“HandataLink3000”

Afstudeerverslag

Student: Roland van der Plas
Studentnummer: 99005571

Opleidingsinstituut: Haagse Hogeschool
Opleiding: Informatievoorziening en Informatie Technologie
Differentiatie: Ontwikkeling van Software en Technische Infrastructuren

Afstudeerbedrijf: Handata B.V.
Afstudeerperiode: 2004-2.1 - 30 augustus 2004 – 14 januari 2005
Bedrijfsmentoren: Dhr. G. Menkveld
Dhr. P.J. van de Velde

Examinatoren: Dhr. A. Andrioli
Dhr. A. van der Molen

Referaat

van der Plas, R., AFSTUDEERVERSLAG, Katwijk, Handata B.V., januari 2005

Het afstudeerverslag is een beschrijving van mijn werkzaamheden bij Handata B.V.. Dit verslag is een onderdeel van mijn afstuderen aan de Haagse Hogeschool sector Informatica in Den Haag.

Het doel van dit verslag is het inzicht geven in de behaalde leerwinst en uitgevoerde werkzaamheden bij Handata B.V..

Een groot deel van dit verslag is gewijd aan de uitgevoerde werkzaamheden, waarbij aandacht wordt besteedt aan de afstudeeropdracht, algemene en specifieke aspecten, aanpak van het werk, problemen / alternatieven, oplossingen en evaluatie.

DESCRIPTOREN

Afstudeerverslag
Afstuderen
Borland Delphi
Handata B.V.
Iteractive Application Design (IAD)
Unified Modelling Language (UML)

Voorwoord

Als student van de Haagse Hogeschool sector Informatica heb ik mijn afstudeerstage gevolgd bij Handata B.V. te Katwijk. Dit verslag beschrijft de uitgevoerde werkzaamheden tijdens de stage.

Op deze plek wil ik iedereen bedanken die op enige wijze heeft geholpen aan de uitvoering van mijn afstudeeropdracht. Tevens wil ik iedereen bedanken die ervoor heeft gezorgd dat het een gezellige en leerzame periode is geweest.

Een aantal mensen wil ik in het bijzonder bedanken:

- Gerrit Menkveld en Peter Jan van de Velde, als bedrijfsmentoren, voor het aanbieden van de afstudeerplaats en de begeleiding tijdens mijn afstudeerperiode.
- De heren van der Molen en Andrioli, als respectievelijk examinator en assessor, voor de begeleiding vanuit school. In het bijzonder met hun kritische beoordeling van mijn conceptverslag.

Roland van der Plas,
Katwijk, januari 2005

Inhoudsopgave

1 Inleiding

Dit afstudeerverslag geeft een overzicht van de werkzaamheden tijdens mijn afstudeerperiode bij Handata B.V.. Het afstudeerverslag is een onderdeel van het afstuderen aan de opleiding Informatie Voorziening en Informatie Technologie aan de sector Informatica van de Haagse Hogeschool.

Hoofdstuk 2 beschrijft de oriëntatiefase. In dit hoofdstuk wordt de aanloop naar het project beschreven. In dit hoofdstuk kan informatie over de organisatie en de afstudeeropdracht worden teruggevonden.

In hoofdstuk 3 worden de uitgevoerde werkzaamheden tijdens de definitiestudiefase beschreven. In de fase definitiestudie worden de doelen van het systeem geanalyseerd.

Hoofdstuk 4 beschrijft de pilotontwikkelfase. In dit hoofdstuk wordt, per uitgevoerde pilot, beschreven hoe de pilot is ontworpen, gerealiseerd en getest.

Hoofdstuk 5 tenslotte, beschrijft de evaluatie van de afstudeeropdracht. Hierin worden de opgeleverde producten en het uitgevoerde proces geëvalueerd.

Deel I:

2 Oriëntatie

Dit hoofdstuk biedt informatie over het afstudeerbedrijf en de opdracht. Als eerste wordt er een korte schets gegeven van de organisatie waar binnen de opdracht is uitgevoerd. Vervolgens zal een beschrijving van de opdracht worden gegeven.

2.1 Handata B.V.

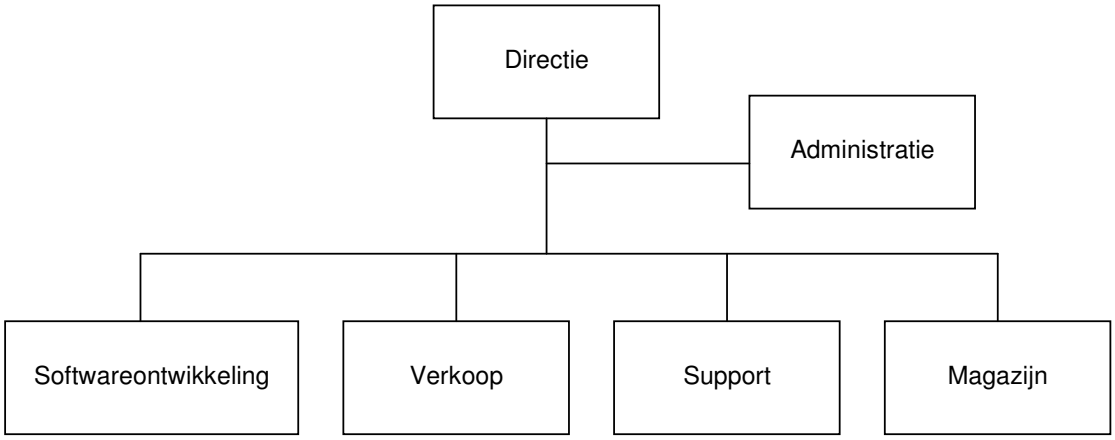
Handata B.V is opgericht op 1 december 1998. Handata bestond op dat moment uit drie enthousiaste mensen, die al enkele jaren werkzaam waren bij een ander automatiseringsbedrijf. Omdat dit bedrijf niet de groeimogelijkheden voor de mobiele markt wilde benutten, besloten de drie om een eigen bedrijf te starten. Dit nieuwe bedrijf, Handata B.V., heeft als specialiteit 'mobiele oplossingen'. Concreet betekent dit dat Handata zich bezig houdt met het bouwen van oplossingen op basis van mobiele apparatuur.

Handata is geselecteerd als business partner van Psion, een leverancier van handhelds. Via het partner programma van Psion zijn een select aantal leveranciers uitgekozen die diensten en producten leveren in combinatie met Psion producten.

De door Handata aangereikte oplossingen bieden een scala aan mogelijkheden, zoals het registreren van orders op beurzen, registratie van inkomende en uitgaande goederen, objectregistratie, magazijnapplicaties, routeverkoop en tijdregistratie. Opvallend is de marketingbenadering van het bedrijf. De mobiele oplossingen zijn geknoopt aan de computerprogramma's van grote softwareleveranciers, zoals bijvoorbeeld Unit4 Agresso en Muis Software. Zo kunnen klanten vanuit de bestaande automatiseringssystemen de oplossingen direct hanteren.

Mobiele oplossingen zijn op veel gebieden te verwezenlijken. Handata heeft bijvoorbeeld verschillende verkoopsystemen, enquête systemen, inventarisatie systemen en ordersystemen. Deze systemen bestaan uit een applicatie op de PC (HandataLink) en een applicatie op een handheld. HandataLink verzamelt de stamgegevens van het logistiek administratieve systeem van de klant (onder andere klant-, artikel-, prijsgegevens) en bewerkt deze voor gebruik in de handheld. Vervolgens communiceren de handheld en HandataLink (onder andere d.m.v. modem, GSM, kabel) en wisselen informatie uit. Het kan zijn dat er stamgegevens naar de handheld gaan of dat er gegevens vanuit de handheld naar HandataLink komen. Als HandataLink gegevens (onder andere orders) heeft ontvangen worden deze verwerkt zodat ze weer kunnen worden aangeboden aan het logistiek administratieve systeem van de klant.

Handata B.V. heeft op dit moment zes mensen in dienst. Vier vaste medewerkers en twee parttime medewerkers. De organisatiestructuur ziet er als volgt uit. Allereerst is er de directie, bestaande uit twee man. Daaronder bevinden zich de verschillende afdelingen.



Tijdens mijn afstuderen ben ik werkzaam geweest op de afdeling softwareontwikkeling. Op de afdeling softwareontwikkeling wordt de software voor zowel HandataLink als de handhelds ontwikkeld.

2.2 Opdracht HandataLink3000

In deze paragraaf wordt de opdracht beschreven. Het doel van deze paragraaf is om inzicht te geven in de inhoud van de opdracht. Ik ben met Handata in contact gekomen nadat ik een reportage op de lokale televisie had gezien. De reportage ging over het feit dat Handata winnaar was geworden van de Katwijkse Ondernemersprijs 2001 – 2003. Via een e-mail bericht heb ik contact opgenomen met het bedrijf en werd ik uitgenodigd voor een gesprek. Dit gesprek was een kennismakingsgesprek. Bij het tweede gesprek zijn de eerste ideeën met betrekking tot tot de opdracht besproken. Toen werd mij ook aangeboden om voor aanvang van de afstudeerperiode bij Handata te beginnen zodat ik me kon inwerken in Delphi 8. In deze periode is ook de opdrachtomschrijving verder uitgewerkt. De volgende paragrafen worden ook beschreven in het algemene plan van aanpak. Voor het gehele algemene plan van aanpak wordt verwezen naar de externe bijlage Plan van aanpak algemeen.

2.2.1 Aanleiding

HandataLink is het PC programma dat:

- de communicatie verzorgt tussen de applicatie van de klant en de Psion Workabout [1];
- de gegevens uit het systeem van de klant inleest en converteert voor gebruik met de Workabout;
- na gebruik van de WorkAbout, de gegevens terugleest om vervolgens deze gegevens te exporteren naar het systeem van de klant;
- afhankelijk van de versie, diverse business functies regelt, zoals:
    - beheer van de door de klant gebruikte valuta's met koersen
    - afdrukken van overzichten onder andere (foto)orders, voorraden en dagtotalen
    - afdrukken van barcode etiketten
    - communicatie via modem of mobiele telefoon
    - orders van nieuwe klanten koppelen aan een nieuwe klantcode

[1] Een WorkAbout is een handheld type van het merk Psion

2.2.2 Probleemstelling

Handata B.V. heeft besloten dat de huidige HandataLink opnieuw ontworpen moet worden voor een objectgeoriënteerde (Microsoft .NET) omgeving. Dit betekent dat er een nieuwe architectuur moet worden ontworpen. Aangezien er vrijwel geen technische documentatie van de huidige HandataLink aanwezig is, zullen de bestaande functies en / of wensen van de opdrachtgever moeten worden gedocumenteerd. Handata B.V. verwacht dat deze omzetting veel voordeel kan opleveren wanneer er een HandataLink applicatie moet worden geïmplementeerd voor een nieuwe klant of een bestaande HandataLink moet worden uitgebreid met een nieuwe functionaliteit. Het codewoord voor dit project is HandataLink3000".

