwijderd.
  De status van de overdracht wordt bewaard in een logboek.
- De winkel meldt de schermen dat ze opnieuw presentaties kunnen afspelen.

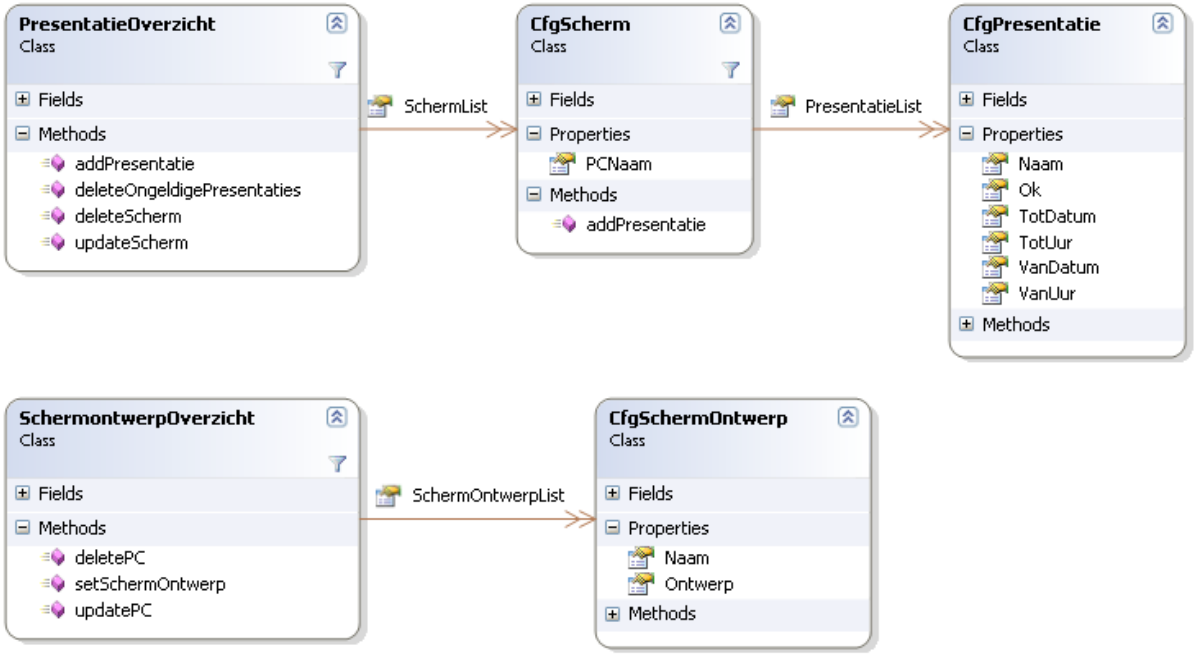
Zowel op het niveau van de centrale gedeelde map, de FTP-server, de winkel en de display units wordt dezelfde mappenstructuur gebruikt.
In de eerste tabel kan u de mapstructuur zien van de gedeelde map en de FTP-server in de centrale. De tweede tabel geeft een overzicht van de mappen op de display units en de server in de winkel.

Centrale	
Centrale gedeelde map	**FTP-server**
\share\media\ \share\media\pictures \share\media\pictures\products\ \share\media\pictures\photos\ \share\media\movies\ (Mediabestanden van **alle** presentaties.)	\ftpshare\media\ \ftpshare\media\pictures \ftpshare\media\pictures\products\ \ftpshare\media\pictures\photos\ \ftpshare\media\movies\ (Mediabestanden gebruikt in presentaties die **reeds geëxporteerd** zijn.) \ftpshare\presentations\ (Bevat per presentatie een overzicht van de mediabestanden die erin gebruikt worden. De service moet dan niet meer uitzoeken welke bestanden waar gebruikt worden.) Per winkel is er een map (mapnaam = naam winkel).
	\ftpshare\winkel\pres-summary.xml (Bevat een overzicht van alle presentaties die afgespeeld moeten worden in de winkel.)
	\ftpshare\winkel\module-summary.xml (Bevat een overzicht van alle schermontwerpen die gebruikt worden in de winkel.)
\share\presentations\	\ftpshare\winkel\presentations\ (Bevat per presentatie een XML-bestand.)
\share\modules\	\ftpshare\winkel\modules\ (Bevat per schermontwerp een XML-bestand.)
	\ftpshare\winkel\communication\ \ftpshare\winkel\communication\config.xml (Configuratiebestand dat ingelezen wordt door de service in de winkel.)
	\ftpshare\winkel\log\ (Bevat logbestanden)

Winkel	**Display Unit**
\setup\media\ \setup\media\pictures \setup\media\pictures\products\ \setup\media\pictures\photos\ \setup\media\movies\ (Mediabestanden gebruikt in presentaties bestemd voor **deze winkel**.)	\Flex\media\ \Flex\media\pictures \Flex \media\pictures\products\ \Flex\media\pictures\photos\ \Flex\media\movies\ (Media bestanden gebruikt in presentaties bestemd voor **dit scherm**)
\setup\pres-summary.xml (Bevat een overzicht van alle presentaties die afgespeeld moeten worden in deze winkel.)	\Flex\pres-summary.xml
\setup\module-summary.xml (Bevat een overzicht van alle schermontwerpen die gebruikt worden in deze winkel.)	\Flex\module-summary.xml
\setup\presentations\ (Bevat alle presentaties bestemd voor deze winkel.)	\Flex\presentations\ (Bevat alle presentaties bestemd voor het scherm waar de display unit aan gekoppeld is.)
\setup\modules\ (Bevat alle schermontwerpen bestemd voor de schermen in deze winkel.)	\Flex\modules\ (Bevat het schermontwerp dat bestemd is voor het scherm waaraan de display unit gekoppeld is.)
\setup\communication\ \setup\communication\config.xml (Configuratiebestand dat ingelezen wordt door de service in deze winkel.)	

De bestanden pres-summary.xml en module-summary.xml zorgen beiden voor een koppeling van respectievelijk een presentatie en een schermontwerp met een scherm. Aanpassing of creatie van deze bestanden gebeurt als de beheerder respectievelijk een presentatie of een schermontwerp exporteert. In de presentatiebeheerder zijn deze configuratiebestanden gedefinieerd als klassen. Als het configuratiebestand reeds bestaat, wordt het omgezet naar een object. Indien dit niet het geval is, wordt een nieuw object aangemaakt. Als de configuratie gewijzigd is, wordt het object opnieuw omgezet naar XML. Door het gebruik van objecten moet ik geen aparte methodes schrijven om

knopen uit een XML-bestand te overlopen en te wijzigen. Bovendien verloopt de serialisatie en deserialisatie volledig automatisch. Onderstaand klassendiagram geeft een overzicht van de configuratieobjecten.



Figuur 46 Klassendiagram: PresentatieOverzicht en SchermontwerpOverzicht

Een presentatieoverzicht bevat alle schermen van een bepaalde winkel. Per scherm is er een overzicht van de presentaties die erop getoond moeten worden.
Een schermontwerpoverzicht bevat een lijst met schermen en de naam van het schermontwerp dat eraan gekoppeld is. Na serialisatie van deze objecten krijgen we volgende XML-structuur:

```
<?xml version="1.0" encoding="utf-8" ?>
- <PresentatieOverzicht
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <Schermen>
    - <Scherm PCNaam="192.168.91.4">
      - <Presentaties>
          <Presentatie Naam="KRISTOF" VanDatum="2008-
            05-14T00:00:00+02:00" TotDatum="2008-05-
            15T00:00:00" VanUur="2008-05-14T09:36:42"
            TotUur="2008-05-14T12:36:42" />
          <Presentatie Naam="oo" VanDatum="2008-05-
            14T00:00:00+02:00" TotDatum="2008-05-
            15T00:00:00" VanUur="2008-05-
            14T10:40:02.53125+02:00" TotUur="2008-05-
            14T15:45:02" />
        </Presentaties>
      </Scherm>
    + <Scherm PCNaam="192.168.91.7">
    + <Scherm PCNaam="ggtest">
    + <Scherm PCNaam="192.168.91.3">
    </Schermen>
</PresentatieOverzicht>
```

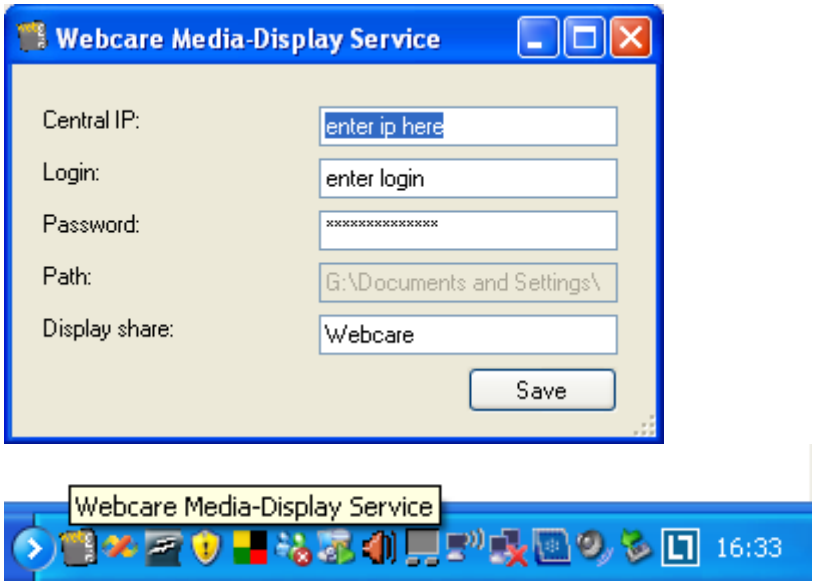
Figuur 47 pres-summary.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <SchermontwerpOverzicht
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Scherm Naam="192.168.91.4" Ontwerp="1280x800.xml" />
    <Scherm Naam="192.168.91.7" Ontwerp="169.xml" />
    <Scherm Naam="192.168.91.3" Ontwerp="1280x800.xml" />
    <Scherm Naam="ggtest" Ontwerp="1280x800.xml" />
</SchermontwerpOverzicht>
```

Figuur 48 module-summary.xml

3.5. De winkelservice

Presentaties downloaden en verspreiden naar de juiste schermen is de taak van de service die in elke winkel draait. De configuratie van deze service staat beschreven in een XML-bestand. Om de configuratie van de service te vereenvoudigen heb ik een programma geschreven dat zichtbaar is in de taakbalk.



Figuur 49 Configuratie van de service

De beheerder kan de volgende instellingen wijzigen:

- Het IP van de centrale.
- Gebruikersnaam en wachtwoord die toegang geven tot de persoonlijke map van de winkel op de FTP-server van de centrale.
- De map waarin alle bestanden gedownload moeten worden.
- De naam van de gedeelde map op alle display units.

De service zelf is een speciaal soort Windowsprogramma dat een aantal taken uitvoert waarvoor geen interactie met een gebruiker nodig is.

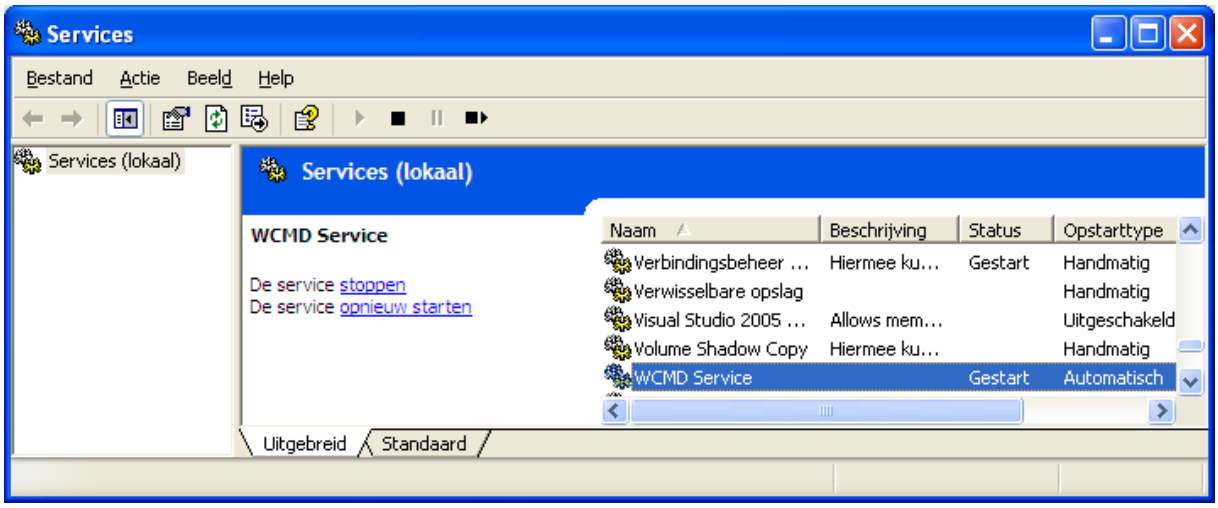
In Visual Studio 2005 is er een speciaal projecttype om een zogenaamde Windows Service te ontwikkelen.
Als we een project van het type Windows Service creëren dan worden er twee klassen aangemaakt: een *Service* en een *ProjectInstaller*. De *Service* is een klasse die afgeleid is van de klasse *ServiceBase*. Met behulp van de methodes *OnStart* en *OnStop* kunnen we definiëren wat er moet gebeuren als de service gestart of gestopt wordt.

```
public partial class WCMDService : ServiceBase{
    public WCMDService()

    protected override void OnStart(string[] args)
    protected override void OnStop()
    …
}
```

De ProjectInstaller definieert eigenschappen zoals de naam en de omschrijving van de service eenmaal die geïnstalleerd is. Om de service te installeren, starten we eerst de Visual Studio .NET Command Prompt, bladeren naar de map waarin het uitvoerbaar bestand van de service staat en voeren volgend commando uit:

```
installutil WCMDService.exe.
```

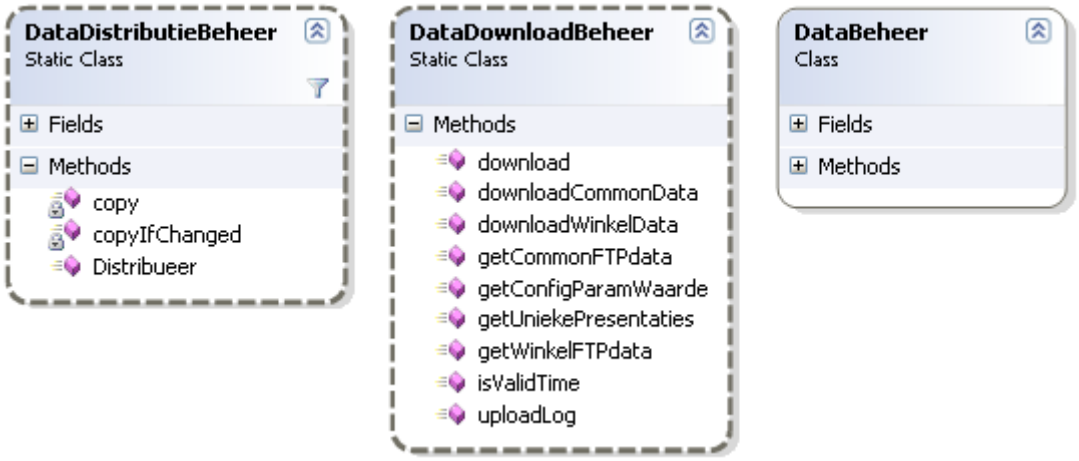
We zien dat de service nu actief is in het servicebeheer van Windows. Het is natuurlijk ook mogelijk om een *Setup Project* te maken zodat iemand zonder Visual Studio de service ook kan installeren.



Figuur 50 De Webcare mediadisplay service is gestart

De service moet maar één taak uitvoeren: nieuwe of gewijzigde presentaties downloaden van de centrale. Er zijn wel twee mogelijke manieren om deze taak te starten: als een bepaald aantal minuten (beschreven in het configuratiebestand) verstreken zijn zal de service automatisch controleren of er presentaties gedownload moeten worden. Maar de centrale kan elke winkel ook de opdracht geven om op een bepaald moment de gewijzigde en nieuwe presentaties onmiddellijk te downloaden. Dit kan handig zijn als er bijvoorbeeld een spellingfout verbeterd werd.

De taak bestaat dus uit twee grote delen: downloaden en verspreiden. Ik heb voor elke deeltaak een statische klasse geschreven. DataDownloadBeheer is verantwoordelijk voor het downloaden van presentaties en DataDistributieBeheer is verantwoordelijk voor het verspreiden van de presentaties naar de juiste display units. Verder is er ook nog een klasse DataBeheer die ondermeer aangesproken kan worden om presentaties, overzichten en configuratiebestanden uit te lezen. Deze klasse zorgt er ook voor dat alles in de juiste map terecht komt. De belangrijkste klassen van de service zijn weergegeven in het volgende klassendiagram:



Figuur 51 Klassendiagram: Service

Zoals u kunt zien in volgende code worden eerst alle presentaties (en bijhorende mediabestanden) gedownload. En pas als dat gelukt is, krijgen alle schermen de juiste presentaties.