2.2.3 Doelstelling

Het doel van de afstudeeropdracht is om aan de hand van de verschillende pilots, verschillende functionaliteiten van een HandataLink Order Systeem te verkrijgen, die functioneren in een nieuwe, object georiënteerde omgeving. Het moet aan de hand van de verschillende pilots mogelijk zijn, om nieuwe applicaties te ontwikkelen voor nieuwe klanten of bestaande HandataLinks uit te breiden met nieuwe functionaliteiten.

De objectgeoriënteerde opzet van HandataLink zal in ieder geval het volgende moeten bevatten:

- importfunctionaliteit: het inlezen van stamgegevens vanuit de back-end van de klant;
- exportfunctionaliteit: gegevens die afkomstig zijn van de handheld moeten kunnen worden geëxporteerd voor gebruik in de back-end van de klant;
- synchronisatiefunctionaliteit met een handheld: data van HandataLink beschikbaar maken voor gebruik in de handheld en omgekeerd.
- functionaliteit voor het beheren van de handheldinstellingen, het betreft hier het:
    - toevoegen
    - wijzigen
    - verwijderen
- GUI waarin bovenstaande functionaliteiten kunnen worden aangeroepen;

2.3 Gebruikte methode, technieken en software

In deze paragraaf worden de belangrijkste methode, technieken en softwarepakketten beschreven die gedurende het project zijn gebruikt. Het doel van deze beschrijving is inzicht te geven in wat de belangrijkste gebruikte methode, technieken en software inhouden en waarom er voor is gekozen.

2.3.1 IAD

In de aanloop tot de opdracht heb ik met Handata gesproken over de te hanteren ontwikkelmethode. Vanuit Handata was er geen bepaalde voorkeur voor een ontwikkelmethode en ik werd vrijgelaten in de keuze. Ik heb tijdens projecten voor school en mijn stage gebruik gemaakt van SDM I. Tijdens de modulen SO-07 "methoden en technieken van systeemontwikkeling" en IP-03 (Integrerend Practicum) heb ik kennis gemaakt met IAD.

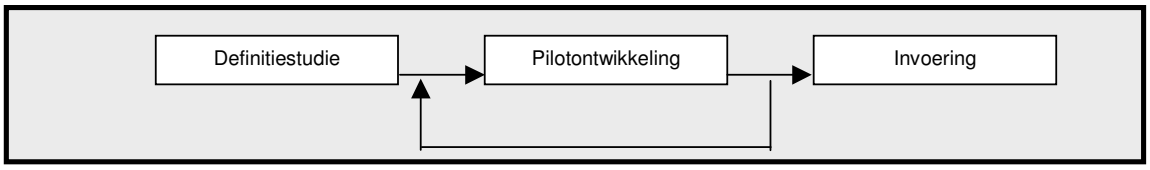
Ik heb voor IAD gekozen. Het nauw betrekken van de opdrachtgever bij het project spreekt mij aan. In projecten, waarbij SDM I als methodiek wordt gebruikt, is deze betrokkenheid van de opdrachtgever vaak minder, waardoor het voor kan komen dat het eindresultaat niet precies is wat de opdrachtgever voor ogen had. Dit risico wordt verkleind als de opdrachtgever nauw betrokken is bij het project.

Iterative Application Development (IAD) is een uitgebreide set van activiteiten om een systeem te ontwikkelen. Iterative, vertaalt iteratief, wil zeggen dat de ontwikkelfasen verschillende malen worden doorlopen. Puntsgewijs ziet een ontwikkeltraject volgens IAD er als volgt uit:

- Definitiestudie
- Pilotontwikkeling
- Invoering

Strategiekeuze

Binnen IAD zijn er verschillende strategieën, waaruit de ontwikkelaar kan kiezen.
Tijdens de ontwikkelscenario workshop uit de fase definitiestudie heb ik in overleg met de opdrachtgever gekozen voor de strategie 'incrementeel ontwikkelen' (RAD). Dit is een strategie, waarin eerst de systeemeisen in de definitiefase in kaart worden gebracht. In de tweede fase wordt het systeem d.m.v. een aantal iteraties van de pilotontwikkelfase gerealiseerd. De invoering vindt vervolgens in één keer plaats.



Figuur: IAD Ontwikkeltraject

Ik heb voor deze strategie gekozen, omdat vooraf het systeem goed kon worden gespecificeerd aangezien de functionaliteiten al aanwezig zijn in de huidige HandataLink. Het vooraf kunnen specificeren van het systeem is een kenmerk van deze strategie. Daarom was het niet nodig om voor elke pilot opnieuw de definitiefase te doorlopen. Het iteratief doorlopen van de pilotontwikkelfase is naar mijn mening wel zinvol. Aparte systeemdelen worden in dezelfde iteratie ontworpen en gebouwd. Hierdoor wordt steeds aan één aparte bouwsteen gewerkt. Ik verwacht dat dit leidt tot betere software, omdat meteen na het ontwerp de bouwsteen wordt gerealiseerd in plaats van dat eerst moet worden nagedacht over een andere bouwsteen. De fase invoering wordt niet doorlopen. Er wordt namelijk geen systeem opgeleverd bij een klant. Uiteraard worden de producten wel opgeleverd, maar ik wil hierbij niet spreken van een specifieke fase invoering.

2.3.2 UML

UML staat voor Unified Modelling Language. UML is onder andere te gebruiken in combinatie met de ontwikkelmethode IAD. Een IAD traject is vaak iteratief. Bij een iteratief proces kunnen in een laat stadium van het ontwikkeltraject extra functionaliteiten worden toegevoegd of kan het systeemconcept worden aangepast. Bij klassieke modelleringtechnieken, zoals Yourdon, kan dit problemen opleveren. Als een wijziging moet worden doorgevoerd, moeten vaak alle diagrammen uit een klassieke modelleringtechniek worden herschreven om de consistentie te bewaren. Tijdens projecten gedurende de opleiding aan de Haagse Hogeschool heb ik ervaren dat dit veel werk met zich mee brengt. Het aanpassen van een diagram in UML gaat vaak veel sneller dan bij een klassieke modelleringtechniek. Dit omdat diagrammen vaak niet opnieuw dienen te worden

geschreven. Vaak is een kleine toevoeging aan het diagram voldoende. UML is onderwezen tijdens de module SO-08. Ik heb onderdelen uit UML gebruikt tijdens mijn stage.

In de ontwerpfase van de applicatie kan met UML een groot aantal diagrammen worden gebruikt. UML laat de ontwikkelaar vrij in het kiezen van de te gebruiken diagrammen. Veel gebruikte diagrammen zijn:

- Use case diagram. Dit diagram toont hoe het systeem kan worden gebruikt door externe entiteiten zoals menselijke gebruikers.
- Klassediagram. Toont de statische structuur van het softwaresysteem, weergegeven als klassen en relaties.
- Sequencediagram. Toont de volgorde in tijd van de boodschappen die in het systeem worden verstuurd en ontvangen.

2.3.3 Borland Delphi

Borland Delphi is een programmeeromgeving dat in veel bedrijven wordt gebruikt. Voor de opdracht wordt gebruik gemaakt van Delphi, omdat dit de standaard ontwikkelomgeving bij Handata is. Binnen Handata wordt gebruik gemaakt van Delphi 5 voor de huidige HandataLink en Delphi 8 for .Net / Delphi 2005 voor de objectgeoriënteerde HandataLink.

Delphi is gebaseerd op de programmeertaal Borland Object Pascal. Object Pascal houdt in dat het een objectgeoriënteerde taal is. Delphi is een visuele ontwikkelomgeving. Dat wil zeggen dat het ontwerpen van de applicatie door middel van het drag-en-drop principe gaat. Standaard zijn er al een aantal componenten beschikbaar. Deze zijn opgeslagen in de Visual Component Library (VCL). Een component kan bijvoorbeeld een knop of een tekstveld zijn. De VCL kan worden aangevuld met nieuwe componenten. Nieuwe componenten kunnen zelf worden gemaakt of via externe bronnen worden geïnstalleerd.

Ik heb zelf al eerder ervaring opgedaan met deze ontwikkelomgeving. Tijdens mijn stage (Delphi 5) en tijdens de module IP-03 (Delphi 6).

2.4 Werkzaamheden, planning en op te leveren producten

In deze paragraaf worden de werkzaamheden beschreven die worden uitgevoerd tijdens de opdracht. Het doel hiervan is een tijdschatting voor het project te kunnen maken.

2.4.1 Te verrichten werkzaamheden

De, door mij, uit te voeren activiteiten kunnen worden onderverdeeld in drie fasen, namelijk inventarisatie, definitiestudie en pilotontwikkeling. Hieronder volgt een opsomming van de activiteiten per fase. Ik ben tot deze opsomming gekomen met behulp van het boek “IAD, Het evolutionair ontwikkelen van informatiesystemen”, geschreven door R.J.H. Tolido. Per fase uit IAD heb ik bekeken welke activiteiten van belang zijn voor het project. Dit zijn de volgende activiteiten:

- Inventarisatie:
  - Stel globaal plan van aanpak op
- Definitiestudie:
  - Stel plan van aanpak definitiestudie op

- Definieer ontwikkelscenario
- Bereid pilotplan workshop voor
- Definieer systeemeisen
- Bepaal systeemconcept
- Beschouw technische structuur
- Stel pilotplan op
- Schrijf rapport definitiestudie

- Pilotontwikkeling:
  - Stel plan van aanpak op
  - Bereid pilotontwerp workshop voor
  - Specificeer globaal functionele structuur pilot
  - Specificeer globaal technische structuur pilot
  - Stel pilotontwikkelplan op
  - Ontwerp software bouweenheden
  - Bouw software bouweenheden
  - Ontwerp handmatige procedures pilot
  - Integreer bouweenheden
  - Beoordeel en test pilot

2.4.2 Planning

Met behulp van de beschreven uit te voeren activiteiten, heb ik de projectplanning opgesteld. De planning wordt beschreven in onderstaand diagram. De planning van de verschillende pilots heb ik op opgesteld na het opstellen van het pilotplan in de fase definitiestudie.

Week / Onderdeel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Opstellen algemeen p.v.a.	■																	
Definitiestudie	■	■	■															
Pilotontwikkeling (iteratief)				■	■	■	■	■	■	■	■	■	■	■	■	■	■	
Pilot 1: Importeren				■	■	■												
Pilot 2: Synchroniseren							■	■	■									
Pilot 3: Exporteren										■	■							
Pilot 4: Beheren handheldinstellingen												■	■	■				
Pilot 5: GUI															■	■		
Pilot 6: Overig																	■	
Verslaglegging HHS	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

Figuur: Planning

2.4.3 Op te leveren producten

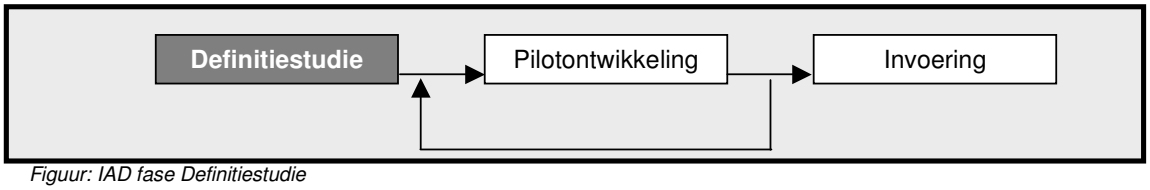
Tijdens het project zullen verschillende mijlpaalproducten worden opgeleverd. De volgende mijlpaalproducten zullen worden opgeleverd:

- Plan van aanpak algemeen;
- Rapport definitiestudie;
- Pilotontwikkelrapport per pilot, inclusief:
  - Plan van aanpak;
  - Pilotontwerp;
  - Testrapport;
- Pilots (realisatie van het pilotontwerp per pilot)

Deel II:

3 Uitgevoerde activiteiten definitiestudie

Zoals eerder beschreven heb ik voor de ontwikkelmethode IAD gekozen om het eindproduct te kunnen realiseren. De eerste fase uit IAD is de definitiestudie. In de fase definitiestudie worden de doelen van het systeem geanalyseerd. Dit in nauwe samenwerking met de opdrachtgever. De fase definitiestudie wordt slechts eenmaal doorlopen.



Figuur: IAD fase Definitiestudie

IAD is geschikt voor grote en kleinere projecten. IAD kent twee workshops tijdens de definitiestudie, de ontwikkelscenario workshop en de pilotplan workshop. In de pilotplan workshop zitten bepaalde activiteiten die voor dit project niet relevant zijn, te weten:

- Evalueer pilot. De fase definitiestudie wordt slechts eenmaal uitgevoerd. Voorafgaande aan de fase definitiestudie wordt geen fase pilotontwikkeling uitgevoerd. Er hoeft dus geen pilot te worden geëvalueerd;
- Beschouw organisatorische inrichting. Er worden geen veranderingen in de organisatie verwacht.

De overige activiteiten uit de pilotplan workshop heb ik wel uitgevoerd. Wegens de enigszins beperkte tijd van de opdrachtgever heb ik besloten om beide workshops tegelijk te houden. Dit heb ik gedaan nadat ik de pilotplan workshop had voorbereid. De keuze voor het ontwikkelscenario heb ik reeds beschreven in § 2.3.1 IAD.

3.1 Opstellen plan van aanpak

De fase definitiestudie ben ik begonnen met het opstellen van het plan van aanpak. In het plan van aanpak beschrijf ik o.a het doel, de reikwijdte, de op te leveren producten, de projectstructuur en de planning voor de definitiestudie. Ik heb invulling aan deze zaken kunnen geven na het uitvoeren van de workshops. Het doel van het plan van aanpak is om duidelijke afspraken vast te leggen over de inhoud en omvang van de op te leveren producten en de daarvoor uit te voeren werkzaamheden. De reikwijdte van de definitiestudie omvatte het gehele project. Aangezien de gekozen iteratiestrategie incrementeel ontwikkelen is.

De fase definitiestudie dient de volgende resultaten te bevatten:

- In overleg met de opdrachtgever dient er overeenstemming te worden bereikt over het te volgen ontwikkelscenario;
- De globale systeemeisen dienen te worden gedefinieerd;
- Het systeemconcept dient te worden bepaald;
- De technische structuur dient te worden beschouwd;
- Het pilotplan dient te worden opgesteld;

Tot deze resultaten is gekomen met behulp van het boek van IAD van Tolido [2].

[2] R.J.H. Tolido, IAD Het evolutionair ontwikkelen van informatiesystemen.

Ik heb per activiteit uit de definitiestudie tijd ingepland. Ik heb dit gedaan door een schatting te maken van de activiteit en de benodigde tijd. Deze planning wordt beschreven in de externe bijlage Rapport Definitiestudie.

3.2 Voorbereiden pilotplan workshop

Het voorbereiden van de pilotplan workshop houdt in:

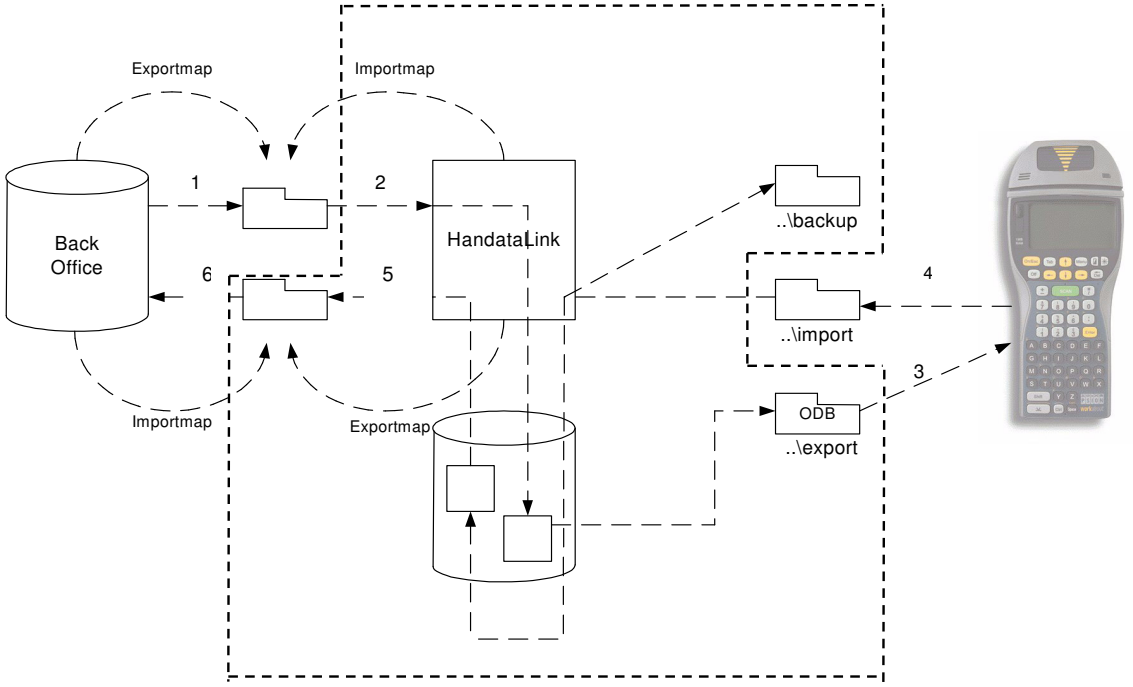
- het organiseren van de workshop zelf;
- het voorbereiden van de onderwerpen die tijdens de workshop aan de orde moeten komen.

3.2.1 Organiseren van de pilotplan workshop

IAD beschrijft een workshop, waarin een aantal betrokkenen worden uitgenodigd. Deze betrokkenen denken mee over het te bouwen systeem. In het huidige project zijn er slechts twee betrokkenen, namelijk Gerrit Menkveld, nader opdrachtgever genoemd, en mijzelf. In overleg met de opdrachtgever is een datum gekozen om de workshop te houden. De workshop is gehouden in de vergaderruimte van Handata. Het organiseren van de pilotontwerp workshops is op een gelijke manier uitgevoerd.

3.2.2 Voorbereiden van de te bespreken onderwerpen

Als voorbereiding op de onderwerpen die aan de orde zouden komen, heb ik enkele bestaande versies van een HandataLink Order Systeem bekeken. Door de menu's van deze applicaties af te lopen en de code te bekijken, kon ik me een beeld vormen van de verschillende functionaliteiten van HandataLink. Hieronder volgt een visualisering van de globale werking van een HandataLink applicatie. De verantwoordelijkheid van een HandataLink applicatie ligt binnen de doorbroken weergegeven rechthoek.



(1) Het back office systeem exporteert stamgegevens voor HandataLink
(2) HandataLink importeert deze stamgegevens
(3) De stamgegevens worden geconverteerd, zodat ze kunnen worden gebruikt in een handheld
(4) Handheld exporteert gegevens voor HandataLink. HandataLink maakt een back-up van deze gegevens en importeert ze.
(5) HandataLink exporteert gegevens voor back office.
(6) Back office systeem importeert deze gegevens.

Figuur: Globale werking HandataLink

Tolido beschrijft een initiële lijst van systeemeisen als een belangrijk hulpmiddel bij het opstellen van een adequate definitie van de systeemeisen. Ik heb met behulp van de verschillende HandataLink applicaties de volgende initiële lijst van systeemeisen op kunnen stellen:

- Het systeem moet gegevens kunnen importeren vanuit de back-end (database) van de klant;
- Het systeem moet gegevens kunnen synchroniseren met de handheld;
- Het systeem moet gegevens kunnen exporteren voor gebruik in de back-end van de klant;
- Het systeem moet, na import vanuit de handheld, een back-up van de geïmporteerde gegevens maken;
- Het systeem moet de gebruiker de mogelijkheid bieden een back-up te herstellen;
- Het systeem moet een logboek bijhouden met betrekking tot de handelingen van de gebruiker binnen het systeem;
- Het systeem moet de gebruiker de mogelijkheid bieden de handheld-instellingen te beheren;
- Het systeem moet de gebruiker de mogelijkheid bieden de bestandslocaties van de import- en exportbestanden met betrekking tot de back-end van de klant, te beheren;
- Het systeem moet orders van nieuwe klanten kunnen beheren;
- Het systeem moet de gebruiker de mogelijkheid bieden om de modeminstellingen te beheren;

Figuur: Initiële lijst van systeemeisen

[3] R.J.H. Tolido. Auteur van "IAD Het evolutionair ontwikkelen van informatiesystemen".

Aan de hand van deze systeemeisenlijst heb ik globale use cases geschreven, die ik tijdens de pilotplan workshop heb voorgelegd aan de opdrachtgever.

3.3 Uitvoeren pilotplan workshop

De pilotplan workshop heeft een aantal belangrijke doelen, te weten:

- vaststellen van de systeemeisen;
- bepalen systeemconcept;
- opstellen van het pilotplan

3.3.1 Vaststellen systeemeisen

Na het opstellen van de systeemeisenlijst stond het prioriteren van de eisen centraal. Het aangeven van prioriteiten is een belangrijke activiteit, omdat hiermee kan worden aangegeven welke onderdelen zeker in het systeem komen en welke onderdelen eventueel worden opgenomen. Tijdens het overleg heeft de opdrachtgever aangegeven welke eisen belangrijk waren en welke minder belangrijk. Hieruit is de volgende geprioriteerde lijst ontstaan.