```
if (DataDownloadBeheer.download())
{
    DataDistributieBeheer.distribueer();
}
```

De methode die verantwoordelijk is voor het downloaden ziet er (ingekort) als volgt uit:

```
public static bool download()
{
    bool gelukt = false;
    if (downloadWinkelData())
    {
        if (downloadCommonData())
        {
            gelukt = true;
        }

    }
    return gelukt;
}
```

Als de XML-bestanden (presentaties en overzichten) succesvol gedownload zijn dan kunnen de mediabestanden gedownload worden. Enkel als ook deze bestanden zonder problemen gedownload zijn, is het downloaden gelukt en kunnen de bestanden naar de display units gekopieerd worden.

In de vorige versie van het systeem (ontwikkeld door Sien D'Hooghe in VB.NET) werd een speciale FTP-klasse gebruikt om de bestanden te downloaden. Ik heb samen met Sien een klasse FTPBeheer geschreven die deze VB.NET-dll aanspreekt. Tijdens de tests bleek dat de overdracht veel te lang duurde. Een andere collega, Gunther Govaerts, stelde ons daarom voor om het pakket edtFTPnet[9] te gebruiken. Aangezien we enkel

[9] http://www.enterprisedt.com

wijzigingen moesten aanbrengen in de klasse FTPBeheer vormde het gebruik van deze nieuwe bibliotheek geen enkel probleem.

Na de downloadprocedure moet de service wel weten naar welke schermen welke presentaties gekopieerd moeten worden. Al deze informatie staat beschreven in het presentatieoverzicht, een XML-bestand dat met behulp van de methode openPresentatieOverzicht van de klasse DataBeheer omgezet wordt naar een object. In volgend, heel sterk ingekort, stukje code kan u zien hoe dit in zijn werk gaat:

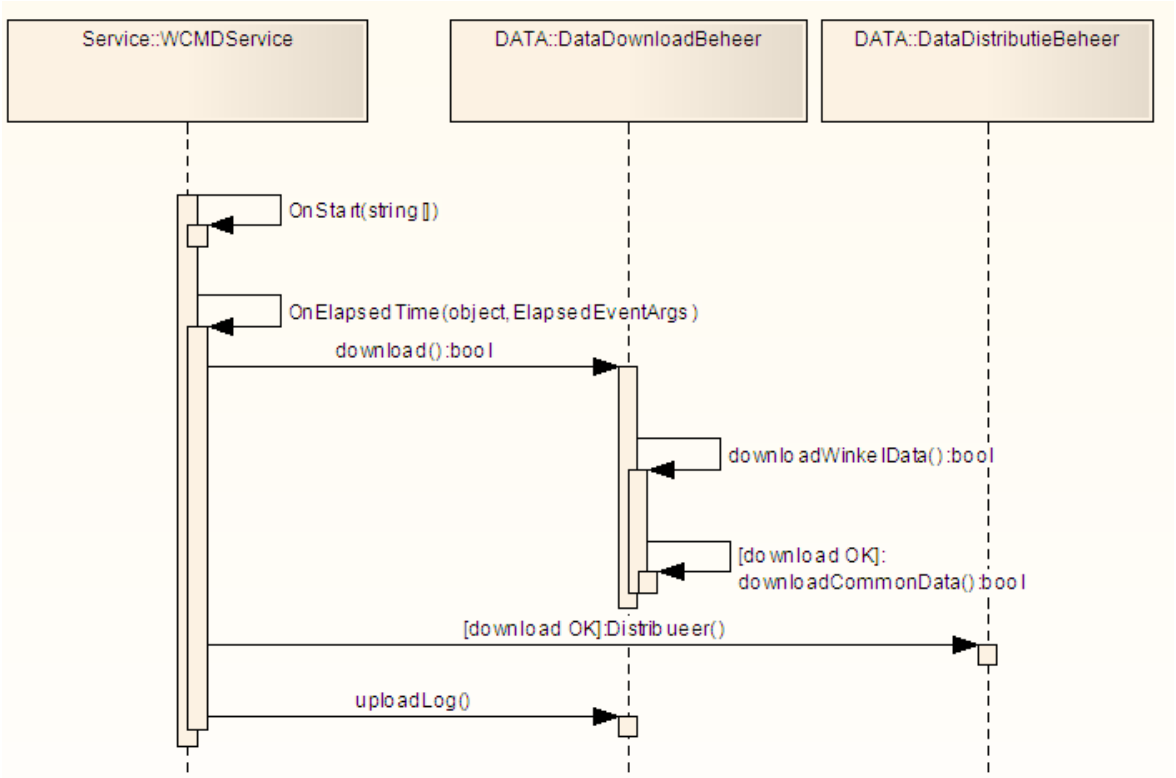
```
//presentatieoverzicht omzetten naar een object
presOverzicht = DataBeheer.getInstance().openPresentatieOverzicht();

//Alle display units overlopen
foreach (CfgScherm scherm in presOverzicht.SchermList){
    try{
        //Controleren of de display unit bereikbaar is
        try{
            DataBeheer.getInstance().maakMappenScherm(scherm.PCNaam);
        }
        catch{
            throw new Exception(scherm.PCNaam + " is not reachable");
        }

        //Alle presentaties bestemd voor de display unit overlopen
        foreach (CfgPresentatie pres in scherm.PresentatieList)
        {
            //Alle bestanden kopiëren
            foreach (String bestand in
            DataBeheer.getInstance().openPresMediaLijst(pres.Naam)
            {  //…   }
        }
}
```

Bovenstaande code is sterk vereenvoudigd, daarom zal ik het distributieproces in detail beschrijven.
Nadat het presentatieoverzicht omgezet is naar een object worden alle display units overlopen. Op elke display unit moeten een aantal mappen aangemaakt worden, als dit mislukt, is de display unit niet bereikbaar en wordt het proces afgebroken. Vervolgens wordt er een bestand aangemaakt op de display unit met de naam update.txt. Het programma dat de presentaties toont controleert om de 5 seconden of dit bestand aanwezig is. Eventuele actieve presentaties worden gestopt en er wordt gewacht tot het updatebestand verwijderd is. Een alternatief voor dit updatebestand is het aanroepen van een methode zoals ik gedaan heb bij het nummer van de volgende klant. Helaas kon dit niet doorgaan omdat het POS-systeem dan te veel gewijzigd zou moeten worden.
Het systeem weet nu dat er een update plaatsvindt en kan het distributieproces hervatten. Er wordt eerst een back-up gemaakt van het presentatieoverzicht van de display unit. Zo weten we tenminste nog welke presentaties op een bepaald scherm getoond werden als er tijdens de distributie iets mis zou gaan. Dan wordt er een configuratiebestand gekopieerd dat ondermeer de naam van het scherm bevat. Het programma dat de presentaties toont kan het IP of de computernaam namelijk niet zelf opvragen. Vervolgens worden alle mediabestanden van de presentaties gekopieerd. Pas als alle mediabestanden zonder problemen gekopieerd zijn, worden de presentaties, het presentatieoverzicht, de schermontwerpen en het schermontwerpoverzicht gekopieerd. Zo zijn we zeker dat alle mediabestanden die in een presentatie gebruikt worden ook echt aanwezig zijn. Moest er toch iets verkeerd gaan tijdens de distributie dan is het presentatieoverzicht nog niet overschreven en worden gewoon de oude presentaties

getoond. Nu we zeker zijn dat de nieuwe/gewijzigde presentaties op de display unit staan, kunnen we alle bestanden verwijderen die overbodig zijn. De distributie is afgelopen en het updatebestand mag dus verwijderd worden. Ook als de distributie mislukt, zal de service proberen om het updatebestand te verwijderen zodat toch nog de oude presentaties getoond kunnen worden. De *try* en *catch* rond deze code zorgt ervoor dat de service niet crasht maar dat de fout weggeschreven wordt naar een logboek. De *finally* clausule zal steeds proberen het updatebestand te verwijderen. Als alles zonder problemen verlopen is, weten we dat de presentaties in het presentatieoverzicht getoond zullen worden. Indien er iets verkeerd gelopen is dan zullen de presentaties getoond worden uit het oude presentatieoverzicht.

Bovenstaande paragraaf beschrijft de procedure om één display unit te voorzien van nieuwe presentaties. We herhalen deze procedure voor alle schermen en houden telkens bij welke presentaties getoond worden. Als alle display units voorzien zijn van nieuwe presentaties wordt er een soort samenvatting gemaakt die aangeeft welke presentaties op welke schermen getoond worden. Deze samenvatting wordt nadien geüpload naar de persoonlijke map van de winkel.

Het vereenvoudigd sequentiediagram verduidelijkt de opeenvolgende stappen:



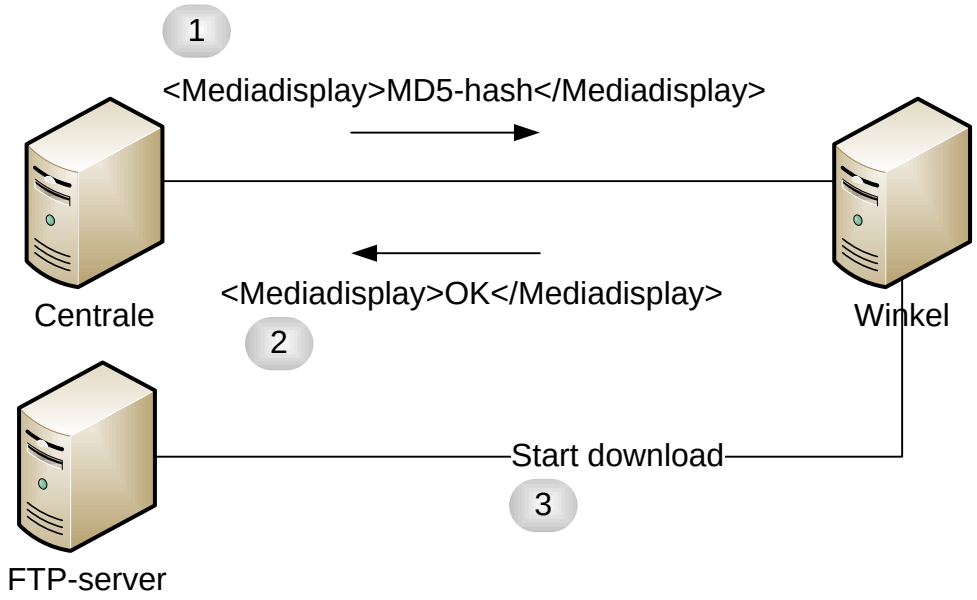
Figuur 52 Sequentiediagram: presentaties downloaden en distribueren

We weten nu hoe de service de presentaties downloadt en kopieert naar de juiste schermen telkens een bepaald aantal minuten verstreken zijn. Nu zullen we de tweede manier bekijken om de downloadprocedure te starten.
De beheerder kan in de centrale de opdracht geven om op een bepaald ogenblik de downloadprocedure onmiddellijk te starten. Om dit mogelijk te maken heb ik een TCP-server ontwikkeld en deze ingebouwd in de service. De TCP-server luistert naar aanvragen van de centrale in een afzonderlijke thread. Als de centrale verbinding maakt met de TCP-server wordt er gewacht tot een boodschap in een bepaald formaat binnenkomt. Ik zou natuurlijk ook de downloadprocedure kunnen starten op het moment dat iemand een verbinding probeert te maken maar dat is niet echt veilig. Elke hacker die een verbinding kan maken met de server in de winkel zou dan de downloadprocedure

kunnen starten. Het is dus noodzakelijk om te valideren dat het weldegelijk de centrale is die een verbinding probeert te maken. Ik wacht dus tot er een boodschap binnenkomt die het volgende formaat heeft: *<Mediadisplay>MD5-hash</Mediadisplay>*. De MD5-hash moet iets zijn dat zowel de centrale als de winkel afzonderlijk kunnen berekenen. Zowel de centrale als de winkel kennen de gebruikersnaam en het wachtwoord van de persoonlijke map van de winkel op de FTP-server. Ik maak dus één lange tekenreeks die de gebruikersnaam en het wachtwoord bevat en bereken hiervan de MD5-hash. De TCP-server wacht op een boodschap in het juiste formaat en controleert nadien of de MD5-hash geldig is. Enkel geldige aanvragen hebben dus tot gevolg dat de service de downloadprocedure zal starten. Volgende code geeft aan hoe dit juist werkt:

```
//Het is een boodschap uit ons protocol dwz:
<MediaDisplay>Boodschap</MediaDisplay>
if (totalMessage.StartsWith(STARTTAG) &&
totalMessage.EndsWith(ENDTAG))
{

    //login en wachtwoord uit configuratiebestand
    String login =
DataBeheer.getInstance().getServiceconfig().WinkelLogin;
    String password =
DataBeheer.getInstance().getServiceconfig().WinkelPassword;

    //MD5-hash parsen uit <MediaDisplay>MD5-hash</MediaDisplay>
    String ontvangen_md5hash =
totalMessage.Substring(STARTTAG.Length, totalMessage.Length -
STARTTAG.Length - ENDTAG.Length);

    //controleren of login en wachtwoord juist zijn
    if (ontvangen_md5hash == EncodePassword(login + password))
    {
        String strCommand = STARTTAG + "OK" + ENDTAG;
        byte[] buffer = encoder.GetBytes(strCommand);

        //Alles is inorde dus we sturen OK naar de centrale
        clientStream.Write(buffer, 0, buffer.Length);
        clientStream.Flush();
        einde = true;
        tryToDownload = true;

    }
}
…
```

De figuur op de volgende pagina toont de verschillende stappen.



Figuur 53 Communicatieprotocol dat gebruikt wordt tussen centrale en winkel

De timer en de TCP-server draaien elk in een afzonderlijke thread. Het is dus belangrijk om op te merken da de downloadprocedure *gelockt* is en dus maar door één thread tegelijkertijd uitgevoerd kan worden.

De taak die de service moet uitvoeren is vrij complex en er kan veel misgaan. Het was dus van het grootste belang dat eventuele fouten bijgehouden werden.
Ik heb besloten om een logboek bij te houden met alle acties die de service onderneemt. In dit tekstbestand staat vermeld op welke poort de service luistert, wanneer de service begint met downloaden, welke bestanden gedownload en gekopieerd worden en wat er allemaal misgegaan is.
Stel dat een display unit uit het netwerk gehaald wordt dan kunnen we volgende melding lezen in het logboek: 'computernaam not reachable'. Als de beheerder merkt dat er op een bepaald scherm niks meer verschijnt, kan hij dus steeds het logboek raadplegen om de oorzaak van het probleem te lokaliseren. Hieronder kunt u een voorbeeld van zo'n logboek bekijken:

```
14/05/2008 13:53:46 #Info    #   Service started
14/05/2008 13:53:46 #Info    #   Listening at port: 3000
14/05/2008 13:53:46 #Info    #   Timer interval set: 1,10355 min
14/05/2008 13:53:46 #Push    #   Listen for clients
14/05/2008 13:53:46 #Info    #   OnStart
14/05/2008 13:54:52 #Download#   [Service] Download is possible -> Check if
time is valid ...
14/05/2008 13:54:52 #Download#   [Service] Time valid between - starthour:
1/01/1900 0:00:00 - stophour 1/01/1900 23:59:00
14/05/2008 13:54:52 #Download#   [Service] Time is valid -> Start download
...
14/05/2008 13:54:52 #Download#   FTP Connect to 192.168.91.1 (shop1)
14/05/2008 13:54:52 #Download#   Start downloading shopdata
14/05/2008 13:54:53 #Download#   Downloading: communication\config.xml
14/05/2008 13:54:53 #Download#   Downloading: module-summary.xml
14/05/2008 13:54:53 #Download#   Downloading: modules\1280x800.xml
14/05/2008 13:54:53 #Download#   Downloading: modules\1280x800b.xml
14/05/2008 13:54:54 #Download#   Downloading: modules\169.xml
14/05/2008 13:54:54 #Download#   Downloading: modules\martijn.xml
```

```
14/05/2008 13:54:54 #Download#   Downloading: modules\q.xml
14/05/2008 13:54:54 #Download#   Downloading: modules\weegschaal.xml
14/05/2008 13:54:55 #Download#   Downloading: presentations\benx.xml
14/05/2008 13:54:57 #Download#   Downloading: pres-summary.xml
14/05/2008 13:54:57 #Download#   Finished downloading shopdata
14/05/2008 13:54:57 #Download#   FTP Connect to 192.168.91.1 (common)
14/05/2008 13:54:57 #Download#   Start downloading mediafiles
14/05/2008 13:54:58 #Download#   Downloading: presentations\_benx.xml
14/05/2008 13:54:59 #Download#   Downloading: media\movies\djpbin-
ben.x_1.flv
14/05/2008 13:58:01 #Download#   Finished downloading shopdata
14/05/2008 13:58:01 #Download#   [Service] Download finished
14/05/2008 13:58:06 #Copy    #   start
14/05/2008 13:58:12 #Copy    #   Start copy to 192.168.91.7
14/05/2008 13:58:17 #Copy    #   Start copy logo
14/05/2008 13:58:17 #Copy    #   Copy (filechanged) not needed:
\\192.168.91.7\Webcare\logo.jpg already exists
14/05/2008 14:00:17 #Copy    #   Copy OK:
D:\Webcare\Service\common\media\movies\djpbin-ben.x_1.flv ->
\\192.168.91.7\Webcare\common\media\movies\djpbin-ben.x_1.flv
14/05/2008 14:00:17 #Copy    #   Copy OK:
D:\Webcare\Service\presentations\benx.xml ->
\\192.168.91.7\Webcare\presentations\benx.xml
14/05/2008 14:00:17 #Copy    #   Copy OK:
D:\Webcare\Service\modules\1280x800.xml ->
\\192.168.91.7\Webcare\modules\1280x800.xml
14/05/2008 14:00:17 #Copy    #   Copy OK: D:\Webcare\Service\module-
summary.xml -> \\192.168.91.7\Webcare\module-summary.xml
14/05/2008 14:00:17 #Copy    #   Copy OK: D:\Webcare\Service\pres-
summary.xml -> \\192.168.91.7\Webcare\pres-summary.xml
14/05/2008 14:00:17 #Copy    #   Copy OK: D:\Webcare\Service\pres-
summary.xml -> D:\Webcare\Service\log\192.168.91.7.xml
14/05/2008 14:00:17 #Copy    #   End copy to 192.168.91.7
14/05/2008 14:00:17 #Copy    #   Start copy to ggtest
14/05/2008 14:00:20 #Error   #   Failed copy to ggtest: The network path
was not found.
14/05/2008 14:00:20 #Copy    #   Start copy to 192.168.91.3
14/05/2008 14:00:41 #Error   #   Failed copy to 192.168.91.3: The network
path was not found.
14/05/2008 14:00:41 #Copy    #   Start copy to ggtest
14/05/2008 14:00:43 #Error   #   Failed copy to ggtest: The network path
was not found.
14/05/2008 14:00:43 #Copy    #   Start copy to 192.168.91.3
14/05/2008 14:00:43 #Error   #   Failed copy to 192.168.91.3: The network
path was not found.
14/05/2008 14:00:43 #Copy    #   Start copy to ggtest
14/05/2008 14:00:43 #Error   #   Failed copy to ggtest: The network path
was not found.
14/05/2008 14:00:43 #Copy    #   Start copy to 192.168.91.3
14/05/2008 14:00:43 #Error   #   Failed copy to 192.168.91.3: The network
path was not found.
14/05/2008 14:00:43 #Copy    #   stop
14/05/2008 14:00:43 #Info    #   Timer interval set: 1,09791666666667 min
14/05/2008 14:00:43 #Download#   FTP Connect to 192.168.91.1 (shop1)
14/05/2008 14:00:43 #Download#   Log upload OK
```

Laten we dit logboek even analyseren. De service wordt gestart en ongeveer een minuut later gaat de timer af en wordt er gecontroleerd of er nieuwe presentaties zijn. Eerst worden alle XML-bestanden gedownload en vervolgens halen we de mediabestanden op die in de presentaties gebruikt worden. In de volgende fase kopiëren we alle bestanden

naar de display units. Zoals u kunt zien, lukt dit voor de display unit met IP 192.168.91.7. De andere display units zijn blijkbaar offline.
Als er na drie pogingen nog steeds geen verbinding mogelijk is, wordt de distributieprocedure afgebroken. Uiteindelijk wordt het presentatieoverzicht geüpload zodat men in de centrale weet wat er zichtbaar is op elk scherm.

3.6. De logboeken

Zoals we gezien hebben, zorgt de service in de winkel voor twee logboeken. Enerzijds worden alle acties en problemen bijgehouden. En anderzijds wordt er een overzicht bijgehouden van de presentaties getoond worden op elk scherm.
Dit laatste overzicht moet ook geraadpleegd kunnen worden in de centrale. De winkel heeft schrijfrechten in zijn persoonlijke map op de FTP-server zodat hij de logboeken kan uploaden naar de centrale.
In de centrale worden de presentatieoverzichten vergeleken met de logboeken die de winkels hebben achtergelaten. Op die manier kan ik achterhalen welke presentaties zonder problemen gedownload zijn en getoond worden op een bepaald scherm en welke niet.

De beheerder kan op elk moment een presentatieoverzicht opvragen. Hij krijgt een boomstructuur (zoals bij het exporteren van presentaties) te zien met op het voorlaatste niveau de schermen. Per scherm kan hij alle presentaties zien die afgespeeld worden (groen gekleurd) en alle presentaties die niet getoond kunnen worden omwille van een fout tijdens het downloaden of distribueren (rood gekleurd). Ik heb twee knoppen voorzien die het mogelijk maken om enkel de groene of enkel de rode presentaties weer te geven.

De beheerder kan dus op elk moment bekijken wat er op elk scherm in elke winkel getoond wordt. Het is dus heel eenvoudig om op te sporen welke display units eventueel buiten werking zijn.

4. Testen

Tijdens en na de realisatie van het digital signage management system werd er uitvoerig getest. Ik heb een aantal verschillende testmethodes gebruikt.
Zo heb ik een aantal testklassen geschreven die testen of een methode van een klasse wel het gewenste resultaat heeft. Om de presentatiebeheerder te testen heb ik bijvoorbeeld een testklasse geschreven waarin een presentatie wordt aangemaakt, dia's toegevoegd en van volgorde verwisseld worden. Na elke stap wordt het resultaat uitgeschreven zodat het eenvoudig is om fouten op te sporen.

De laatste twee weken van de stage heb ik het systeem samen met Gunther Govaerts, Martijn Ringoot en Sien D'Hooghe getest in een testomgeving: een netwerk met centrale, winkel, weegschalen en display units. Dit netwerk was een belangrijk testplatform. Het systeem bestaat uit een aantal verschillende onderdelen die moeten samenwerken. De communicatie tussen al deze componenten kan dus enkel goed getest worden in een echt netwerk. Er werden niet enkel functionele tests uitgevoerd, ook het geheugenverbruik en de CPU-belasting werden gecontroleerd.

De laatste testmethode bestond uit gebruikerstests. Het team dat de opleiding van elk softwarepakket verzorgt, heeft het systeem dan ook uitvoerig getest en feedback gegeven.

5. Nieuwe technologieën

In de beginfase van dit project werden een aantal technologieën onderzocht.
Het was vrij snel duidelijk dat de presentatiebeheerder en de service in de winkel geprogrammeerd zouden worden in C#. Deze taal had ik reeds geleerd tijdens de ontwikkeling van een aantal projecten in mijn opleiding en vormde dus niet onmiddellijk een probleem. Wel was er bijkomend onderzoek nodig naar XML-serialisatie en de werking van GDI+.
Voor de ontwikkeling van de presentatiespeler had ik Flex voorgesteld als alternatief voor Flash (dat gebruikt werd in vorige versies van de software). Om de presentaties vanuit Flex op te halen was eerst gedacht aan AMF PHP maar dat is helaas niet doorgegaan.
In dit hoofdstuk zal ik alle technologieën beschrijven en uitleggen waarom ik ze al dan niet gebruikt heb.

Hieronder kan u een overzicht vinden van de verschillende programma's en de technologie waarmee ze ontwikkeld zijn:

Centrale

- OS: Windows Server 2003
- Servers: FTP, MySQL
- Software:
    - Presentatiebeheerder: .NET
    - Presentatievoorbeeld / Diavoorbeeld: Flex

Winkel

- OS: Windows Server 2003
- Software:
    - Windows Service: .NET
    - Servicebeheerprogramma: .NET

Schermen

- OS: Windows XP
- Software:
    - Schermbeheerder: .NET
    - Presentatiespeler: Flex

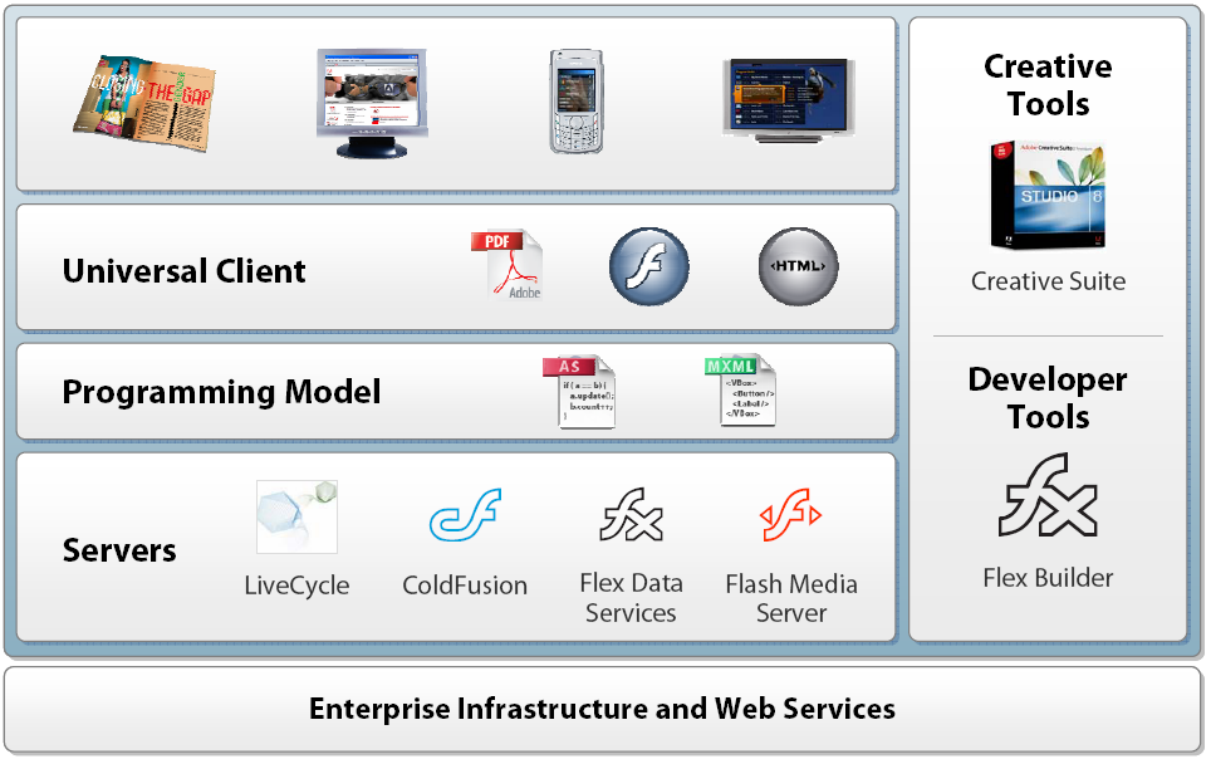
Weegschalen

- OS: Windows Embedded
- Software:
    - POS-systeem (niet door mij ontwikkeld)
    - Presentatiespeler: Flex

5.1. Adobe Flex

Het programma dat de presentaties afspeelt op de display units en weegschalen werd ontwikkeld in Adobe Flex, een framework om Rich Internet Applications te ontwikkelen. De eerste versie werd gelanceerd in 2004 en op 25 februari 2008 werd Flex 3 beschikbaar.

Het framework is gebaseerd op Adobe Flash, dat u zeker zal kennen van allerhande fraaie animaties en spelletjes op het internet. Er zijn wel enkele grote verschillen tussen Flex en Flash. In Flash wordt er gewerkt met een tijdlijn en een *stage* en zijn er allerhande gereedschappen zoals het emmertje en de pen om grafische componenten te creëren. Om interactie met de gebruiker mogelijk te maken, kan er geprogrammeerd worden in ActionScript. Flash wordt dan ook vooral gebruikt om animaties en interactieve sites te bouwen. Flex is beter geschikt voor de ontwikkeling van sites met de functionaliteit van desktopapplicaties (in de technische literatuur vaak Rich Internet Applications genoemd).
Er kan gesteld worden dat Flex dezelfde functionaliteiten en mogelijkheden biedt als Flash plus een aantal extra componenten. Flex is bedoeld voor programmeurs terwijl Flash eerder voor designers is. Zoals .NET biedt Flex een aantrekkelijke ontwikkelomgeving aan. Natuurlijk is het ook mogelijk om applicaties te ontwikkelen in een eenvoudige tekstverwerker maar Adobe maakt het leven van de programmeur heel wat aangenamer door de introductie van Flex Builder. Deze IDE is gebaseerd op Eclipse en kan ook als plug-in geïnstalleerd worden. Flex Builder bevat een aantal handige gereedschappen zoals het autoaanvullen van code, een ingebouwde compiler en verscheidene mogelijkheden om fouten op te sporen.



Figuur 54 Wat is Flex?

5.1.1. Flexprogrammeertalen

Flex biedt twee programmeertalen aan: MXML en ActionScript 3.0.

MXML is gebaseerd op XML en werd reeds geïntroduceerd door Macromedia in 2004. Een officiële verklaring voor het acroniem heeft Macromedia nooit gegeven maar een aantal ontwikkelaars suggereren dat MXML staat voor Magic eXtensible Markup Language. MXML wordt vooral gebruikt om de lay-out van een applicatie te bepalen. Naast de declaratie van grafische elementen kan MXML ook gebruikt worden om niet-visuele componenten aan te maken (zoals een HttpService om een bestand te downloaden).

Veel webdesigners en ontwikkelaars zijn vertrouwd met HTML en het is dan ook geen grote stap naar MXML. Deze magische opmaaktaal biedt een aantal voordelen: zo is deze taal beter gestructureerd en minder dubbelzinnig dan HTML. Bovendien heeft de ontwikkelaar keuze uit meer en krachtigere *tags*. Zo maken DataGrid, TabNavigator, Accordion, Tree en Menu deel uit van de standaard tagbibliotheek van MXML.
De ontwikkelaar kan ook MXML componenten uitbreiden of volledig nieuwe componenten ontwikkelen. Het grootste verschil met HTML is dat de gebruikersinterface (gedefinieerd met MXML) afgespeeld wordt door de Flash Player. Dit biedt de ontwikkelaar veel meer mogelijkheden dan de traditionele webpagina's die er bovendien verschillend kunnen uitzien in een andere browser.

Opmaaktalen zijn niet voldoende om een programma van logica te voorzien. Flex biedt dus een tweede taal aan: ActionScript 3.0.
Dit is een *strongly typed* objectgeoriënteerde scriptaal die afgeleid is van ECMAScript[10].

Tabel 1 ECMAScript

Application	Dialect	Latest dialect version	Corresponding ECMAScript edition
Mozilla Firefox, the Gecko layout engine, SpiderMonkey, and Rhino	JavaScript	1.7	ECMA-262, edition 3
Internet Explorer	JScript	5.7	ECMA-262, edition 3
Opera	ECMAScript, with extensions to both JavaScript and JScript	1.3/1.5	ECMA-262, edition 3
KHTML layout engine, KDE's Konqueror, and Apple's Safari[8]	JavaScript	1.5[6]	ECMA-262
Microsoft .NET Framework	JScript .NET	8.0	ECMA-262, edition 3
Adobe Flash and Adobe Flex	**ActionScript**	**2** **3**	**ECMA-262, edition 3** **ECMA-262, edition 4**
Adobe Acrobat	JavaScript	1.5	ECMA-262, edition 3
General purpose scripting language	DMDScript	1.06	ECMA-262
OpenLaszlo Platform	JavaScript	1.4	ECMA-262, edition 3
iCab	InScript	3.22	ECMA-262, edition 3
Max/MSP	JavaScript	1.5	ECMA-262, edition 3
Samba 4 and embedded servers.	Embedded JavaScript	***?***	ECMA-262

De syntax die in ActionScript 3.0 gebruikt wordt, is bijna identiek aan de syntax van ActionScript 2.0. Een ontwikkelaar die gewend is om ActionScript te schrijven voor het Flashplatform zal zijn programmeerkennis zeker kunnen gebruiken in een Flexomgeving. Maar ook voor mij, een programmeur die nog nooit met Flash heeft gewerkt, was de stap naar Flex snel gemaakt. ActionScript 3.0 is een volledig objectgeoriënteerde taal. Elke

[10] ECMAScript is een scripttaal die beschreven wordt door Ecma International in de ECMA-262 specificatie. De taal wordt veel gebruikt op het web en wordt vaak benoemd als JavaScript of JScript, de twee bekendste dialecten van de specificatie.

programmeur die reeds ervaring heeft met een aantal objectgeoriënteerde talen zal dus snel de basis van ActionScript 3.0 onder de knie hebben.

Naast de objectgeoriënteerde omgeving biedt de laatste versie van ActionScript nog een aantal belangrijke voordelen die de stap naar Flex en ActionScript 3.0 zeker de moeite maken.

Een van die voordelen is het melden van fouten tijdens de uitvoering van applicaties. In ActionScript 2.0 werden deze fouten verzwegen voor de programmeur. ActionScript 2.0 verzekerde dat een applicatie uitgevoerd werd zonder dat er ook maar één foutmelding zou verschijnen. Deze eigenschap werd wel als negatief beschouwd door de ontwikkelaars, aangezien zij niet de mogelijkheid hadden om een fout in de applicatie op te sporen tijdens de uitvoering ervan.

In ActionScript 3.0 krijgt de ontwikkelaar een foutmelding die interessante informatie bevat waaronder het lijnnummer waar de fout zich heeft voorgedaan. De ontwikkelaar heeft dus de mogelijkheid om fouten tijdens de uitvoering van de applicatie (zogenaamde *runtime errors*) op te sporen.

Een derde verbetering in ActionScript 3.0 is de mogelijkheid om een zogenaamde *sealed* klasse te gebruiken. Dit betekent dat de eigenschappen en methoden van een klasse niet kunnen worden aangepast tijdens de uitvoering. Naast het aanpassen, is ook het toevoegen van eigenschappen en methoden onmogelijk. Het is dus niet mogelijk om in een applicatie een instantie te maken van een klasse, en om daarna een eigenschap via de puntnotatie toe te voegen aan deze klasse. Een voorbeeld van een dergelijke klasse is de Stringklasse.
Het is ook mogelijk om gebruik te maken van een dynamische klasse, maar daarvoor moet de klasse voorzien worden van het sleutelwoord *dynamic*.

Een vierde verbetering is de invoering van *ECMAScript for XML*, beter bekend als E4X. Via E4X heb je de mogelijkheid om een XML-bestand op een eenvoudige manier te doorlopen, zodat het ophalen van data sneller en efficiënter verloopt. Waar je in ActionScript 2.0 een lus nodig had om de verschillende knopen te doorlopen, kan je in ActionScript 3.0 door de ondersteuning van E4X de puntnotatie gebruiken.
Een vijfde verbetering is beslist de ondersteuning van reguliere expressies. Je hebt nu de mogelijkheid om gebruik te maken van methodes zoals match(), replace() en search() om op een eenvoudige manier tekst op te zoeken.

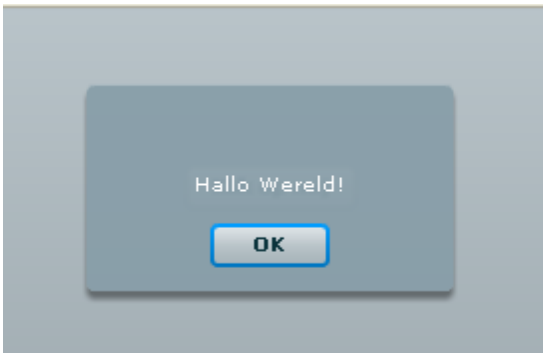
Een programmeur kan Flexapplicaties ontwikkelen in een mengeling van twee krachtige talen. De broncode wordt tijdens de compilatiefase omgezet naar een SWF-bestand dat afgespeeld kan worden door Flash Player 9. Deze laatste versie van de Flash Player biedt een groot aantal voordelen tegenover de vorige versies. Er is een nieuwe ActionScript Virtual Machine (AVM) waardoor de performantie met een factor 10 verbeterde. De Just In Time Compiler vertaalt de ActionScript bytecode naar machinecode om een maximale uitvoeringssnelheid te behalen. Bovendien ondersteunt Flash Player 9 ook nog applicaties die ontwikkeld werden in ActionScript 1.0 en 2.0. Er zijn twee virtuele machines, die afhankelijk van de versie van ActionScript die gebruikt werd, in werking treden.

Flexapplicaties worden dus ontwikkeld met een mengeling van MXML en ActionScript. MXML wordt gebruikt om de lay-out te definiëren en ActionScript om de applicatielogica vast te leggen.

Om het gebruik van MXML en ActionScript 3.0 te illustreren bekijken we een eenvoudig voorbeeldje.