Basis
HandataLink moet stamgegevens vanuit CSV en vaste lengte bestanden kunnen importeren.
HandataLink moet deze stamgegevens kunnen converteren en exporteren naar ODB bestanden voor gebruik in de WorkAbout.
HandataLink moet ODB bestanden met orderregels, die zijn opgesteld door de WorkAbout, kunnen converteren en importeren.
HandataLink moet de Orders kunnen exporteren naar een vaste lengte bestand.
Binnen HandataLink moet het mogelijk zijn om gegevens van de WorkAbout, die essentieel zijn voor de communicatie tussen HandataLink en WorkAbout, toe te voegen, te wijzigen of te verwijderen.
De functionaliteiten moeten kunnen worden aangeroepen vanuit een GUI.
Comfort
Na het importeren van orderbestanden moet HandataLink een back-up bestand van het orderbestand aanmaken.
Back-up bestanden moeten kunnen worden geïmporteerd in HandataLink.
De bestandslocaties van de import en export bestanden kunnen worden beheerd.
Modem instellingen kunnen worden beheerd.
Luxe
HandataLink houdt een logboek bij met betrekking tot de handelingen die de gebruiker heeft uitgevoerd.
Het is mogelijk om orders van nieuwe klanten te beheren.

Figuur: Geprioriteerde lijst van systeemeisen

3.3.2 Opstellen systeemconcept

Het systeemconcept heb ik opgesteld aan de hand van use cases en een klassediagram. Het systeemconcept geeft een eerste omschrijving van het uiteindelijk te bouwen systeem.

Use cases

Ik heb use cases opgesteld om de functionaliteiten van het te bouwen systeem te beschrijven. De use cases beschrijven hoe de toekomstige gebruiker met het systeem kan

omgaan. Met de use cases wordt een globaal beeld geschetst op welke events het systeem moet reageren en wat de reactie van het systeem moet zijn. De events zijn afkomstig van de actor HandataLink gebruiker.

Hier volgt de opgestelde use case voor de importfunctionaliteit van stamgegevens binnen HandataLink:

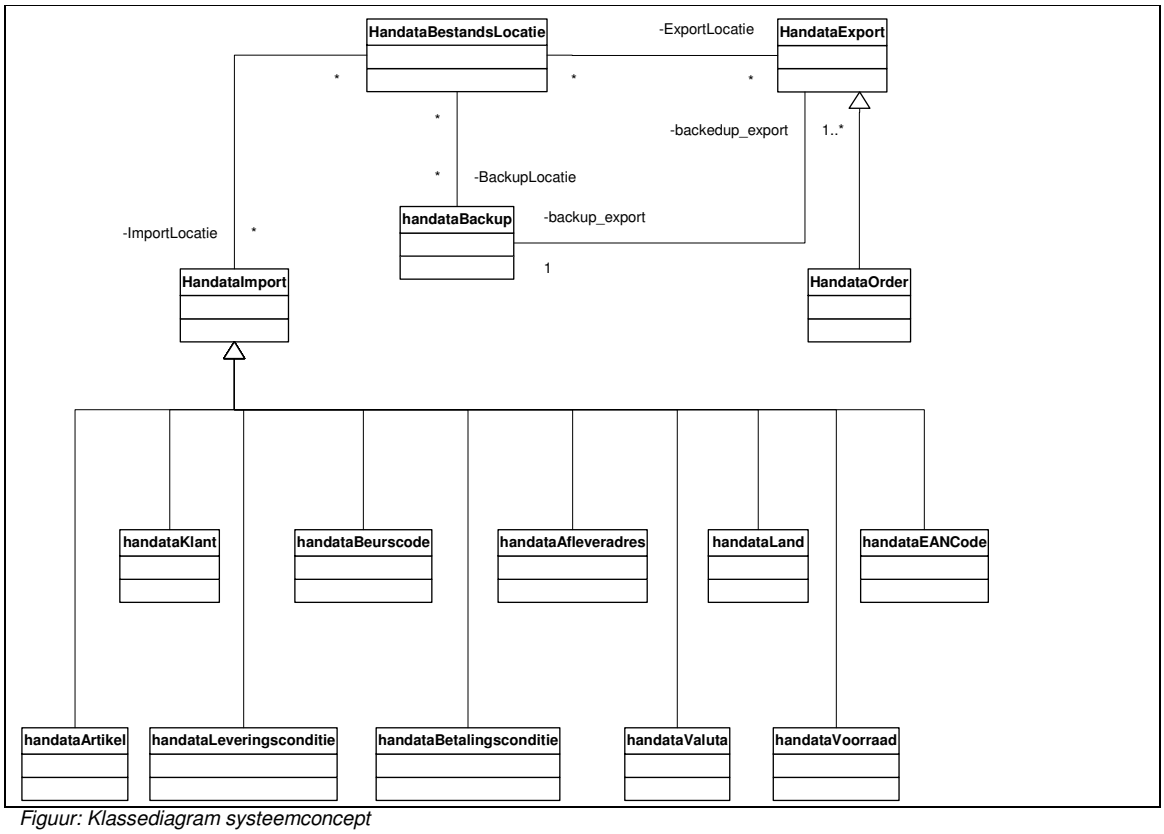
Naam	Stamgegevens importeren
Samenvatting	Er worden stamgegevens geïmporteerd vanuit back-end van de klant. De stamgegevens worden klaargezet voor communicatie met handheld.
Actoren	HandataLink gebruiker.
Aannamen	Systeem is gestart.
Beschrijving	(1) Gebruiker maakt aan het systeem bekend dat er gegevens moeten worden geïmporteerd vanuit de back-end van de klant. (2) Het systeem controleert of er importgegevens aanwezig zijn. Als dit het geval is worden de gegevens geïmporteerd. (3) De geïmporteerde gegevens worden geconverteerd en klaargezet voor gebruik met handheld.
Uitzonderingen	[Geen importgegevens] Als er na controle blijkt dat er geen importgegevens beschikbaar zijn, zal het systeem hiervan een melding geven aan de gebruiker.
Resultaat	De importgegevens zijn geïmporteerd en klaargezet voor gebruik met handheld.

Figuur: Use case beschrijving Stamgegevens importeren

De use cases heb ik op kunnen stellen a.d.h.v. de globale use cases die ik tijdens het voorbereiden van de workshop heb opgesteld. Tijdens de workshop heb ik samen met de opdrachtgever deze use cases bekeken en aangepast waar dit nodig bleek. Door de verschillende opgestelde use cases heb ik me een goed beeld kunnen vormen hoe HandataLink moet reageren op input van de gebruiker. Deze use cases beschrijven globaal de verschillende functionaliteiten. Tijdens de fase pilotontwikkeling worden deze verder gedetailleerd. De verschillende use cases heb ik samengevoegd in een use case diagram.

Klassediagram

Waar de use cases het gedrag van een systeem beschrijven, wordt het klassediagram gebruikt om de structuur van een systeem te beschrijven. Het klassediagram dat ik heb opgesteld beperkt zich tot de eerste vier eisen uit de prioriteitenlijst. Ik heb dit gedaan omdat dit de systeemeisen met de hoogste prioriteit zijn. Het klassediagram dat ik heb opgesteld voor het systeemconcept ziet er als volgt uit::



Figuur: Klassediagram systeemconcept

Om tot het klassediagram te komen heb ik eerste de globale klassen beschreven. Deze klassen zijn verkregen door de mogelijke klassen uit de systeemeisen lijst te noteren. Dit volgens de werkwijze die UML voorschrijft. De lijst van kandidaatklassen is vervolgens gefilterd zodat alleen de klassen overblijven die in het klassediagram beschreven zijn.

Naast de systeemeisen lijst heb ik ook gekeken naar welke gegevens door HandataLink worden geïmporteerd en geëxporteerd. Deze gegevens komen uit de verschillende import- en exportbestanden. Hieruit zijn de verschillende subklassen van HandataImport en HandataExport voortgekomen. Dit klassediagram is niet volledig. Dat is ook niet noodzakelijk aangezien het om een concept van het systeem gaat. Het klassediagram zal tijdens de pilotontwikkeling verder worden gedetailleerd / uitgebreid.

3.3.3 Opstellen pilotplan

In het pilotplan worden de verschillende functionaliteiten ingedeeld in pilots. Een pilot kan gezien worden als bouwsteen. Alle pilots bij elkaar vormen het systeem. Het doel van het pilotplan is het indelen van de eisen in pilots, zodat het verder te volgen ontwikkeltraject kan worden gepland.

De pilotindeling, zoals hieronder is vermeld, is tot stand gekomen aan de hand van de systeemeisen en het systeemconcept. Hierbij kwam vooral het use case diagram uit het systeemconcept van pas. In het use case diagram zijn de verschillende functionaliteiten beschreven, waardoor ik de functionaliteiten met enige overlap of dezelfde strekking heb kunnen clusteren tot pilot. In onderstaande tabel worden de pilots, met de bijbehorende systeemeisen, beschreven.

Pilot	Systeemeisen	Prioriteit
1. Importeren	HandataLink moet stamgegevens vanuit CSV en vaste lengte bestanden kunnen importeren.	Basis
2. Synchroniseren	HandataLink moet deze stamgegevens kunnen converteren en exporteren naar ODB bestanden voor gebruik in de WorkAbout.	Basis
	HandataLink moet ODB bestanden met orderregels, die zijn opgesteld door de WorkAbout, kunnen converteren en importeren.	Basis
	Na het importeren van orderbestanden moet HandataLink een back-up bestand van het orderbestand aanmaken.	Comfort
3. Exporteren	HandataLink moet de Orders kunnen exporteren naar een vaste lengte bestand.	Basis
4. Beheren handheldinstellingen	Binnen HandataLink moet het mogelijk zijn om gegevens van de WorkAbout, die essentieel zijn voor de communicatie tussen HandataLink en WorkAbout, toe te voegen, te wijzigen of te verwijderen.	Basis
5. GUI	De functionaliteiten moeten kunnen worden aangeroepen met behulp van een GUI.	Basis
6. Overig	De bestandslocaties van de import en export bestanden kunnen worden gewijzigd.	Comfort
	Back-up bestanden kunnen worden geïmporteerd in HandataLink.	Comfort
	HandataLink houdt een logboek bij m.b.t de handelingen die de gebruiker heeft uitgevoerd.	Luxe
	Modem instellingen kunnen worden beheerd.	Luxe
	Het is mogelijk om orders van nieuwe klanten te beheren.	Luxe

Figuur: Pilotindeling

De pilots staan in de tabel al in de gekozen ontwikkelvolgorde. Ik heb voor deze pilotontwikkelvolgorde gekozen aan de hand van de prioriteiten van de systeemeisen. Tevens is de ontwikkelvolgorde logisch opgesteld gezien de globale werking van HandataLink. HandataLink heeft gegevens nodig om te kunnen werken (importeren). Vervolgens heeft een WorkAbout gegevens nodig (synchroniseren). De orders worden weer ingelezen in HandataLink (synchroniseren). Tenslotte worden de gegevens klaargezet voor gebruik in de back-end (exporteren).