```
2  <mx:Application
3      xmlns:mx="http://www.adobe.com/2006/mxml"
4      layout="absolute"
5      width="280"
6      height="180">
7
8      <mx:Script>
9          <![CDATA[
10             import mx.controls.Alert;
11
12             private function toonBoodschap():void{
13                 Alert.show("Hallo Wereld!");
14             }
15         ]]>
16     </mx:Script>
17     <mx:Button
18         id="btn"
19         label="Klik op mij"
20         horizontalCenter="0"
21         verticalCenter="0"
22         click="toonBoodschap()"/>
23 </mx:Application>
```

Figuur 55 Een eenvoudige Flexapplicatie



In de voorbeeldapplicatie wordt er een boodschap getoond als de gebruiker op de knop klikt.

De applicatie zelf wordt gedefinieerd met de MXML-tag *mx:Application* (regel 2).
Het attribuut *xmlns:mx* (XML namespace) geeft aan welke namenruimte gebruikt wordt als een tag begint met mx. MX is de naam van het pakket. De attributen *width* en *height* geven aan hoe groot de applicatie is en *layout* bepaalt hoe de elementen erin gepositioneerd moeten worden.

Het volgende blok code begint met een *mx:Script* tag die aangeeft dat alles wat volgt ActionScript 3.0 is. Om te voorkomen dat speciale tekens geïnterpreteerd worden als MXML moet de code opgenomen worden in een CDATA-sectie.
Er wordt een private methode met de naam toonBoodschap gedefinieerd die niets teruggeeft (void). In deze methode wordt een boodschap getoond aan de gebruiker. Hiervoor wordt de methode *show* van de klasse *Alert* gebruikt. Boven de methode wordt

deze klasse geïmporteerd zodat de compiler weet dat het woord *Alert* een klasse is uit het pakket mx.controls.

Na het scriptblok wordt een knop gedefinieerd met *btn* als naam (aangegeven door het attribuut *id*). Via het attribuut label wordt bepaald welke tekst er op de knop getoond wordt. *HorizontalCenter* en *verticalCenter* bepalen de plaats van de knop. Het attribuut *click* geeft aan welke methode moet uitgevoerd worden als de gebruiker op de knop klikt.

Als we dit MXML-document compileren, wordt het omgezet naar een ActionScriptklasse:

```
public class Voorbeeld extends mx.core.Application
{
	public var btn : mx.controls.Button;

private var _documentDescriptor_ : mx.core.UIComponentDescriptor = new
mx.core.UIComponentDescriptor({

  type: mx.core.Application  ,  propertiesFactory: function():Object { return {
    width: 280,
    height: 180,
    childDescriptors: [
      new mx.core.UIComponentDescriptor({
        type: mx.controls.Button,
        id: "btn"
        ,
        events: {
          click: "__btn_click"
        }
        ,
        stylesFactory: function():void {
          this.horizontalCenter = 0;
          this.verticalCenter = 0;
        }
        ,
        propertiesFactory: function():Object { return {
          label: "Klik op mij"
        }}
      })
    ]
  }}
})
//	constructor (Flex display object)

	public function Voorbeeld()
	{

		super();
		mx_internal::_document = this;

		//	properties
		this.layout = "absolute";
		this.width = 280;
		this.height = 180;

	}



	//	initialize()

	override public function initialize():void{
		mx_internal::setDocumentDescriptor(_documentDescriptor_);
```

```
                super.initialize();
        }

        //      <Script>, line 9 - 15

                        import mx.controls.Alert;

                        private function toonBoodschap():void{

                                Alert.show("Hallo Wereld!");

                        }
        //      end scripts

    // supporting function definitions for properties, events, styles, effects

public function __btn_click(event:flash.events.MouseEvent):void
{
        toonBoodschap()
}
```

Het bestuderen van deze code kan erg verhelderend zijn zeker als je zelf bepaalde functionaliteiten in ActionScript wil implementeren.
In bovenstaand voorbeeld heb ik alle code aangeduid die u meteen zal herkennen als u de MXML-versie ernaast legt.

Een MXML-applicatie komt dus overeen met een ActionScriptklasse. Het is dus perfect mogelijk om zelf alle ActionScriptcode te schrijven en geen gebruik te maken van MXML. Op die manier verlies je natuurlijk wel de voordelen die MXML te bieden heeft: gestructureerde, overzichtelijke code en een enorme tijdswinst, zeker als je de lay-out van grafische elementen wilt vastleggen. Als je enkel gebruik maakt van MXML dan moet wel alle ActionScriptcode binnen het MXML-document geplaatst worden (zoals ik in het voorbeeld gedaan heb). Dit heeft dan weer als nadeel dat er geen echte scheiding is tussen de UI en de programmalogica. Men kan dus best voor de gulden middenweg kiezen: MXML om de lay-out te definiëren en ActionScriptklassen om de programmalogica vast te leggen.

5.1.2. Werken met componenten in Flex

Componenten zijn de bouwblokken van een Flexapplicatie. Ze definiëren wat de gebruiker te zien krijgt en bepalen hoe de gebruiker kan communiceren met de applicatie.
In veel gevallen volstaan de standaard componenten die meegeleverd zijn maar soms is er nood aan een nieuwe component of uitbreiding van een bestaande component. Flex biedt drie mogelijkheden aan om *custom components* te ontwikkelen: met MXML, met ActionScriptklassen of met een combinatie van beide.
Componenten ontwikkelen in MXML gaat veel sneller maar je verliest de flexibiliteit die OO ontwikkeling met ActionScript met zich meebrengt. ActionScriptcomponenten kunnen bovendien getest worden met *unit testing frameworks* en de functionaliteit van de component kan verdeeld worden over verschillende klassen. Het is ook perfect mogelijk om MXML en ActionScriptklassen te combineren.

5.1.2.1. Componenten maken met MXML

Je kan MXML-componenten creëren om de functionaliteit van bestaande Flexcomponenten uit te breiden. De naam van de component wordt bepaald door de naam die het MXML-bestand krijgt: naam_van_de_component.mxml.

Het eerste element in het MXML-bestand is de naam van de component die je wilt uitbreiden.

Het ontwikkelen van een MXML-component gebeurt meestal in een aantal stappen:

1. Creëren van een nieuw MXML-bestand (best in een aparte map waar dan alle componenten bewaard worden).

2. Het MXML-bestand een naam geven die de component duidelijk beschrijft.

3. In plaats van de <mx:Application> tag wordt nu de componenttag gebruikt die als basis voor de eigen component dient.

4. De nodige componenten worden toegevoegd aan de root van het MXML-document.

5. Vaak wordt ActionScript toevoegt om bepaalde diensten aan te bieden of gebeurtenissen af te handelen.

Als dit proces voltooid is, kan er vanuit een Flexapplicatie naar de zelfgemaakte component verwezen worden. Het is ook mogelijk om in een zelfgemaakte component andere zelfgemaakte componenten op te nemen, wat de flexibiliteit en de herbruikbaarheid alleen maar bevordert.

Om te kunnen verwijzen naar zelfgemaakte componenten moet een namenruimte toegevoegd worden aan de <mx:Application> tag.
Als je een component hebt gemaakt met de naam mijnComponent en bewaard hebt in de map componenten dan kan volgende namenruimte gebruikt worden:

```
xmlns:comps=”be.hogent.iii.componenten.mijnComponent”.
```

De component mijnComponent kan dan als volgt aangesproken worden in de applicatie:

```
<comp:mijnComponent>
```

Het is dus vrij eenvoudig om met MXML een zelfgemaakte component af te leiden van een reeds bestaande component. Volgend voorbeeld illustreert het gebruik van een zelfgemaakte component die een vervolgkeuzelijst bevat met een aantal landen.

```
// CountriesCombo.mxml
<?xml version="1.0" encoding="utf-8"?>

<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:dataProvider>
<mx:Object label="Australia" data="AU"/>
<mx:Object label="Canada" data="CA"/>
<mx:Object label="New Zealand" data="NZ"/>
<mx:Object label="United Kingdom" data="UK"/>
<mx:Object label="United States of America" data="US"/>
</mx:dataProvider>
</mx:ComboBox>
…
<mx:Application xmlns:mx=http://www.adobe.com/2006/mxml
layout="absolute" xmlns:comp="components.*">
<mx:VBox x="10" y="10">
```

```
        <comp:CountriesCombo id="countryCBO"/>

    </mx:VBox>
    </mx:Application>
```

5.1.2.2. Componenten maken met ActionScript

Ook in ActionScript is het mogelijk om zelf componenten te maken. Ze kunnen op dezelfde manier gebruikt worden als MXML-componenten maar kunnen ook complexere inhoud bevatten zoals niet zichtbare elementen.

De ontwikkeling verloopt ook in een aantal stappen:

1. Er moet bepaald worden of de component een uitbreiding vormt op een bestaande component.

2. De ActionScriptklasse creëren en vervolgens eigenschappen en methodes toevoegen.

3. Nagaan of de component methodes van de basisklasse moet overschrijven.

4. Beslissen of de component gebeurtenissen moet doorgeven.

```
package myControls{
    public class MyButton extends Button {
            public function MyButton() {
            ...
            }
    ...
    }
}
```

5.1.2.3. Componenten maken via de code-behind techniek

Beide technieken, ActionScriptcomponenten en MXML-componenten, hebben voor- en nadelen. MXML-componenten kunnen snel ontwikkeld worden omdat Flex Builder het mogelijk maakt om op een eenvoudige manier de lay-out van de verschillende componenten te bepalen. De ActionScriptcode zit bij deze componenten tussen de MXML wat het geheel niet echt overzichtelijk maakt. ActionScriptcomponenten zijn dan weer heel flexiebel en herbruikbaar maar het is vrij lastig om de verschillende kindcomponenten uit te lijnen.
De *code-behind* techniek combineert de voordelen van de twee vorige technieken. Alles wat te maken heeft met lay-out komt in een MXML-bestand (extensie .mxml) en alle code komt in een ActionScriptbestand (extensie .as).

De ontwikkeling verloopt als volgt:

1. Creëren van een ActionScriptklasse voor de component.

2. Creëren van een MXML-component die de ActionScriptklasse als hoofdelement gebruikt en dus overerft van de ActionScriptklasse.

3. Creëren van de gepaste lay-out in de MXML-component (al dan niet met Flex Builder).

4. Creëren van methodes, eigenschappen en luisteraars in de ActionScriptklasse

In de Flexapplicatie die de presentaties afspeelt, heb ik elke soort dia en module volgens de *code-behind* techniek ontwikkeld.

5.1.2.4. Events in zelfgemaakte componenten

Zelfgemaakte componenten kunnen ook eigen gebeurtenissen genereren. De gebeurtenissen die binnen een component optreden kunnen afgehandeld worden door de component zelf maar ze kunnen ook doorgespeeld worden aan bovenliggende componenten (de applicatie of een andere component). Deze laatste techniek noemt men *Event Dispatching*.

Eerst moet de eigen gebeurtenis gedefinieerd worden.

```
[Event(name=”naamVanHetEigenEvent”, type=”flash.events.Event”)]
```

Vervolgens kan de gebeurtenis doorgespeeld worden met de methode dispatchEvent.

```
dispatchEvent(new Event(“naamVanHetEigenEvent”)
```

In bovenstaande code wordt een eenvoudige gebeurtenis van het type flash.events.Event gebruikt.

Het is ook mogelijk om zelf een nieuw type gebeurtenis aan te maken. Een eigen gebeurtenis moet wel aan een aantal voorwaarden voldoen.
Zo moet de gebeurtenis overerven van de basisgebeurtenis event flash.events.Event.
In de constructor moet de methode *super* opgeroepen worden met als argument het type van de eigen gebeurtenis.
De clonemethode van de basisgebeurtenis moet overschreven worden. Deze methode moet een kopie van het eigen gebeurtenisobject teruggeven. Op die manier wordt aangegeven wat het type is van de gebeurtenis en wat de eigenschappen zijn van de gebeurtenis.

In het volgende voorbeeld wordt deze techniek toegepast.

```
Import flash.events.Event;

public class MijnEigenEvent extends Event{
	public var mijnParameter:String;
	public static const MIJNEVENT:String = “tekstVeranderd”;
	public function MijnEigenEvent(type:String, eenObject:Object){
		super(type);
		mijnParameter= eenObject.text;
	}
```

```
        override public function clone():Event{
                return new MijnEigenEvent(type, mijnParameter);
        }
}
```

5.1.3. Werken met XML-bestanden in Flex

Een applicatie zonder data stelt niet veel voor. In dit hoofdstuk gaan we dieper in op het gebruik van XML-bestanden in Flex.

5.1.3.1. Waar staan de XML-bestanden?

Er zijn 3 manieren om vanuit Flex toegang te krijgen tot XML-data.

1. De XML-data rechtstreeks in het SWF-bestand invoegen.
2. De data uit een XML-bestand op dezelfde server halen.
3. De data uit XML-bestand op een andere server halen.

De eerste manier is niet erg flexiebel en kan enkel gebruikt worden als de data beperkt is en niet vaak verandert.

Via de Modelklasse kunnen de eerste twee manieren gerealiseerd worden.
Ofwel wordt het XML-document gedefinieerd binnen de Modeltag zoals in het eerste voorbeeld:

```
<mx:Model id=”boekenWinkel”>
<boeken>
        <boek>
                <titel taal=”Nederlands”>Flex 2</titel>
                <beschrijving>Een goed boek</beschrijving>
        </boek>
</boeken>
</mx:Model>
```

Maar er kan ook verwezen worden naar een XML-bestand:

```
<mx:Model id=”boekenWinkel” source=”boeken.xml”/>
```

We kunnen dus naar een XML-bestand verwijzen met het sourceattribuut of we kunnen het XML-document definiëren binnen de Modeltag.
Als de Modelklasse gebruikt wordt om een XML-bestand aan te spreken dan zal Flex de XML-data bij het compileren in het SWF-bestand verpakken. Deze techniek kan gebruikt worden als de data nooit zal veranderen maar is dus niet echt flexibel.

Flex biedt daarom nog een tweede manier aan om externe XML-data op te halen.
De HTTPServicetag kan XML-data op een dynamische manier inlezen. Dit wil zeggen dat de data pas opgehaald wordt als we daar om vragen en dus niet verpakt wordt in het SWF-bestand. Als we de methode *send* oproepen dan wordt er tijdens de uitvoering van de Flexapplicatie een asynchroon verzoek gedaan naar het gewenste XML-bestand. Via

het URL-attribuut kan aangegeven worden waar het XML-bestand zich bevindt. Het XML-bestand kan zich op dezelfde computer als de Flexapplicatie bevinden maar dit is zeker niet noodzakelijk.

```
<mx:HTTPService id="boekData" url="http://abc.be/boeken.xml" />
```

Bovenstaande code geeft enkel aan waar het XML-bestand zich juist bevindt. De data wordt nog niet opgehaald, daarvoor moeten we de methode *send* aanroepen Wanneer alle XML-data ontvangen is, wordt de resultgebeurtenis gegenereerd. De data is nu beschikbaar in het lastResultattribuut van de HTTPService.

XML-data kan in Flex op een eenvoudige manier gebonden worden aan bijvoorbeeld een DataGrid. Als de XML-data wijzigt dan zal ook de DataGrid ververst worden.

```
<mx:DataGrid dataProvider="{boekData.lastResult.boeken.boek}"/>
```

Het dataProviderattribuut geeft aan welke data weergegeven moet worden in de DataGrid. In dit geval wordt de data die werd opgehaald door de HTTPService (lastResult) weergegeven: elke rij bevat de eigenschappen van één boek.

5.1.3.2. XML en ActionScript

De technieken om XML-data op te halen die we hiervoor behandeld hebben, zijn goed te gebruiken in eenvoudige situaties. Als we echter complexe data willen weergeven of deze op een logische en efficiënte manier willen overlopen dan volstaan deze eenvoudige technieken niet. In ActionScript kunnen we XML-bestanden op twee manieren benaderen: via een ArrayCollection of via E4X.

5.1.3.2.1. ArrayCollections

Met ActionScript kunnen we complexe XML-data eenvoudig overlopen en manipuleren. Laten we eerst even kijken wat er juist gebeurt achter de schermen als er XML-data opgehaald wordt. ActionScript vormt het XML-document om naar een multidimensionale array. De *remote procedure component* van ActionScript zorgt voor de omzetting van het XML-document naar een array en de HTTPService verpakt deze array in een ArrayCollection. Een ArrayCollection is een klasse die zich rond een array nestelt en heeft een aantal functies om de data in de array zelf te manipuleren.

Via ActionScript kunnen we de data zelf opvangen, eventueel manipuleren en nadien doorsturen of binden aan een component. Er kunnen ook fouten optreden tijdens het ophalen van de XML-data. Deze kunnen ook opgevangen worden in ActionScript.

```
<mx:HTTPService id="boekData" url="http://abc.be/boeken.xml"
result="boekHandler(event)"
fault="foutHandler(event)"/>

private function boekHandler(evt:ResultEvent):void{
        trace(evt.result.boeken.boek);
}

private function foutHandler(evt:FaultEvent):void{
        trace("Er is een fout opgetreden: " + evt.message);
```

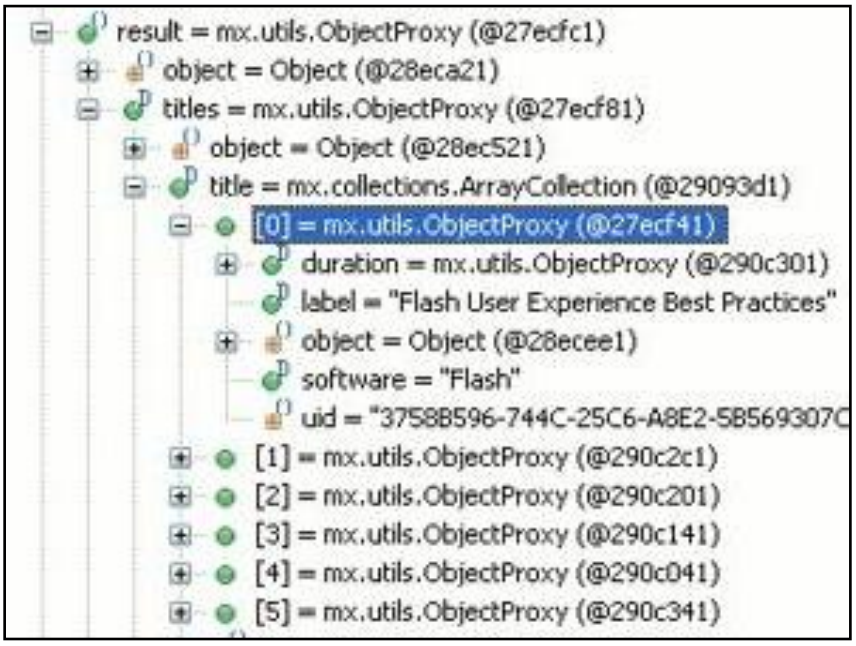
```
        }
```

Dit stuk code haalt het XML-bestand boeken.xml op. Als alles zonder problemen verloopt dan wordt de methode boekHandler uitgevoerd. Fouten worden afgehandeld door de methode foutHandler.

5.1.3.2.2. E4X

Zoals we gezien hebben wordt XML-data bewaard als een ArrayCollections in een Flexapplicatie. Dit is niet de enige mogelijke structuur om XML-data te bewaren. In ActionScript 3.0 kunnen we de XML-data ook benaderen via ECMAScript for XML (E4X). E4X maakt het mogelijk om op een eenvoudige manier door de XML-data te navigeren en de data te filteren. Het is dan ook bedoeld als een alternatief voor DOM.

De XML-data wordt nu behandeld als een primitief type (bij het gebruik van ArrayCollections werd de data aangesproken op objectniveau). Dit verschil is duidelijk merkbaar op de twee volgende figuren.



Figuur 56 ArrayCollections

Figuur 57 E4X

Omdat we het XML-document nu op een andere manier (niet meer als ArrayCollection) willen benaderen moeten we de HTTPService vertellen wat het formaat van de XML-data in de Flexapplicatie moet zijn. Dit kan gerealiseerd worden met het attribuut resultFormat van de HTTPService.

```
<mx:HTTPService id="boekData" url="http://abc.be/boeken.xml"
result="boekHandler(event)"
resultFormat="e4x"/>
```

Wanneer de XML-data opgehaald is, kunnen we het resultaat behandelen als een XML-object:

```
private function boekHandler(evt:ResultEvent):void{
	var result:XML = evt.result as XML;
}
```

De variabele *result* in bovenstaande code verwijst nu naar het rootelement van het XML-document.

Aan de hand van E4X-expressies kunnen we vervolgens bepaalde elementen uit het XML-object ophalen.

```
var boekenLijst:XMLList = result.boek as XMLList;
```

Met bovenstaande expressie wandelen we als het ware van het rootelement naar het eerstvolgende kindelement, boek, en omdat er meerdere kindelementen zijn, krijgen we een lijst terug van alle boekelementen.

Hieronder nog een aantal voorbeelden van E4X-expressies:

- Het eerste element uit een lijst van elementen ophalen:

```
var boek:XML = result.boek[0] as XML
```

- Een element met een bepaalde naam op gelijk welke diepte in de hiërarchie ophalen:

```
var boekenLijst:XMLList = result..titel as XMLList;
```

- Een element met bepaalde eigenschappen ophalen (filteren):

```
var boekenLijst:XMLList = result..titel.(@taal == 'Nederlands') as XMLList;
```

Wie vroeger reeds gewerkt heeft met X-Path zal deze manier van werken zeker herkennen. De syntax is wel niet helemaal dezelfde. In onderstaande tabel kan u een overzicht vinden van de belangrijkste verschillen.

Tabel 2 Het verschil tussen XPath en E4X (Tank, 2007)

XPath	Meaning	E4X Equivalent
`element/*`	Select all children of element	`element.*`
`element/@*`	Select all attributes of element	`element.@*`
`element//descendent`	Select all descendents (children, grandchildren, etc.) of element	`element..descendent`
`..` `parent::element`	Select the parent of element	`element.parent()`

`xmlns:foo="..."`
 	
`element/foo:bar`
	Select the foo:bar child of element where foo is the prefix of a declared namespace	`var foo = new Namespace(...);`
		
		`element.foo::bar`

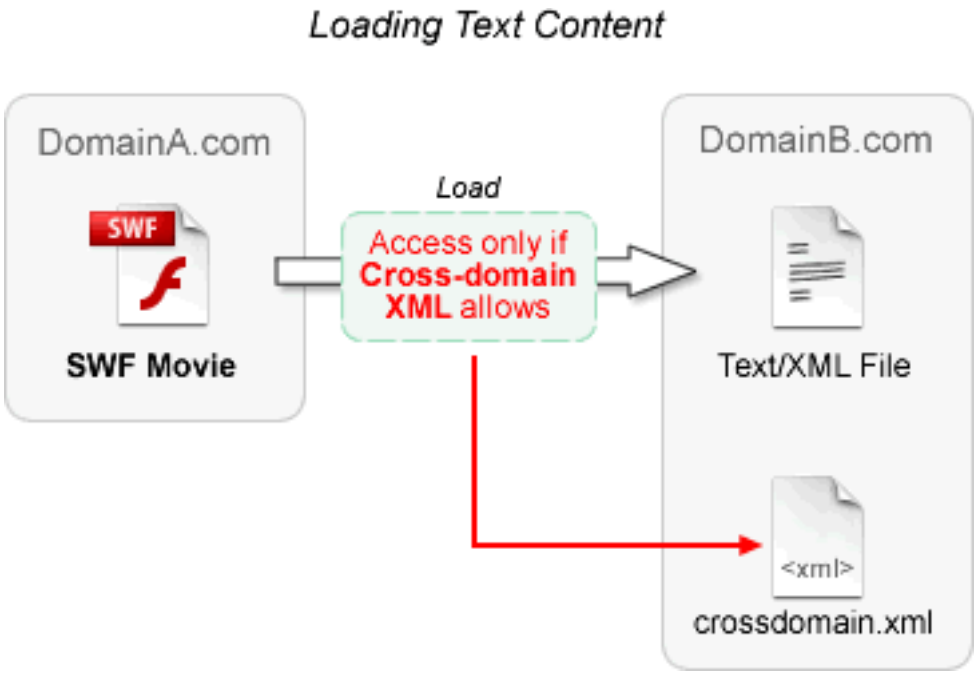
`name(element)`	Return the full name (including prefix if any) of element	`element.name()`
`local-name(element)`	Return the local name of element	`element.localName()`
`namespace-uri(element)`	Return the namespace uri (if any) of element	`element.namespace()`
`element/namespace::*`	Return the collection of namespaces as an Array of Namespace objects (E4X) or a nodeset of Namespaces nodes (XPath)	`element.inScopeNamespaces()`
`element/processing-instructions(name)`	Return the processing instruction children of element with the specified name (if omitted, all are returned).	`element.processingInstructions(name)`

`string(element)`	Return the concatenated text nodes of this element and all its descendants	`stringValue(element);`
		
		`stringValue.visible = false;`
		`function stringValue(node) {`
		`  var value = "";`
		`  if (node.hasSimpleContent()) {`
		`  value = node.toString();`

		`} else {`
		`  for each (var c in node.children()) {`
		`  value += stringValue(c);`
		`  }`
		`  }`
		`  return value;`
		`}`

5.1.3.3. Is het wel veilig?

We hebben gezien dat het mogelijk is om externe XML-documenten te laden in een Flexapplicatie. Het XML-document kan zich dus ook ergens op een server bevinden waar we zelf geen controle over hebben. Dit kan onveilig zijn. De Flash Player, die verantwoordelijk is voor het afspelen van Flexapplicaties, draait in een zandbak. Dat wil zeggen dat alle data die de Flexapplicatie gebruikt zich in hetzelfde domein moet bevinden.
Deze veiligheidsmaatregel kan soms voor onverwachte situaties zorgen.
Wie zich niet bewust is van deze veiligheidspolitiek zal vaak volgende foutboodschap zien verschijnen "Security error accessing url" die aangeeft dat het XML-bestand zich niet in hetzelfde domein bevindt en dus mogelijk onveilig is.



Figuur 58 Externe XML-bestanden inladen

Door deze strenge veiligheidspolitiek kunnen we dus geen XML-bestanden inladen die op een andere server staan.
Gelukkig bestaat er een eenvoudige oplossing voor dit probleem. Het volstaat om een configuratiebestand (crossdomain.xml) aan te maken op externe server die het XML-bestand beheert.