De planning van de verschillende pilotdelen is opgenomen in de planning van het algemene plan van aanpak. Ik heb op globaal niveau time boxing toegepast. Wanneer de ingeplande tijd voor een pilot was verstreken ben ik verder gegaan met de volgende pilot.

De planning van de verschillende pilots heb ik opgesteld met hulp van de opdrachtgever. De importfunctionaliteit is belangrijk binnen HandataLink. Deze pilot heb ik redelijk ruim ingepland om er zeker van te zijn dat de eis uit deze pilot werd behaald. De overige pilots heb ik gepland aan de hand van de functionaliteit die moest worden geboden en de geschatte tijd die nodig zou zijn om deze functionaliteit te bewerkstelligen. Vooraf was al duidelijk dat ik voor de pilot Overig te weinig tijd heb ingepland, maar de opdrachtgever vond dit niet erg aangezien deze pilot uit comfort- en luxe eisen bestaat.

4 Uitgevoerde activiteiten pilotontwikkeling

De tweede fase uit het IAD ontwikkeltraject is de ontwikkelfase. In de ontwikkelfase wordt de IAD fase pilotontwikkeling uitgevoerd. De pilotontwikkelfase heeft als doel pilots, zoals beschreven in de definitiefase, te specificeren en te realiseren.

Figuur: IAD fase Pilotontwikkeling

In de volgende paragrafen zullen de uitgevoerde activiteiten van de uitgevoerde pilots worden beschreven. De pilots uit het pilotplan van de definitiestudie die zijn uitgevoerd, zijn:

- Pilot Importeren;
- Pilot Synchroniseren;
- Pilot Exporteren;
- Pilot Beheren handheldinstellingen;
- Pilot Overig.

De pilot GUI is niet uitgevoerd. Reden hiervoor is dat de functionaliteit van de pilots Importeren, Synchroniseren en Exporteren in de achtergrond van HandataLink wordt uitgevoerd. Tevens heeft de pilot Beheren handheldinstellingen zijn eigen GUI. Hierdoor is, in overleg met de opdrachtgever, besloten om deze pilot te laten vervallen. In plaats daarvan heb ik onderzoek gedaan naar Borlands versiebeheer programma StarTeam. Dit onderzoek wordt beschrijven in § 4.5.

4.1 Activiteiten pilot Importeren

Om met een handheld te kunnen werken zijn er verschillende stamgegevens nodig. Dit kunnen bijvoorbeeld klant- en artikelgegevens zijn. Deze gegevens worden aangeboden uit een back-end systeem van een klant. Dit kan op verschillende manieren gebeuren, onder andere door middel van tekstbestanden of rechtstreeks uit een database. De geïmporteerde gegevens worden vervolgens opgeslagen. Handata heeft geen invloed op het formaat waarin bestanden worden aangeboden aan HandataLink.

Tijdens de pilotontwerp workshop is met de opdrachtgever besloten dat de importfunctionaliteit zich beperkt tot het importeren van Comma Separated Values (CSV) en vaste lengte tekstbestanden. Hiertoe is besloten omdat dit formaat tekstbestanden veel wordt gebruikt om stamgegevens, vanuit de back-end van de klant, aan te bieden aan een HandataLink applicatie. Het samenstellen van de importbestanden valt buiten het bereik van een HandataLink applicatie.

Hieronder volgt een korte beschrijving van een CSV- en een vaste lengte bestand. Deze beschrijving is afgeleid van twee bestaande importbestanden.

CSV bestand:

```
EN,"1          ","freightcosts                    "
EN,"202010  ","mirror lead 220x45cm              "
EN,"222245K","side-table copper 80x30x90cm  "
```

```
EN,"570015   ","glass flowerpot   h.13cm í14cm “
```

Figuur: Voorbeeld inhoud CSV bestand

Vaste lengte bestand:

```
00055      ;Kandelaar                ;      0,60;KANDELAAR            ;OUT
00146      ;Kaarshouder 3cm          ;     50,36;KAARSHOUDER  3CM;OUT
```

Figuur: Voorbeeld inhoud vaste lengte bestand

In het vaste lengte bestand is het overigens niet standaard dat verschillende velden gescheiden worden door een puntkomma. Het komt ook voor dat twee verschillende velden elkaar direct opvolgen zonder scheidingsteken. De bestanden die worden gebruikt om de importfunctionaliteit te testen zijn afkomstig van een bestaande klant.

In de volgende paragrafen worden de verschillende activiteiten, die ik heb uitgevoerd binnen deze pilot, nader belicht.

4.1.1 Ontwerpen van de pilot

Tijdens de pilotontwerp workshop is begonnen met het aanscherpen van de basis systeemeisen uit de fase definitiestudie. De opdrachtgever had enkele concrete eisen met betrekking tot het importeren van gegevens. Deze eisen hadden onder meer betrekking op het volgende:

- er moet kunnen worden opgegeven of de doeltabel wordt geleegd voordat er wordt geïmporteerd.
- er moet kunnen worden opgegeven hoe het bestand moet worden ingelezen, namelijk van begin tot eind of andersom.
- voor het importeren van gegevens uit een CSV bestand moet opgegeven kunnen worden of er “kolommen“ en of regels moeten worden overgeslagen.
- er moet kunnen worden opgegeven of er, in het geval van het optreden van een fout:
    - een melding moet worden getoond;
    - het logboek moet worden bijgewerkt.

Use Case

De systeemeisen zijn beschreven in een use case. De uiteindelijke use case voor het importeren van gegevens is als volgt:

Naam	Stamgegevens importeren
Samenvatting	Er worden stamgegevens geïmporteerd d.m.v. een CSV- of een vaste lengte bestand vanuit de back-end van de klant.
Actoren	HandataLink gebruiker.
Aannamen	Systeem is gestart.
Beschrijving	(1) Gebruiker maakt aan HandataLink bekend dat er stamgegevens moeten worden geïmporteerd uit een CSV of vaste lengte bestand vanuit de back-end van de klant. (2) HandataLink controleert of er importbestanden aanwezig zijn in de directory waar de te importeren bestanden aanwezig moeten zijn. Per importbestand: (3) HandataLink bepaalt of de doeltabel geleegd moet worden voordat er wordt geïmporteerd. (4) HandataLink bepaalt of het bestand van begin tot eind of andersom moet worden ingelezen. (5) In het geval van een CSV bestand bepaalt HandataLink het aantal regels en

	kolommen van het bestand die niet moeten worden geïmporteerd. (6) HandataLink importeert de stamgegevens.
Uitzonderingen	[Geen importgegevens] Als na controle blijkt dat er geen importgegevens beschikbaar zijn, zal HandataLink hiervan een melding geven aan de gebruiker.
Resultaat	De importgegevens uit het CSV of vaste lengte bestand zijn geïmporteerd.

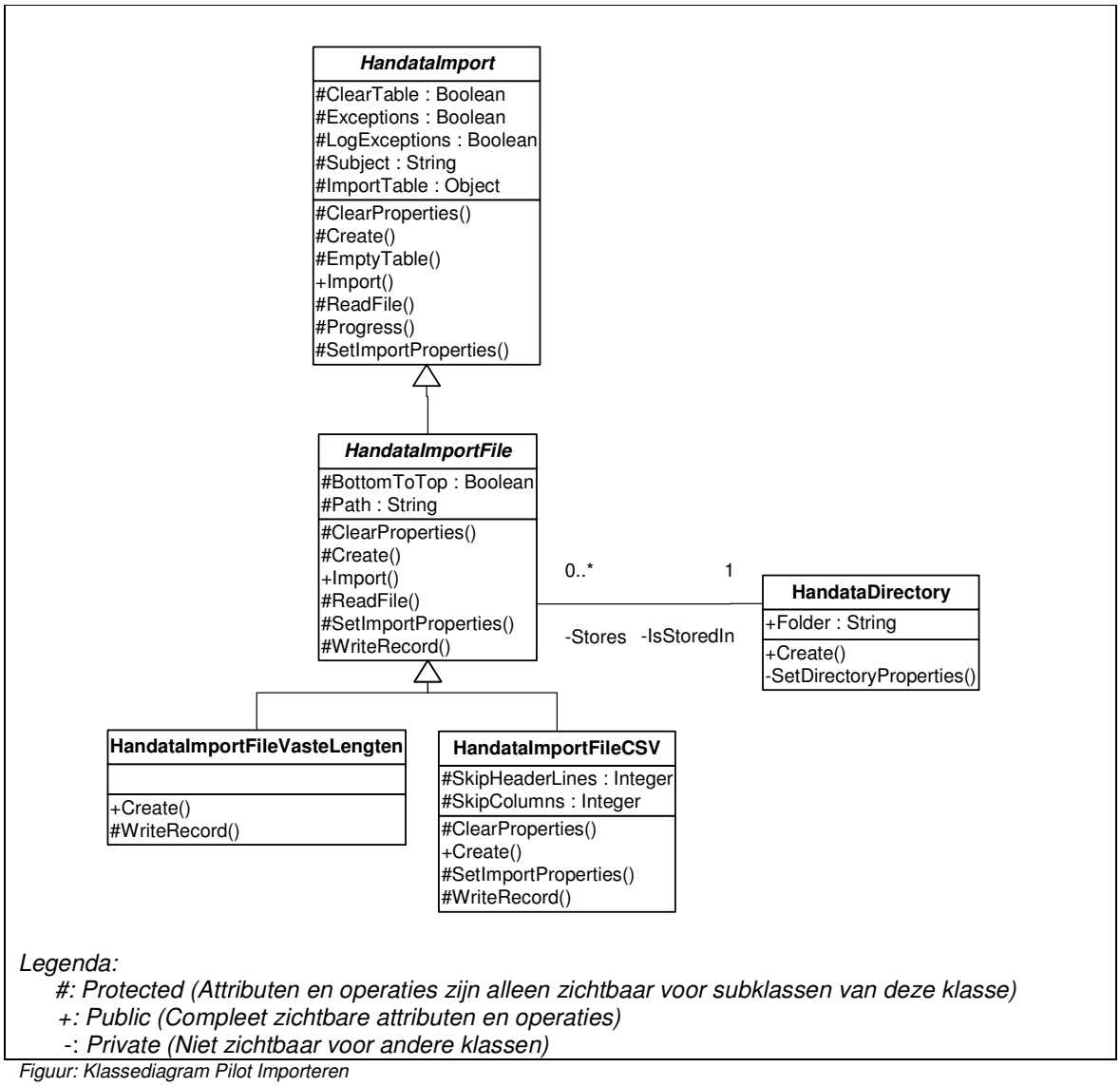
Figuur: Use case beschrijving Stamgegevens importeren

Deze use case is opgesteld met behulp van de globale use case uit de definitiestudie. Deze globale use case is vervolgens uitgebreid met de punten die naar voren zijn gekomen in de pilotontwerp workshop. De use case heb ik opnieuw voorgelegd aan de opdrachtgever die de use case vervolgens goedkeurde. Deze werkwijze heb ik ook toegepast tijdens het opstellen van de use cases in de volgende pilots.