```
<?xml version="1.0"?>
```

```
<!DOCTYPE cross-domain-policy
SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="192.168.10.3"/>
</cross-domain-policy>
```

Het attribuut *domain* van de tag *allow-access-from* geeft aan welke computers toegang hebben tot de bestanden op de server.

We hebben gezien dat het mogelijk is om in een Flexapplicatie XML-bestanden in te laden. De bestanden hoeven niet op dezelfde locatie als de applicatie te staan. Met behulp van een webserver en server-side scripting is het dus mogelijk om de XML-bestanden dynamisch te genereren op basis van bijvoorbeeld gegevens in een database.

Naast XML-data kan Flex ook nog een aantal andere dataformaten verwerken. In het volgende hoofdstuk zullen we dieper ingaan op ActionScript Message Format.

5.1.4. Werken met AMF in Flex

Naast XML-bestanden kunnen we ook AMF (ActionScript Message Format) gebruiken om data uit te wisselen tussen een Flexapplicatie en een server. Dit protocol is ontworpen door Macromedia (Adobe) en maakt geen gebruik van XML-data.
Objecten die met dit protocol communiceren verzenden en ontvangen binaire code wat minder overhead creëert dan bijvoorbeeld SOAP. Het ophalen van data met AMF wordt ook Remoting genoemd. Remoting wordt door Macromedia ondersteund voor J2EE, ColdFusion, .NET, Ruby, PHP en er komen nog steeds implementaties bij.

Laten we even een van de eenvoudigste manieren bekijken om data op te halen met AMF. We hebben alleen een webserver met PHP-ondersteuning en AMFPHP (http://www.amfphp.org), de gratis open-source PHP-implementatie van AMF, nodig.

Eerst maken we de serverklasse:

```
class SimplePerson {
    function getPerson() {
        $fp = fopen("person.txt", "r");
        $contents = fread($fp, filesize("person.txt"));
        fclose($fp);
        $data = explode("|", $contents);
        return $data;
    }
    function savePerson($name, $age, $address){
       $contents = "$name|$age|$address";
         $fp = fopen("person.txt", "w");
       if(fwrite($fp, $contents)){
           fclose($fp);
           return "Data saved";
       };
       return "Error writing data";
    }
}
```

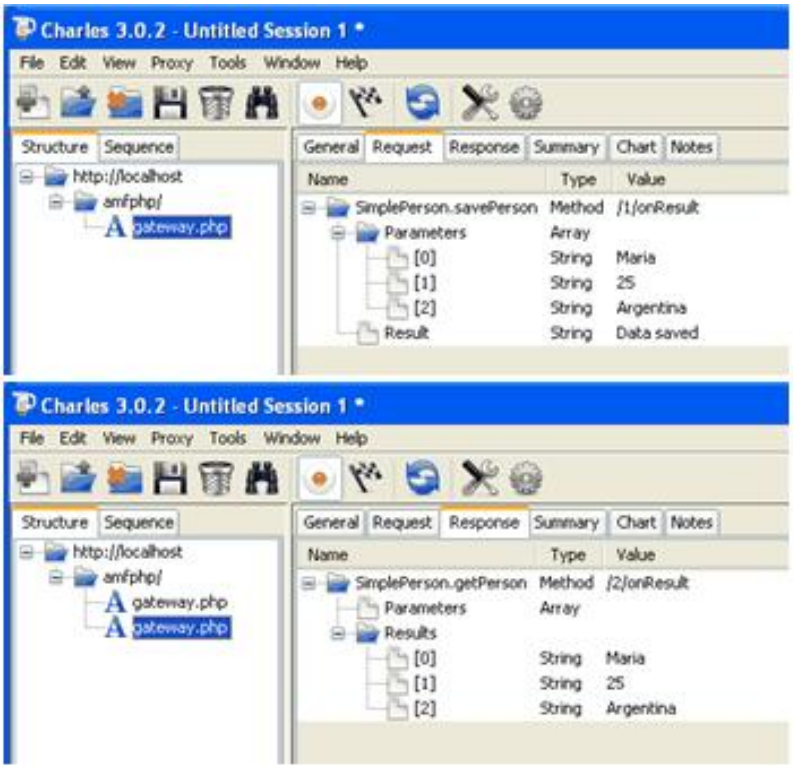
Deze klasse bewaart de gegevens van een persoon in een tekstbestand en kan de data nadien weer uitlezen.

Vervolgens maken we gebruik van de persoonklasse in de Flexapplicatie:

```
//declare remoting service
var myService = new NetConnection()
myService.connect("http://localhost/amfphp/gateway.php") //modify if
neccesary to match your own
//Buttons actions
loadData_btn.addEventListener(MouseEvent.MOUSE_DOWN, loadData)
saveData_btn.addEventListener(MouseEvent.MOUSE_DOWN, saveData)

//load handler
function loadData(evt:MouseEvent){
    var responder = new Responder(getPerson_Result, onFault);
    myService.call("SimplePerson.getPerson", responder);
}
//save handler
function saveData(evt:MouseEvent){
    if(name_txt.text=="" || age_txt.text=="" ||
address_txt.text==""){
        return status_txt.text = "All fields are required"
    }
    var responder = new Responder(savePerson_Result, onFault);
    myService.call("SimplePerson.savePerson", responder,
name_txt.text, age_txt.text, address_txt.text);
}
//Load callback
function getPerson_Result(result){
    name_txt.text = result[0]
    age_txt.text = result[1]
    address_txt.text = result[2]
    status_txt.text = ""
}
//Save callback
function savePerson_Result(result){
    name_txt.text = age_txt.text = address_txt.text = ""
    status_txt.text = result
}
//error callback
function onFault(f:Event ){
    trace("There was a problem: " + f.description);
}
```

Eerst creëren we een *NetConnection* instantie en maken verbinding met de gateway. Vervolgens geven we aan welke functies opgeroepen moeten worden als de gebruiker op een van de knoppen klikt.
Als de gebruiker data wil ophalen wordt de functie *loadData* uitgevoerd. Met de *Responder* geven we aan welke functie uitgevoerd moet worden als alles zonder problemen verloopt (getPerson_Result). We geven ook aan welke functie uitgevoerd moeten worden als er fouten optreden (onFault).
We roepen de methode *getPerson* op van de klasse *SimplePerson* en geven de *Responder* mee. De aanvraag wordt verpakt in ActionScriptpakketten en zal vervolgens naar de

server gestuurd worden waar de gateway de pakketten deserialiseert. De PHP-code wordt uitgevoerd en het resultaat wordt geserialiseerd (AMF-antwoord) en naar de Flexapplicatie gestuurd. Als het antwoord is aangekomen wordt de functie *getPerson_Result* uitgevoerd die ervoor zorgt dat alle tekstvakken de juiste waarde krijgen. Indien de server niet bereikbaar is of er iets misloopt tijdens het uitvoeren van de PHP-code wordt de functie *onFault* uitgevoerd.
De functie *saveData* werkt op dezelfde manier.



Figuur 59 Data versturen (boven) en ophalen (onder) via AMF

Bovenstaand voorbeeld toont aan hoe eenvoudig het is om vanuit een Flexapplicatie functies van een PHP-klasse aan te spreken. We kunnen dit voorbeeld wel nog verbeteren door gebruik te maken van OO-technieken.
Voorlopig wordt er een lijst van variabelen doorgestuurd:

```
myService.call("SimplePerson.savePerson", responder, name_txt.text,
age_txt.text, address_txt.text);
```

Het is beter om het persoonobject in zijn geheel door te sturen dan elk van de eigenschappen van het object afzonderlijk.

We maken dus een nieuwe persoonklasse:

```
class Person {
    var $name;
    var $age;
    var $address;
    // explicit actionscript package
```

```
    var $_explicitType = "com.flashdb.Person";

    /**
    * Constructor with default values
    */
    function Person($name="", $age=0, $address=""){
        $this->name = $name;
        $this->age = $age;
        $this->address = $address;
    }
    /**
    * Load person data from a textfile
    */
    function loadData(){
        $fp = fopen("person.txt", "r");
        $contents = fread($fp, filesize("person.txt"));
        fclose($fp);
        $data = explode("|", $contents);
        $this->name = $data[0];
        $this->age = $data[1];
        $this->address = $data[2];
    }
    /*
    * Saves person data to a textfile
    * @return Boolean on succes
    */
    function logData(){
        $fp = fopen("person.txt", "w");
        if(fwrite($fp, $this->name."|".$this->age."|".$this-
>address)){
            fclose($fp);
            return true;
        };
        return false;
    }
}
```

De persoonklasse heeft nu een aantal attributen die opgevuld worden met data uit het tekstbestand.

Om deze klasse te gebruiken in de Flexapplicatie zal AMFPHP de PHP-klasse moeten mappen op een ActionScriptklasse. Het attribuut *$_explicitType* zorgt ervoor dat Flex de PHP-klasse *Person* correct kan mappen naar een ActionScriptklasse.
Voor de programmeur lijkt het echter alsof het mogelijk is om het PHP-object rechtstreeks aan te spreken vanuit Flex.

Zoals in het vorige voorbeeld schrijven we ook de methodes *getPerson* en *savePerson* maar nu geven we een persoonobject terug in plaats van een tabel van waarden.

```
class TestPerson {
    function getPerson() {
        include_once("vo/com/flashdb/Person.php");
        $p = new Person();
```

```
        $p->loadData();
        return $p;
    }
    function savePerson(Person $p){
        $msg = $p->logData();
        return $msg;
    }
}
```

In Flex kunnen we de instellingen van de AMFPHP-service bewaren in een apart XML-bestand (services-config.xml).

```
xml version="1.0" encoding="UTF-8"
<services-config>
    <services>
        <service id="amfphp-remoting"
class="flex.messaging.services.RemotingService"
messageTypes="flex.messaging.messages.RemotingMessage">
            <destination id="amfphp">
                <channels>
                    <channel ref="my-amfphp"/>
                </channels>
                <properties>
                    <source>*</source>
                </properties>
            </destination>
        </service>
    </services>
    <channels>
        <channel-definition id="my-amfphp"
class="mx.messaging.channels.AMFChannel">
            <endpoint uri="http://localhost/amfphp/gateway.php"
class="flex.messaging.endpoints.AMFEndpoint"/>
        </channel-definition>
    </channels>
</services-config>
```

Als we een PHP-object willen aanspreken, kunnen we naar de AMFPHP-server verwijzen met het woord *amfphp*, dit wordt aangegeven door: `<destination id="amfphp">`.

In de Flexapplicatie maken we een *RemoteObject* aan dat verwijst naar het PHP-object op de AMFPHP-server. Het faultattribuut geeft aan welke ActionScriptfunctie uitgevoerd moet worden indien er fouten optreden. Voor elke methode die het PHP-object aanbiedt, geven we aan welke methode uitgevoerd moet worden als alles correct verlopen is.

```
<?
<mx:RemoteObject id="myservice" fault="faultHandler(event)"
showBusyCursor="true" source="TestPerson" destination="amfphp">
    <mx:method name="getPerson" result="getPerson_Result(event)" />
    <mx:method name="savePerson" result="savePerson_Result(event)" />
</mx:RemoteObject>
```

```
?>
```

We hebben iets meer code moeten schrijven maar dat loont zeker de moeite. Als we nu de methode *savePerson* oproepen, kunnen we gewoon een object meegeven in plaats van een lijst van parameters. Het is ook niet meer nodig om een tabel met waarden te overlopen als we een persoon opvragen. We kunnen gewoon de attributen naam, leeftijd en adres gebruiken.

```
private function savePerson_Result(evt:ResultEvent):void
{
      if(evt.result){
         name_txt.text = age_txt.text = address_txt.text = ""
         status_txt.text = "Data saved"
      } else
         status_txt.text = "Problem saving data"
}

private function getPerson_Result(evt:ResultEvent):void{
      //evt.result is a Person instance
      name_txt.text = evt.result.name
      age_txt.text = evt.result.age
      address_txt.text = evt.result.address
      status_txt.text = ""
}
```

5.1.5. De juiste technologie?

Flex is ideaal om software te ontwikkelen voor mediadisplays. Het is mogelijk om mooie animaties te maken en dankzij ActionScript 3.0 kan een programmeur zonder Flashervaring meteen aan de slag. Bovendien kan je ook als programmeur zonder grafische achtergrond aantrekkelijke applicaties ontwikkelen.
Bij de aanvang van dit project was het de bedoeling om in elke winkel een webserver te plaatsen met daarop alle mediabestanden, presentatieoverzichten en presentaties. Tijdens de analyse heeft Webcare beslist dat dit niet mogelijk was omdat het netwerk continu belast zou zijn. Voor winkels die enkel display units hebben, is deze belasting niet echt een probleem maar voor weegschalen die in de toekomst hoogst waarschijnlijk opgenomen worden in een draadloos netwerk is deze belasting veel te riskant.
Het POS-systeem mag niet leiden onder het dataverkeer dat veroorzaakt wordt door de mediadisplays. Er is dus geen webserver in de winkel en het is dus ook niet mogelijk om met AMFPHP data op te halen. De data wordt door een service, die draait op de server in de winkel, gekopieerd naar alle display units en weegschalen. De overzichtsbestanden worden bewaard als XML-bestanden die de Flexapplicatie op display units en weegschalen inleest met E4X.

5.2. Microsoft .NET

De presentatiebeheerder is ontwikkeld in de taal C# die samen met het .NET framework geïntroduceerd is en beschouwd wordt als een van de belangrijkste talen voor het platform.
Aangezien u vertrouwd bent met C# zal ik enkel een aantal speciale aspecten van deze taal bespreken die ik gebruikt heb.

5.2.1. XML-serialisatie

XML-serialisatie is het proces waarbij objecten omgezet worden naar entiteiten in een XML-bestand.
Dit proces kan gebruikt worden om objecten persistent te maken en ze later weer te gebruiken in het programma. In .NET bestaan er een aantal klassen om deze serialisatie uit te voeren. De XmlSerializer[11] kan objecten serialiseren/deserialiseren en bepaalt hoe de codering gebeurt tijdens deze omzetting. Het resultaat kan bewaard worden in een *stream*. Standaard worden publieke eigenschappen en velden omgezet naar XML-elementen met bijhorende waarde. Dit werkt goed voor eenvoudige objecten maar bij complexe structuren en hiërarchieën loopt het al snel mis.

Laten we eerst een eenvoudig voorbeeld bekijken:

```
class Person
{
     private String personName;
     private Int32 personAge;

     public String Name
     {
       get { return personName;  }
       set { personName = value; }
     }
     public Int32 Age
     {
       get { return personAge; }
       set { personAge = value;}
     }
}
```

De klasse *person* heeft twee publieke eigenschappen: *Name* en *Age*. We maken nu een instantie van deze klasse en kennen waarden toe aan deze eigenschappen:

```
Person oPerson = new Person();
oPerson.Name = "Anthony";
oPerson.Age = 38
```

Vervolgens maken we een object aan van het type *XMLSerializer* en geven aan dat we een object van het type *Person* wensen te serialiseren.

[11] http://msdn2.microsoft.com/en-us/library/system.xml.serialization.aspx

We maken ook een *XMLTextWriter* aan die het resultaat van de serialisatie zal wegschrijven naar de gewenste *stream*.
Het serialiseren gebeurt pas als we de methode *Serialize* oproepen met twee parameters: een *writer* en een object van het type *Person*.