Klassediagram

Vervolgens ben ik begonnen met het opstellen van het klassediagram. Een klassediagram beschrijft de statische structuur van een systeem. Het uiteindelijke klassediagram is na verschillende iteratieslagen opgesteld. De eerste versie van het klassediagram bestond uit één klasse genaamd HandataImport. Deze beschrijft de minimale importfunctionaliteit zonder controles (punt 2 t/m 5 uit de hierboven beschreven use case). De attributen van deze klasse zijn opgesteld aan de hand van de systeemeisen. De klasse had één methode: importeren. Deze klasse is geïmplementeerd in Delphi.

Het uiteindelijke klassendiagram voor het importeren van gegevens ziet er als volgt uit. Het verschil met het systeemconcept is duidelijk te zien. In het systeemconcept had ik voor elke importbestand een aparte klasse. Nu kunnen de verschillende importbestanden van CSV en vaste lengte formaat worden geïmporteerd met behulp van twee klassen. Ik heb met de opdrachtgever afgesproken dat de naamgeving Engelstalig is.



Legenda:
#: Protected (Attributen en operaties zijn alleen zichtbaar voor subklassen van deze klasse)
+: Public (Compleet zichtbare attributen en operaties)
-: Private (Niet zichtbaar voor andere klassen)

Figuur: Klassediagram Pilot Importeren

Dit klassediagram heb ik als volgt opgesteld: Samen met de opdrachtgever heb ik bekeken welke attributen altijd aanwezig moeten zijn wanneer er wordt geïmporteerd. Deze attributen zijn vastgelegd in de superklasse HandataImport. Als er in de toekomst een nieuwe klasse moet worden ontworpen (bijvoorbeeld één die gegevens importeert uit een database), dan zal deze nieuwe klasse als subklasse van HandataImport moeten worden ontworpen.

HandataImportFile is de naam van de klasse die het importeren van een tekstbestand afhandelt. Voor een tekstbestand geldt dat er moet worden bepaald of het bestand van begin tot eind of andersom moet worden ingelezen. Tevens dient de locatie van het bestand bekend te zijn. Om die reden heeft de klasse de attributen BottomToTop en Path. Van een tekstbestand moet bekend zijn in welke directory hij is opgeslagen. De importbestanden staan in dezelfde directory. Daarom is de klasse HandataDirectory toegevoegd aan het klassediagram.

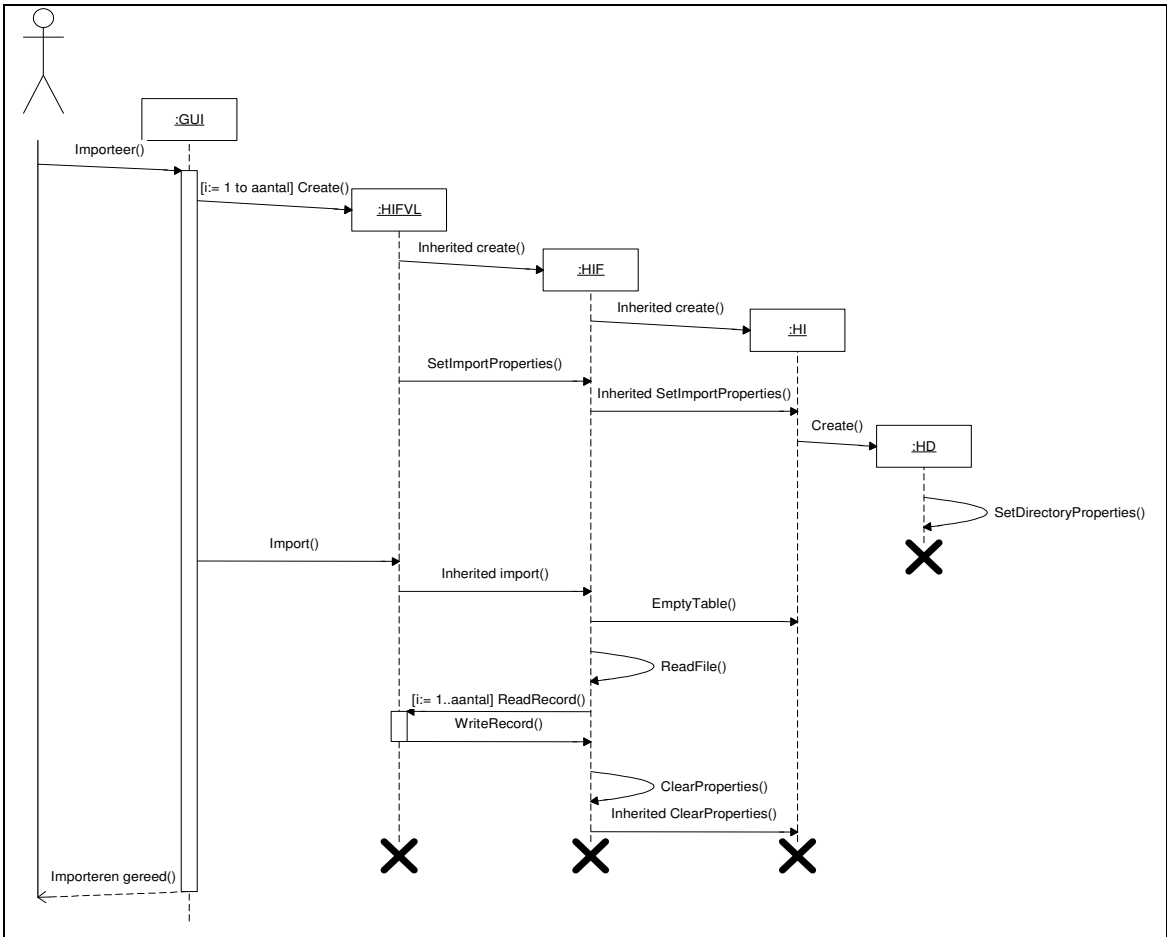
In dit geval zijn er twee soorten tekstbestanden waaruit gegevens worden geïmporteerd. Hiervoor zijn twee subklassen van de klasse HandataImportFile ontworpen. De klasse voor het importeren van een vaste lengte tekstbestand heeft zijn eigen implementatie van de operatie WriteRecord. Deze operatie zorgt ervoor dat er een record wordt weggeschreven

naar de MS Access database. Wanneer een CSV bestand wordt geïmporteerd, moet bepaald worden of er velden of regels uit het bestand kunnen worden weggelaten. Hierdoor heeft de klasse HandataImportFileCSV de attributen SkipColumns en HeaderLines gekregen.

De operaties zijn grotendeels afkomstig uit het opgestelde sequencediagram. De operatie Progress is na het opstellen van het sequencediagram toegevoegd om in de test applicatie de voortgang te tonen.

Sequencediagram

Om een goed beeld te krijgen over de verschillende interacties tussen de objecten biedt UML de keuze uit twee diagrammen, namelijk de sequence en collaboratie diagrammen. Beide nemen de use case als uitgangspunt en leggen informatie vast over het interne gedrag van het systeem. Ik heb besloten om het sequencediagram te gebruiken en niet het collaboratiediagram. Dit heeft als reden dat ik persoonlijk het sequencediagram een overzichtelijker diagram vind en ik meer ervaring heb in het opstellen van een sequencediagram. Het volgende sequencediagram is opgesteld. De gevonden interacties zijn opgenomen in het klassediagram als operaties.



Legenda:
HIFVL : HandataImportFileVasteLengte
HIF : HandataImportFile
HI : HandataImport
HD : HandataDirectory

Figuur: Sequencediagram Importeren vaste lengte bestand

Ik heb besloten om alleen een sequencediagram voor het importeren van een vaste lengte bestand op te stellen. Het sequencediagram voor het importeren van een CSV bestand heeft namelijk veel overeenkomsten met het zojuist beschreven sequencediagram en zou geen toegevoegde waarde hebben. Het sequencediagram is opgesteld met behulp van de use case. De verschillende acties uit de use case beschrijving zijn vertaald naar boodschappen in het sequencediagram. Ik heb een scenario opgesteld om het diagram te verduidelijken. Het scenario is als volgt:

De gebruiker van HandataLink heeft bepaald dat de aanwezige vaste lengte invoerbestanden moeten worden geïmporteerd. Voor elk aanwezig invoerbestand moet het volgende gebeuren.

Een instantie van HandataImportFileVasteLengten dient te worden gecreëerd. Dit gebeurt aan de hand van de constructor *Create*. Aangezien HandataImportFileVasteLengten een subklasse is van HandataImportFile, erft hij de constructor create van HandataImportFile (*Inherited create*). Om dezelfde reden erft HandataImportFile de constructor create van HandataImport (*Inherited create*) De juiste properties om het bestand in te kunnen lezen worden ingesteld (*SetImportProperties*). Dit dient voor elke subklasse van HandataImport te gebeuren. Om te kunnen bepalen in welke folder het te importeren bestand opgeslagen is, wordt er een instantie van HandataDirectory gecreëerd. De juiste directory naam wordt ingesteld (*SetDirectoryProperties*).

Om te importeren wordt de operatie *Import* van HandataImportASCII aangeroepen. Deze operatie override de abstracte operatie Import van HandataImport. Voordat er iets wordt ingelezen en opgeslagen in de database, wordt eerst gecontroleerd of de tabel, waarin moet worden geschreven, moet worden geleegd. Dit gebeurt aan de hand van de operatie *EmptyTable*.

Het bestand kan worden ingelezen. Dit gebeurt aan de hand van de operatie *ReadFile*. Binnen deze operatie wordt gecontroleerd of het bestand van begin tot eind of andersom moet worden ingelezen. In dit geval wordt het bestand van begin tot eind ingelezen (*[i := 1..aantal] ReadRecord*). Vervolgens wordt elk ingelezen record weggeschreven naar de database (*WriteRecord*)

Vervolgens wordt de operatie *ClearProperties* aangeroepen. Deze operatie zet de properties op null waarden.