```
MemoryStream memoryStream = new MemoryStream ();
XmlSerializer xs = new XmlSerializer ( typeof (Person ) );
XmlTextWriter xmlTextWriter = new XmlTextWriter ( memoryStream,
coding.UTF8 );
xs.Serialize( xmlTextWriter, oPerson );
```

Als we de inhoud van het object *memoryStream* uitprinten, krijgen we volgende XML-data te zien:

```
<Person>
<Name>Anthony</Name>
<Age>38</Age>
</Person>
```

Zoals u kan zien, worden de eigenschappen *Age* en *Name* omgezet naar de overeenkomstige XML-elementen en wordt er voor de klasse *Person* een overeenkomstig rootelement aangemaakt.

Voor complexere structuren zoals een presentatie met verschillende dia's kan het serialisatieproces preciezer geconfigureerd worden. In C# kan dat op twee manieren gebeuren.

Ten eerste kan men speciale codes (ook attributen genoemd) schrijven boven een klasse, eigenschap of veld. Tijdens het serialisatieproces worden deze codes geïnterpreteerd en zal de *XMLSerializer* speciale acties ondernemen.
De syntax van deze codes is als volgt:

```
[System.Xml.Serialization.NaamVanAttribuut(argumenten)]
```

Met *XmlRoot* kan bijvoorbeeld aangegeven worden dat we een andere naam dan de naam van de klasse wensen te gebruiken voor het rootelement.
XmlAttribute geeft dan weer aan dat we een veld of een eigenschap naar een XML-attribuut willen vertalen en dus niet de standaard regel volgen die een eigenschap of veld vertaalt naar een XML-element.
Soms bevatten objecten ook eigenschappen of velden die we niet wensen te serialiseren, hiervoor kan het *XmlIgnoreAttribute* gebruikt worden.

Enkele voorbeelden (we gebruiken hier het persoonobject uit het vorige voorbeeld):

Tabel 3 XML-serialisatie via attributen

Klasse Person	**Geserialiseerd object**

`[XmlRootAttribute(Name="Human")]`
`public class Person`
`{`
	`<Human>`
	`<Name>Anthony</Name>`
	`<Age>38</Age>`
	`</Human>`

} 	

class Person
  {
[[XmlAttribute(AttributeName="ID")]
  public String Name
  {
  get { return personName; }
  set { personName = value; }
  }
  }
	<Person>
	  <ID>Anthony</ID>
	  <Age>38</Age>
	</Person>

class Person
  {
 	
  [XmlIgnore]
  public Int32 Age
  {
  get { return personAge; }
  set { personAge = value;}
  }
  }
	**<Person>
	  <Name>Anthony</Name>
	</Person>**

Naast het gebruik van speciale attributen is het ook mogelijk dat een klasse de interface *IXmlSerializable* implementeert en zo aangeeft hoe de omzetting van en naar XML moet gebeuren.

Een klasse die deze interface implementeert moet drie methodes implementeren:

- GetSchema: bedoeld om een XML-schema te definiëren, dit schema wordt veel gebruikt voor de creatie van de WSDL van een Webservice.
- ReadXml: wordt gebruikt bij de deserialisatie (een XML-document omzetten naar een object).
- WriteXml: wordt gebruikt bij de serialisatie (een object omzetten naar een XML-document).

```
interface IXmlSerializable
{
  System.Xml.Schema.XmlSchema GetSchema (  )
  void ReadXml ( System.Xml.XmlReader reader )
  void WriteXml ( System.Xml.XmlWriter writer )
}
```

5.2.2. GDI+

Afbeeldingen bewerken in Photoshop is kinderspel maar zelf code schrijven om eenvoudige beeldbewerkingen uit te voeren is al heel wat lastiger. Gelukkig bestaat er in .NET een grafische bibliotheek, GDI+, die het coderen van deze beeldbewerkingen een pak vereenvoudigt.
Ik heb deze bibliotheek gebruikt om afbeeldingen uit te snijden en te vergroten. Hoe dit juist in zijn werk gaat staat beschreven in hoofdstuk 3.1.2.2.

6. Nabeschouwing

Een aantal maanden voor de aanvang van de stage had ik reeds een grondige analyse gemaakt van het digital signage management system. Het was niet eenvoudig om te voorspellen wat er allemaal gerealiseerd zou kunnen worden nog voor er één letter code geschreven was. Maar als ik het uitgebreid voorstel van de stage vergelijk met het systeem zoals het nu is dan kan ik terugkijken op een geslaagde stage. Het was een uitdaging om nieuwe technologieën te onderzoeken en ze te combineren in een project dat ook echt gebruikt zal worden.
Ik ben dus zeker een aangename en leerrijke ervaring rijker.

Deel 1. Software om een presentatie aan te maken en te beheren (.NET)

Een presentatie bestaat uit verschillende soorten dia's:

- Foto: een dia met een foto en tekst.
- Video: een dia met een filmpje.
- Promotie: een dia waarop promoties worden weergegeven
  - ➔ Het scherm past zich automatisch aan. De promoties worden mooi verdeeld over het scherm.
  - ➔ Er kunnen artikelen toegevoegd worden met een beschrijving, prijs/eenheid, prijs/maateenheid (liter/kg/m/...), taksen, foto, ...
- Het moet eenvoudig zijn om een nieuw type dia toe te voegen.

De drie soorten dia's zijn aanwezig.
Foto's voor een fotodia worden bijgesneden en geschaald als de verhoudingen niet kloppen.
Op een promotiedia kunnen maximum vier artikels geplaatst worden. Afhankelijk van het aantal promoties heeft de dia een andere lay-out. De promoties kunnen geïmporteerd worden uit de artikeldatabank die onafhankelijk van de promoties beheerd kan worden.
Door het ontwerp is het vrij eenvoudig om een nieuw type dia toe te voegen. Er moet een nieuwe type dia gecreëerd worden in de presentatiebeheerder (.NET). De nieuwe dia moet erven van de abstracte diaklasse en heeft op die manier automatisch een aantal algemene eigenschappen. Specifieke eigenschappen kunnen in de klasse van de nieuwe dia gespecificeerd worden. In het programma om de presentaties af te spelen (Flex) moet er een grafische component gemaakt worden die de IDia interface implementeert.

Deze dia's kunnen beheerd worden:

- De volgorde van de dia's in de presentatie kan aangepast worden.
- De parameters van elke dia kunnen aangepast worden. Specifieke parameters zoals de prijs bij de promotiedia alsook algemene parameters zoals tijdsduur van een dia.

De beheerder kan met drag and drop op een intuïtieve manier de volgorde van de dia's veranderen. Hij ziet links een overzicht van alle dia's in de presentatie, in het midden een voorbeeld van de geselecteerde dia en rechts de dia-eigenschappen.
De algemene dia-eigenschappen tijdsduur en titel kunnen gewijzigd worden via een User Control (die hergebruikt wordt voor elk type dia). Eigenschappen die specifiek zijn voor een dia zijn ook gegroepeerd op een User Control.
Op een promotiedia kunnen maximum vier promoties staan waarvan de prijs, promotietekst en de doorstreepte prijs gewijzigd kunnen worden.

De presentaties kunnen beheerd worden:

- Er kan aangegeven worden welke periode de presentaties getoond worden.

- Een presentatie kan getoond worden op 1 of meerdere opgegeven schermen of op allemaal.

De beheerder kan op elk moment een presentatie exporteren. Dit wil zeggen dat hij kan aangeven op welke schermen (die op een heel overzichtelijke manier worden weergegeven) en gedurende welke periode de presentatie getoond moet worden. Omdat de schermen hiërarchisch zijn ingedeeld, is het heel eenvoudig om bijvoorbeeld alle schermen in een bepaalde regio met één muisklik te selecteren.

- De locatie van de databaseserver en de dataserver kan opgegeven worden. (De database en de mediabestanden moeten zich dus niet noodzakelijk op dezelfde server bevinden)

In het App.conf bestand staan een aantal parameters die aangepast kunnen worden (ook na de installatie).

- de connectiestring om verbinding te maken met de databaseserver (bevat o.a. de poort en het IP van de server)
- de locatie van de FTP-server
- de locatie van de centrale gedeelde map

Men kan er dus voor kiezen om drie verschillende servers te gebruiken maar het is ook mogelijk om slechts één server te gebruiken.

Het aanmaken van de presentaties kan gebeuren op de centrale server alsook op de lokale server van een van de 'remote locations'. De 'remote locations' kunnen ook presentaties van de centrale server binnenhalen en de lokale kopie naar wens aanpassen. Deze software zal dus zowel op de centrale server als op de lokale servers van de 'remote locations' geïnstalleerd worden. Er zal dan een soort Master/Slave configuratie zijn (Master = centrale server, Slave = lokale server).

Er werd bij aanvang van deze stage beslist om de winkels geen rechten te geven in deze versie van het systeem. Men kan dus enkel presentaties aanmaken en wijzigen in de centrale. Het systeem is wel zo ontworpen dat er geen grote aanpassingen nodig zijn om ook de winkels de mogelijkheid te geven om de presentaties te bewerken. Zo is op elk niveau de mappenstructuur identiek. Omdat presentaties nu bewaard worden als XML-bestanden en niet meer in een database, is er ook geen nood meer aan een database in elke winkel tenzij men daar ook de artikels moet kunnen beheren.

De presentaties worden door de klanten zelf aangemaakt, het is dus belangrijk dat deze software heel gebruiksvriendelijk is.

Volgende eigenschappen van het systeem dragen bij tot de gebruiksvriendelijkheid:

- Intuïtieve interface: drag and drop

- Abstractie: de beheerder hoeft enkel een aantal artikels te importeren en er wordt een mooie promotiedia gemaakt (men moet geen grafisch expert zijn, in PowerPoint zou men veel meer moeite moeten doen om hetzelfde resultaat te bekomen).

- Hiërarchische structuur: de schermen zijn hiërarchisch ingedeeld zodat de beheerder snel en eenvoudig het juiste scherm kan vinden.

- Voorbeeld: de beheerder ziet een voorbeeld van de actieve dia en hij kan ook op elk moment een voorbeeld van de hele presentatie bekijken.

- Overzicht: de beheerder kan op elk moment een overzicht bekijken van alle schermen en de presentaties die erop getoond (zouden moeten) worden.

- Centraal: de beheerder kan alles centraal beheren, na de installatie is er dus geen beheerder meer nodig in de winkels. Bovendien kan men in de centrale aan de hand van een logboek volgen welke presentaties correct getoond worden en welke niet.

- Dynamisch: de beheerder kan de presentaties op twee manieren naar de winkels verspreiden. Hij kan instellen wanneer welke winkel mag controleren of er nieuwe of gewijzigde presentaties zijn. Daarnaast kan hij ook op elk moment een winkel verplichten om onmiddellijk deze controle uit te voeren.

Als de presentaties klaar zijn dan kunnen ze gepubliceerd worden. De filialen kunnen de presentaties van de centrale server afhalen en eventueel aanpassen.

In het huidige systeem is er op de centrale server windows server 2003 geïnstalleerd waarop een FTP-server draait. Lokale servers maken dus gebruik van het FTP-protocol om nieuwe presentaties binnen te halen. Een presentatie bestaat uit een XML-bestand met alle gegevens uit de databank en een ZIP-bestand met alle mediabestanden. Per presentatie is er een map op de centrale server met daarin een bestand dat aangeeft voor welk beeldscherm de presentatie bedoeld is. (Elke display unit heeft een naam en weet zo welke presentaties het mag tonen)
Dit systeem heeft als nadeel dat er momenteel slechts 1 login is voor alle filialen. Filialen die bijvoorbeeld zelfstandig worden, hebben nog steeds toegang tot alle presentaties.
Elk filiaal kan de presentaties (die soms vrij groot zijn) op elk moment downloaden. Om de belasting van de server beter te spreiden, zou een systeem met een soort wachtlijst beter zijn. De filialen zouden dan ook kunnen aangeven dat ze de presentatie 's nachts willen downloaden zodat het netwerkverkeer overdag niet te veel verstoord wordt.
In het nieuwe systeem moet het afhalen van de presentaties dus op een gestructureerde manier kunnen verlopen.

Er wordt nog steeds een FTP-server gebruikt.
Per winkel is er een map die beveiligd is met een gebruikersnaam en een wachtwoord. In deze map worden de presentaties en presentatieoverzichten van de winkel bewaard. De winkel zal in deze map ook een overzicht plaatsen van de status (OK of fout tijdens overdracht) van elke presentatie en heeft dus schrijfrecht in zijn eigen map.
Er is ook een gemeenschappelijke map die alle mediabestanden bevat.
Verschillende presentaties in verschillende winkels kunnen dus verwijzen naar dezelfde mediabestanden wat redundantie vermijdt.
Als een winkel sluit, moet enkel de map van die winkel verwijderd worden op de centrale.
In de vorige versie moest het wachtwoord veranderd worden voor alle andere winkels.
De service die in elke winkel draait, kan geconfigureerd worden vanuit de centrale. De beheerder kan bepalen tussen welke uren en om de hoeveel minuten de service moet controleren of er nieuwe presentaties zijn. In de service wordt het tijdsinterval verminderd of vermeerderd met een willekeurige waarde zodat de kans heel erg klein is dat alle winkels op hetzelfde moment beginnen downloaden. Deze willekeurige waarde wordt beperkt door een percentage dat eveneens ingesteld kan worden op de centrale.
Het is ook mogelijk om vanuit de centrale een winkel te verplichten om onmiddellijk te controleren of er nieuwe presentaties zijn. Dit pushsysteem is beveiligd zodat een hacker geen updateprocedure kan starten.
Er wordt ook gecontroleerd of de mediabestanden al aanwezig zijn in de winkel (omdat ze al eens gebruikt zijn in een andere presentatie) zodat er geen bandbreedte verspild wordt.
Alle acties die de service onderneemt, worden gelogd zodat het opsporen van fouten kinderspel is.

Deel 2. Software om de presentatie weer te geven (Adobe Flex)

De presentaties worden ingelezen door een programma op de display unit. Dit programma zal de data en configuratie van de presentatie ophalen van de lokale server en vervolgens de presentatie tonen op het scherm.

Tijdens de stage werd beslist dat de data gekopieerd moest worden naar elke display unit en dat er dus geen webserver mocht draaien in de winkel. Deze beslissing werd genomen door het POS-team en de systeembeheerder omdat het ophalen van data via een webserver veel te belastend zou zijn voor het netwerk zeker als men in de toekomst draadloze weegschalen zou gebruiken.

Voor de ontwikkeling zal Adobe Flex gebruikt worden:

'Adobe® Flex™ 2 is a complete, powerful application development solution for creating and delivering cross-platform rich Internet applications (RIAs) within the enterprise and across the web. It provides a modern, standards-based language and programming model that supports common design patterns and includes an Eclipse™ based development environment; advanced data services; and a fast, enterprise-class client runtime based on ubiquitous Adobe Flash® Player *software. Flex enables enterprises to create engaging, interactive, expressive applications that dramatically enhance user experience, increasing customer satisfaction and user productivity'*

In het huidige systeem wordt een flashapplicatie gebruikt om presentaties te tonen. Adobe Flex biedt het voordeel dat er op een meer gestructureerde (en OO) manier gewerkt kan worden dankzij de Macromedia Flex Markup Language (MXML) en ActionScript 3.0.

Alle display units hebben een naam en weten dus welke presentaties voor hun bedoeld zijn. De mediabestanden van de presentatie moeten 1 keer binnengehaald worden en nadien lokaal bewaard worden. Teksten, productinformatie, … zullen via Action Message Format vanuit Adobe Flex opgehaald worden.

Het programma om de presentaties af te spelen is ontwikkeld in Flex en wordt ingebed in Delphi op weegschalen, en in .NET op gewone display units. Alle bestanden worden naar elke display unit gekopieerd en er is dus geen nood aan Action Message Format om data op te halen. De presentaties worden bewaard als XML-bestanden die met behulp van E4X ingelezen worden.
Op de display units staat ook een klein programma waarmee men kan bepalen op welk scherm (het eerste of het tweede) de presentaties getoond moeten worden.

Probleemstellingen

Tijdens de (voorlopige) analyse van dit project kwam ik reeds een aantal problemen tegen.

- Dubbele producten

  Een bedrijf met verschillende filialen beschikt over een heel gamma producten. Een filiaal kan dezelfde producten verkopen als het hoofdfiliaal al dan niet met een andere prijs. Het is ook mogelijk dat een filiaal nog bijkomende producten aanbiedt die niet afkomstig zijn van de hoofdzetel.
  Er ontstaan vaak problemen als een filiaal een product toevoegt (bijvoorbeeld bij de promotiedia) en de hoofdzetel later hetzelfde product toevoegt. Als het filiaal de nieuwe presentatie ophaalt dan is er een product dat tweemaal voorkomt. Het product in het filiaal mag dan niet zomaar overschreven worden. Er kan geopteerd worden om per product bij te houden waar het werd aangemaakt, zo kan een filiaal zelf kiezen welke versie van het product het wenst te gebruiken.

Er werd beslist om de winkels geen rechten te geven in deze versie van het systeem en dus vervalt dit probleem.

- Verbinding tussen display unit, lokale server en centrale server

  Zoals reeds eerder besproken, wordt er nu een FTP-server gebruikt om presentaties beschikbaar te maken voor de verschillende filialen. Er moet onderzocht worden of er geen beter systeem bestaat (eventueel naast de FTP-server) om de verspreiding van presentaties gestructureerd te laten verlopen.
  De identificatie van de verschillende servers en display units moet ook onderzocht worden (vaste IP-adressen of DHCP)

  Er wordt nog steeds een FTP-server gebruikt om de presentaties en de mediabestanden vanuit de centrale te verspreiden naar de winkels. Elke winkel heeft zijn eigen FTP-account.
  De manier waarop display units in hun lokale netwerk geïdentificeerd worden, is van weinig belang. In de centrale moet men aan een scherm naast een logische naam ook een identificatie toekennen. Dit kan het IP-adres of de computernaam van de display unit zijn. De beheerder kan dus zelf kiezen hoe de schermen geïdentificeerd worden.
  De service in de winkel downloadt alle presentaties voor die winkel en verspreidt ze naar de juiste schermen.

- Nieuwe soorten dia's

  Het systeem moet zo ontwikkeld worden dat het eenvoudig is om later nieuwe soorten dia's aan te maken. Een nieuw soort dia vereist aanpassingen zowel in de software om presentaties te beheren als in de software om ze weer te geven. Er moet dus gezocht worden naar een soort formaat dat door beide programma's verstaan wordt.

  Presentaties worden in de presentatiebeheerder behandeld als objecten. Als men een presentatie bewaart dan wordt ze geserialiseerd naar een XML-bestand. In Flex worden deze XML-bestanden ingelezen met E4X.

- Data ophalen van de lokale server

  Het is de bedoeling dat de software die de presentaties weergeeft de data dynamisch kan ophalen. Vanuit een Adobe Flex front-end kan via AMF en een PHP back-end data uitgewisseld worden. Hoe dit juist in zijn werk gaat zal nog verder onderzocht moeten worden.

  Zoals ik reeds eerder heb vermeld, wordt er in dit systeem geen gebruik gemaakt van een webserver. De communicatie tussen Flex en een webserver via AMF werd wel onderzocht bij de aanvang van de stage.

Mogelijke uitbreidingen

- Volgende klant:
  Het nummer van de volgende klant die bediend kan worden, kan ook getoond worden op de presentatie.
  Als een verkoper op een toets drukt dan moet de lokale server aan de verschillende display units laten weten dat het nummer van de volgende klant getoond moet worden. Dit systeem moet dynamisch zijn: een supermarkt heeft bijvoorbeeld een beenhouwerij en een bakkerij die elk afzonderlijke volgnummers gebruiken.

  In Flex heb ik een aantal modules ontwikkeld die vanuit de centrale beheerd kunnen worden. De presentatiemodule toont de presentaties, de logomodule toont het logo van de winkel op de gewenste plaats en de volgende-klant-module toont

het nummer van de volgende klant. In de centrale kan de beheerder bepalen waar en hoe groot deze modules getoond moeten worden. De beheerder kan dus ontwerpen hoe een scherm eruit zal zien. Dit schermontwerp kan gebruikt worden voor één of meerdere schermen. Eén van de modules op zo'n schermontwerp toont het nummer van de volgende klant. Het nummer wordt verspreid door het POS-systeem en wordt vervolgens getoond door Flex. Als het nummer wijzigt, krijgt de klant een animatie te zien.

- Nieuwe soorten dia's:
  Bijvoorbeeld prijslijsten (broodje hesp----EUR 2.75 broodje kaas ----- EUR 2.75).

- Lichtkrant: lopende tekst onderaan het scherm.

- Koppeling RSS, weerbericht

  Elke uitbreiding kan later eenvoudig geïmplementeerd worden als nieuwe module of dia.

- Reclame toelaten:
  Een bedrijf betaalt om zijn reclame (foto) elke 10 min gedurende 15 sec. zichtbaar te zetten.

- Licentie inbouwen zodanig dat er niet kan gekopieerd worden.

  In .NET kan men in een *Setup Project* gebruikmaken van een zogenaamde productcode.

Technologieën

- .NET **(OK)**
- Adobe Flex **(OK)**
- **Communicatie tussen Flex en .NET**
- ActionScript **(OK)**
- AMF / PHP **(onderzocht maar niet geïmplementeerd)**
- **Windows Services**
- **XML-serialisatie**
- **GDI+**
- **ActiveX**

Literatuurlijst

Adobe. (2004). *Adobe - Flex General Discussion*. Opgeroepen op 26 februari 2008, van http://www.adobe.com/cfusion/webforums/forum/messageview.cfm?forumid=60&catid=585&threadid=1245655&highlight_key=y&keyword1=scale

Aid, I. (2007). *Installing PHP 5 on IIS in 5 simple steps*. Opgeroepen op 11 februari 2008, van http://www.iis-aid.com/articles/how_to_guides/installing_php_5_on_iis_in_5_simple_steps?page=0%2C1

Applying behaviors in ActionScript using styles. (2006). Opgeroepen op 19 februari 2008, van http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=behaviors_068_13.html

Balkan, A. (sd). *Adobe - Flex Quick Start Basics: Building components in ActionScript*. Opgeroepen op 15 februari 2008, van http://www.adobe.com/devnet/flex/quickstart/building_components_in_as/

Bebert, R. (2005). *I See Sharp: MySQL with C#*. Opgeroepen op 13 februari 2008, van http://iseesharp.blogspot.com/2005/09/mysql-with-c.html

Casario, M. (2007). *Flex Ssolutions: Essential Techniques For flex 2 And 3 Developers.* New York: friends of ED.

club, e. (2008). *Better documentation for AS3's XML class (e4x) « explorers' club*. Opgeroepen op 1 maart 2008, van http://jwopitz.wordpress.com/2008/02/27/better-documentation-for-as3s-xml-class-e4x/

Coenraets, C. (2003). *An overview of MXML: The Flex markup language*. Opgeroepen op 1 maart 2008, van http://www.adobe.com/devnet/flex/articles/paradigm.html

deHaan, P. (2007). *Detecting whether an image loaded successfully in Flex at Flex Examples*. Opgeroepen op 27 februari 2008, van http://blog.flexexamples.com/2007/11/04/detecting-whether-an-image-loaded-successfully-in-flex/#more-275

deHaan, P. (2008). *Setting effects with ActionScript in Flex at Flex Examples*. Opgeroepen op 10 maart 2008, van http://blog.flexexamples.com/2008/02/27/setting-effects-with-actionscript-in-flex/

Digital Signage. (2008). Opgeroepen op 2 mei 2008, van Disignage: http://www.disignage.nl/digitalsignage.php

Digital signage: van beweging tot interactie? (2007). Opgeroepen op 27 februari 2008, van http://www.digimedia.be/detail05nl.asp?Id=4469

dZine. (2008). *dzine - digital signage & mobile office solutions*. Opgeroepen op 1 maart 2008, van http://www.dzine.be/?page=displaystudio

E. Brown, C. (2007). *The Essential Guide to Flex 2 with ActionScript 3.0.* New York: friends of ED.

Eckel, B. (2007). *Creating Flex Components*. Opgeroepen op 15 februari 2008, van http://www.artima.com/weblogs/viewpost.jsp?thread=212818

Ent, P. (sd). *Adobe - Developer Center : Flex Component Basics – Part 1: Coding an Analog Clock*. Opgeroepen op 15 februari 2008, van http://www.adobe.com/devnet/flex/articles/creating_comp_print.html

Fierro, P. (2007). *Sorting XMLListCollection*. Opgeroepen op 29 februari 2008, van http://www.paulofierro.com/archives/460/

Flash-db. (2008). *Remoting AS3 examples (amfphp-Flash-Flex)*. Opgeroepen op 2008, van http://www.flash-db.com/Tutorials/helloAS3/helloAS3.php?page=5

FLEX Examples: Flex Effects. (2008). Opgeroepen op 25 februari 2008, van http://flexexamples.blogspot.com/search/label/Flex%20Effects

Herrington, J. (2007). *Introduction to Flex Using PHP*. Opgeroepen op 24 februari 2008, van http://www.onlamp.com/pub/a/onlamp/2007/07/19/introduction-to-flex-using-php.html

Hosseini, S. (2007). *Converting , extracting preview image in asp.net via ffmpeg*. Opgeroepen op 28 maart 2008, van http://www.codeproject.com/KB/aspnet/ffmpeg_csharp.aspx

HowtoForge. (2007). *Using Amfphp 1.9 with the Adobe Flex 2 SDK*. Opgeroepen op 12 februari 2008, van http://www.howtoforge.com/amfphp_adobe_flex2_sdk

Jacobs, S., & De Wegghеleire, K. (2008). *Foundation Flex for Developers: Data-Driven Applications with PHP, ASP.NET, ColdFusion, and LCDS.* New York: friends of ED.

Kazoun, C., & Lott, J. (2007). *Programming Flex 2.* Sebastopol: O'Reilly Media.

Koening, K. (2006). *Styling Flex 2 applications with Kai Koening*. Opgeroepen op 10 maart 2008, van https://admin.adobe.acrobat.com/_a200985228/p24244278/

Lott, J., Schall, D., & Peters, K. (2006). *ActionScript 3.0 Cookbook.* Sebastopol: O'Reilly Media.

Ma, A. (2001). *XML Serialization in C#*. Opgeroepen op 19 februari 2008, van http://www.dotnetjohn.com/articles.aspx?articleid=173

Minidxer. (2008). *sample code which guide you on how to communicate between Flex and Php*. Opgeroepen op 12 februari 2008, van http://ntt.cc/2008/02/03/sample-code-which-guide-you-on-how-to-communicate-between-flex-and-php.html

nashcontrol. (2004). *CodeProject: C# Rect Tracker. Free source code and programming help*. Opgeroepen op 15 april 2008, van http://www.codeproject.com/KB/miscctrl/CSharpRectTracker.aspx

onAir Tour Europe. (2008, 7 april). Brussel: Adobe.

Otuome, H., Gonzalez, O., & Charlton, C. (2008). *Advanced Flex Application Development:.* New York: friends of ED.

Pena, M. (2008). *Resize flex app swf in browser*. Opgeroepen op 19 februari 2008, van http://www.nabble.com/Resize-flex-app-swf-in-browser-td15278262.html

Perkings, T. (sd). *Online Tutorial Adobe Flex 2 Training Courses Tutorials*. Opgeroepen op 29 december 2007, van http://apex.vtc.com/adobe-flex.php

Phillips, B. (2007). *Using Flex 2 Custom Events and Public Functions When Creating Custom Components*. Opgeroepen op 25 maart 2008, van http://www.brucephillips.name/blog/index.cfm/2007/10/11/Using-Flex-2-Custom-Events-and-Public-Functions-When--Creating-Custom-Components

Raghu, N. (2007). *E4X « FLEXing My Muscle*. Opgeroepen op 29 februari 2008, van http://raghuonflex.wordpress.com/category/e4x/

S. Soper, D. (sd). *Custom Row and Column Drag and Drop Reordering Operations with the DataGridView*. Opgeroepen op 23 februari 2008, van http://www.danielsoper.com/programming/DataGridViewDragDropRowsColumns.aspx

solipsistic. (2007). *blog.layer2.org :: Use ffmpeg to extract first image out of FLV*. Opgeroepen op 28 maart 2008, van http://blog.layer2.org/2007/12/03/use-ffmpeg-to-extract-first-image-out-of-flv/

Systems, A. (2007). *Flex 2 Primitive Explorer*. Opgeroepen op 5 maart 2008, van http://www.flexibleexperiments.com/Flex/PrimitiveExplorer/Flex2PrimitiveExplorer.html

Tank, O. (2007). *E4X Quick Start Guide*. Opgeroepen op 1 maart 2008, van http://wso2.org/project/mashup/0.2/docs/e4xquickstart.html

TC. (2003). *Load and save objects to XML using serialization*. Opgeroepen op 19 februari 2008, van http://www.codeproject.com/KB/XML/xml_serializationasp.aspx

Technologies, W. R. (2008). *Movie Theaters Digital Signage Software*. Opgeroepen op 2 maart 2008, van http://www.wirelessronin.com/2.33_theaters.html

Tretola, R., Barber, S., & Erickson, R. (2007). *Professional Adobe Flex 2 .* Indianapolis: Wiley Publishing.

Webster, S. (2008). *Foundation ActionScript 3.0 with Flash CS3 and Flex.* New York: friends of ED.

XML Serialization Using C#. (2004). Opgeroepen op 19 februari 2008, van http://www.dotnetjohn.com/articles.aspx?articleid=173

Tijdens de onderzoeksfase heb ik voorral gebruik gemaakt van de boeken die vermeld staan in deze literatuurlijst. Gedurende de realisatiefase was het internet mijn primaire bron.