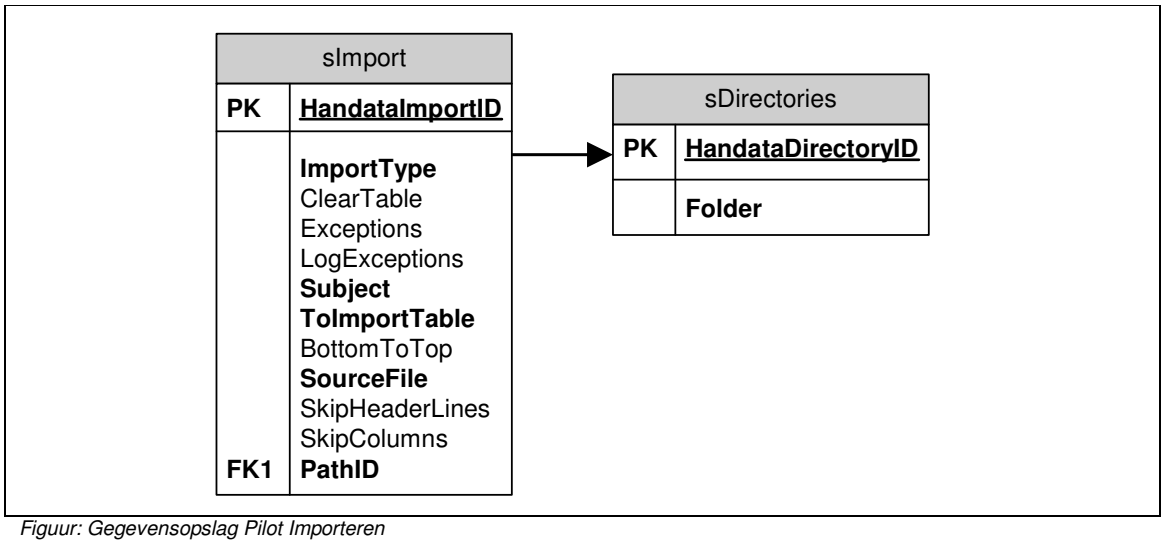
Wanneer alle bestanden zijn geïmporteerd, wordt aan de HandataLink gebruiker gemeld dat het importeren gereed is.

Figuur: Scenario Importeren vaste lengte bestand

Gegevensopslag

De opdrachtgever had als eis dat een programmeur de waarden van de attributen van de verschillende klassen kan wijzigen. Dit moet kunnen zonder code te wijzigen. Op deze manier kan er aangegeven worden dat er nieuwe bestanden moeten worden ingelezen. Het systeem moet als het ware flexibel zijn. De gedachte hierachter is dat de code op deze manier voor verschillende klanten kan worden gebruikt, zonder dat de code gewijzigd en opnieuw hoeft te worden gebuild.

Om deze reden worden de waarden van de attributen opgeslagen in een MS Access database. De keuze voor MS Access is bepaald door de opdrachtgever. De database is Handata genoemd. Ik heb de volgende tabellen gecreëerd in MS Access. De kleine letter s als toevoeging is een afkorting voor systeem. Deze naamgeving wordt binnen Handata gehanteerd om aan te geven dat de tabel is ontworpen binnen Handata.



Figuur: Gegevensopslag Pilot Importeren

Een overervingsrelatie tussen klassen in het objectmodel is niet direct weer te geven in een relationeel schema. Praktisch UML [4] beschrijft drie technieken om overerving op te slaan. Deze technieken, inclusief de gevolgen ervan, zijn:

- iedere klasse in de overervingshiërarchie een aparte tabel;
  - geen redundante attribuutgegevens;
  - volgt één op één de originele overervingstructuur;
  - zoekproces nodig om objecten te reconstrueren;
- voor iedere concrete klasse in de hiërarchie een tabel;
  - redundantie in attribuutdefinities;
  - geen zoekproces nodig om objecten te reconstrueren;
  - voor het doorzoeken van alle objecten van het generieke type dienen meerdere tabellen te worden doorlopen;
  - de originele overervingstructuur is niet meer terug te vinden;
- alle klassen in de hiërarchie in één enkele tabel;
  - geen redundante attribuutdefinities;
  - geen zoekproces nodig voor het reconstrueren van het object;
  - de ruimte voor niet ingevulde attributen wordt verspild;
  - de originele overervingstructuur is niet meer terug te vinden;

Ik heb besloten om in dit geval de laatste techniek te gebruiken. Een programmeur moet de tabel handmatig vullen. Het werk van de programmeur wordt naar mijn inzien gemakkelijker als de benodigde gegevens in één tabel kunnen worden ingevoerd. De opdrachtgever deelde deze mening.

De tabellen zijn opgesteld aan de hand van het klassediagram. Het veld ImportType is toegevoegd om te bepalen of een bestand een vaste lengte (ImportType = 0) of een CSV (ImportType = 1) is.

Ik heb nog een tweede database aangemaakt in MS Access. In deze database, HandataLink3000 genaamd, worden de stamgegevens uit de importbestanden opgeslagen. Over het ontwerp van de tabellen in deze database heb ik geen inspraak gehad. Het ontwerp van de tabellen wordt door de inhoud van het importbestand bepaald. Ik zal dit uitleggen aan de hand van een voorbeeld. Stel dat er een importbestand bestaat met gegevens over artikelen. Een artikel uit het bestand bestaat uit een artikelnummer, artikelnaam en verpakkingseenheid. Om deze gegevens op te kunnen slaan, moet er in de

[4] Jos Warmer & Anneke Kleppe, Praktisch UML 2de editie

HandataLink3000 database een tabel aangemaakt worden met 3 velden zodat het artikelnummer, de artikelnaam en de verpakkingseenheid opgeslagen kunnen worden. De volgende tabellen zijn opgesteld, dit is gedaan aan de hand van de beschikbare importbestanden:

- _Afleveradres
- _Artikel
- _Betalingsconditie
- _Debiteur
- _EANCode
- _Inventarisatie
- _Land
- _Leveringsconditie
- _Voorraad

De underscore als toevoeging dient om aan te geven dat de tabel gegevens bevat afkomstig van het back end systeem van een klant.

Zoals hierboven beschreven is, is er gekozen voor twee verschillende databases. Eén voor gegevensopslag met betrekking tot de werking van HandataLink en één voor de gegevensopslag voor de dataoverdracht tussen HandataLink en handheld. Dit heeft de volgende reden: Uit ervaring binnen Handata blijkt dat het regelmatig voorkomt dat klanten kleine aanpassingen aan HandataLink willen hebben. Deze aanpassingen kunnen betrekking hebben op een nieuw invoerbestand dat ingelezen moet worden. Klanten kunnen de database niet zelf aanpassen omdat deze is beveiligd met een wachtwoord. Hierdoor sturen klanten hun database via de e-mail op naar Handata zodat de juiste aanpassingen kunnen worden gemaakt aan de database. Op dit moment stuurt een klant al zijn eigen data mee. Dit is niet gewenst aangezien er aan deze data zelf geen aanpassingen worden gedaan. Daarom heb ik gekozen voor een scheiding tussen de databases.

Handmatige procedures

De handmatige procedures die ik heb opgesteld zijn van toepassing op een HandataLink programmeur. Voor een juiste werking moet de tabel sImport worden gevuld. Voor het importeren van een tekstbestand moeten de correcte waarden van de volgende databasevelden worden gegeven:

Veldnaam	Omschrijving
Subject	Onderwerp van te importeren gegevens bijvoorbeeld Artikelen.
SourceFile	Bronbestand met de gegevens (Naam + extensie).
ToImportTable	Doeltabel voor de gegevens.
ImportType	Type importbestand: 0 voor vaste lengten; 1 voor CSV.
ClearTable	Moet de doeltabel eerst worden geleegd.
Exceptions	Moeten de foutmeldingen op het scherm worden weergegeven.
LogExceptions	Moeten de foutmeldingen worden weggeschreven naar het logbestand.
BottomToTop	Wordt een bestand van begin tot eind of van eind tot begin ingelezen.
SkipColumns	Hoeveel kolommen moeten er worden overgeslagen vanaf de eerste. Alleen in te vullen in het geval van een CSV bestand.
HeaderLines	Hoeveel regels moeten er worden overgeslagen vanaf de eerste. Alleen in te vullen in het geval van een CSV bestand.
PathID	Correcte DatabaseID van de folder met importbestanden.

Figuur: Handmatige procedures Pilot Importeren

De handmatige procedures die ik heb opgesteld voor de overige pilots hebben ook betrekking op het correct vullen van de database. Die zal ik niet meer beschrijven. Hiervoor wordt verwezen naar de verschillende pilotontwikkelrapporten.

4.1.2 Realiseren van het pilotontwerp

Tijdens de bouw wordt het ontwerp geprogrammeerd en tussentijds getest op correcte functionaliteit. De werkwijze van programmeren die ik heb aangehouden is voor elke pilot gelijk en zal ik eenmaal beschrijven. De werkwijze is als volgt: Ik ben begonnen met het programmeren van de component definities door de verschillende klassen met hun attributen en operaties te coderen. Een unit in Delphi bestaat uit twee delen: een interface gedeelte en een implementatie gedeelte. In Delphi worden de component definities gecodeerd in het interface gedeelte van een unit.

Hier volgt de component definitie van de klasse HandataImport uit het eerder beschreven klassediagram. De oplettende lezer merkt op dat de klasse HandataImport wordt geïmplementeerd als THandataImport. De toevoeging 'T' is binnen Delphi een standaardnotatie om aan te geven dat het om een object gaat.

```
Type
  THandataImport = class(TObject)
  private
    {private declarations}
    FClearTable    : boolean;
    FExceptions    : boolean;
    FLogExceptions : boolean;
    FSubject       : string;
    FImportTable   : TTable;
  protected
    {protected declarations}

    {properties}
    property ClearTable      : boolean read FClearTable      write FClearTable;
    property Exceptions      : boolean read FExceptions      write FExceptions;
    property LogExceptions : boolean read FLogExceptions write FLogExceptions;
    property Subject          : string     read FSubject           write FSubject;
    property ImportTable     : TTable   read FImportTable     write FImportTable;

    {constructor}
    constructor Create;

    {procedures}
    procedure ClearProperties; virtual;
    procedure EmptyTable;
    procedure Progress;
    procedure ReadFile; virtual;
    procedure SetImportProperties; virtual;
  public
    {public declarations}
    procedure Import; virtual;
  end;
```

Figuur: Component definitie THandataImport

Met behulp van de class completion functionaliteit van Delphi worden in het implementatie gedeelte van de unit de procedures gedeclareerd. Vervolgens heb ik elke procedure voorzien van commentaar waarin beschreven wordt wat het gedrag van de procedure moet zijn. Het commentaar is afkomstig uit de scenario's die zijn opgesteld bij de sequence

diagrammen. Tenslotte heb ik de benodigde procedures geprogrammeerd, tussentijds getest en zonodig debugged. De geprogrammeerde procedures zijn, voor zover ze nog niet aanwezig waren, in het klasse- en sequencediagram opgenomen. Op deze manier zijn de klasse- en sequencediagrammen consistent gebleven met de geprogrammeerde component definitie.

Een CSV bestand bestaat uit ASCII-regels, gescheiden door #13#10 (CR/LF, harde return). Elke regel is een record, behalve vaak de eerste regel waarin de veldnamen staan. Elk record bestaat weer uit velden, in het algemeen gescheiden door een komma. Rondom elk veld mogen dubbele apostrofs staan. De apostrofs dienen ervoor om een komma die in de tekst voorkomt als een komma te kunnen herkennen. Anders zou de komma als veldscheider worden herkend. Als er een apostrof binnen een veld voorkomt, moet dat als dubbele apostrof worden opgeslagen, anders zou deze apostrof het einde van het veld markeren.

Binnen Delphi kan een CSV bestand gemakkelijk uitgelezen worden met behulp van een TStringList. Een TStringList heeft de property CommaText die een CSV regel inleest en omzet naar een lijst met strings. Onderstaande code beschrijft hoe een CSV bestand wordt uitgelezen:

```
procedure THandataImportFileCSV.ReadRecord(i: integer; Regel: TStringList);
{ Deze procedure leest per regel de CSV velden in,
  hierbij moet worden gelet op:
      -     skipheaderlines
      -     skipcolumns }
var
  CSVVelden: TStringList;
  j: integer;
begin
  {Geen kopregels inlezen}
  if i > ( SkipHeaderLines - 1 ) then
    begin
    try
      {initialisatie}
      CSVVelden := TStringList.Create;
      {per regel inlezen}
      CSVVelden.CommaText := Regel.Strings[i];
      …
      {worden er minder kolommen overgeslagen dan er velden zijn?}
      if ( ImportTable.FieldCount - 1 ) > skipColumns then
      begin
        {Voor elk veld uit de tabel de waarde bepalen en opslaan}
        for j := skipColumns to ( ImportTable.FieldCount - 1 ) do
        begin
           ImportTable.Fields[j].AsString := trim(CSVVelden[j]);
        end; // for
      end; // if
  …
end;
```

Figuur: Code inlezen CSV bestand

Om de functionaliteit te kunnen testen heb ik een kleine testapplicatie gemaakt. Deze applicatie bestaat uit een form en een button. Wanneer op de button wordt geklikt, wordt afhankelijk van het type importbestand een instantie van THandataImportFileVasteLengten of THandataImportFileCSV gecreëerd. Vervolgens wordt de procedure import aangeroepen en wordt het importeren gestart. Dit gebeurt voor elk bestand (CSV en vaste lengte) dat beschreven is in de tabel sImport in de database. De code die hiervoor gebruikt wordt, kan later in de GUI voor HandataLink gebruikt worden om de importfunctionaliteit van vaste

lengte en CSV bestanden aan te roepen. Hier volgt een deel van de code achter deze button:

```
VL := THandataImportFileVasteLengten.Create;
…
While not tblImport.Eof do
  begin
  {Per record uit de tabel sImport}
    ImportType := tblImport.FieldByName('ImportType').AsInteger;
    {Vaste lengte of CSV inlezen}

    Case ImportType of
      0:
      {Vaste lengte inlezen}
      begin
        {Importeren maar}
        VL.Import;
      end;
      …
    end; //case
    {Door naar het volgende record}
    tblImport.Next;
  end;//while
…
```

Figuur: Code aanroep importfunctie

De geschreven testapplicatie heb ik in de pilots Synchroniseren en Exporteren uitgebreid, zodat ik de functionaliteit van die pilots kon testen. Tijdens de bouw hebben er zich geen problemen voorgedaan.

4.1.3 Testen van de realisatie

Om te kunnen testen heb ik de tabel sImport van de Handata database zo gevuld dat de verschillende importbestanden ingelezen kunnen worden ingelezen. De verschillende tests heb ik uitgevoerd nadat een deelfunctionaliteit gereed was. Om de functionaliteit te kunnen testen heb ik gebruik gemaakt van verschillende testbestanden. Deze testbestanden zijn bestanden die afkomstig zijn van verschillende klanten die HandataLink gebruiken. Voordat ik de code ging testen heb ik verschillende tests beschreven. Dit heb ik gedaan aan de hand van de beschrijving uit de use case.
Ik heb onder andere voor de volgende situaties tests beschreven:

- alleen vaste lengte bestanden inlezen;
- alleen CSV bestanden inlezen;
- vaste lengte en CSV bestanden inlezen;
- verschillende waarden voor de attributen;

De tests zijn onder andere opgesteld aan de hand van de opgestelde use case. Zo heb ik ook getest of de doeltabel wordt geleegd en of het bestand van begin tot eind wordt ingelezen indien dit is aangegeven. Deze punten komen uit de use case.

De tests zijn als volgt opgesteld. Per situatie heb ik beschreven wat de verwachte uitkomst is. Vervolgens heb ik gekeken of de verwachte uitkomst overeenkomt met de werkelijke uitkomst. In de meeste gevallen was dit het geval. Deze tests zijn dus succesvol uitgevoerd. In sommige gevallen kwam de verwachte uitkomst niet overeen met de werkelijke uitkomst. Ik heb toen de verantwoordelijke code aangepast. De test opnieuw uitgevoerd. Ik heb dit gedaan totdat de test wel succesvol werd uitgevoerd.

Hier volgt een beschrijving van een uitgevoerde test. In deze test wordt een CSV bestand geïmporteerd. Het veld HeaderLines heeft de waarde 5 en het bestand moet van begin tot eind ingelezen worden.

Verwachte uitvoer	Correcte uitvoer
De eerste vijf regels uit het importbestand worden niet opgeslagen in de database. De overige regels uit het bestand wel.	√

Figuur: Testuitvoer importeren CSV bestand

Tevens heb ik getest wat er gebeurt wanneer er 'corrupte' CSV bestanden ingelezen worden. Het inlezen en wegschrijven van de gegevens uit het importbestand wordt dan niet correct uitgevoerd. Ik heb dit probleem voorgelegd aan de opdrachtgever. Hij heeft mij toen uitgelegd dat een klant (HandataLink gebruiker) verantwoordelijk is voor het aanbieden van correcte bestanden. Maar dat het niet mag gebeuren dat HandataLink vastloopt. Dat is met mijn code niet gebeurd. Wat dat betreft voldoet de door mij geschreven code aan deze eis die door de opdrachtgever is gesteld.

4.2 Activiteiten pilot Synchroniseren

Deze pilot is in het pilotplan opgedeeld in de volgende pilotdelen:

- synchroniseren HandataLink met handheld: gegevens uit de HandataLink3000 database converteren naar een ODB bestand en klaarzetten in de export directory van HandataLink.
- synchroniseren handheld met HandataLink: gegevens uit een, door de WorkAbout aangemaakt, ODB bestand converteren en opslaan in de HandataLink3000 database.
- aanmaken back-upbestand: na synchronisatie van een handheld met HandataLink moet er een back-upbestand zijn aangemaakt waarin de zojuist geïmporteerde gegevens zijn opgeslagen.

De eerste twee pilotdelen zijn achtereenvolgens uitgevoerd. Het derde pilotdeel is wegens tijdgebrek niet uitgevoerd.

4.2.1 Ontwerpen van de pilot

Deze paragraaf beschrijft de activiteiten die zijn uitgevoerd tijdens het pilotontwerp. Per pilotdeel worden de uitgevoerde activiteiten toegelicht.

Pilotdeel: Synchroniseren HandataLink met handheld

Nadat de gegevens vanuit de back-end van een klant zijn geïmporteerd in HandataLink, moeten deze gegevens worden klaargezet voor gebruik met een handheld. Vanuit de handheld kunnen er gegevens klaarstaan die moeten worden ingelezen door HandataLink. Dit kunnen bijvoorbeeld ordergegevens zijn. Het synchroniseren van gegevens verschilt per type handheld. Een van de type Handhelds die gebruikt kunnen worden met HandataLink is de Psion WorkAbout (in het vervolg WorkAbout genoemd).

Tijdens de definitiestudiefase is besloten dat de gegevens moeten worden gesynchroniseerd met de WorkAbout. Het synchroniseren van gegevens met een WorkAbout gebeurt aan de

hand van ODB (OPL Database) bestanden. Een ODB bestand is een database file voor de WorkAbout.

Structuur ODB export bestand

Zoals zojuist is beschreven vindt de synchronisatie van gegevens plaats d.m.v. een ODB bestand. De opdrachtgever heeft mij de structuur van een standaard ODB bestand, voor gebruik in de WorkAbout, uitgelegd. Een ODB bestand bestaat uit:

- Een vaste fileheader: “OPLDatabaseFile + #0 + #31 + #17 + #22 + #0 + #15 + #17”
- Aantal velden per record, afgesloten met #32.
- Recordopbouw, elk veldtype heeft een eigen waarde in OPL (Organiser Programming Language, de programmeertaal voor Psion handhelds) :
  - Integer = 0
  - Long integer = 1
  - Floating point = 2
  - String = 3
  - Uitgevulde string = 3

  Wanneer een record nu uit een integer en twee strings bestaat, moet worden weggeschreven 033.
- Per Record
  - Lengte van het record, afgesloten met #16. Elk type veld heeft zijn eigen lengte:
    - Integer waarde wordt opgeslagen in 2 karakters
    - Long integer waarde wordt opgeslagen in 4 karakters
    - Floating Point waarde wordt opgeslagen in 8 karakters
    - String heeft wisselende lengte
    - Uitgevulde string string met vaste lengte
  - De velden (data).

Hier volgt het voorbeeld vaste lengte bestand, zoals beschreven in de vorige paragraaf, in ODB formaat:

```
OPLDatabaseFile □□ □□□ □□□□□&□□00055Kandelaar
KANDELAAR333333ã?□OUT3□□00146□Kaarshouder 3cm□KAARSHOUDER
3CM®Gáz□.I@□OUT
```

Figuur: Voorbeeld ODB exportbestand

Use case synchroniseren HandataLink met handheld

Tevens is de use case verder gedetailleerd. De use case uit de definitiestudie is uitgebreid a.d.h.v. de structuur beschrijving van het ODB bestand.

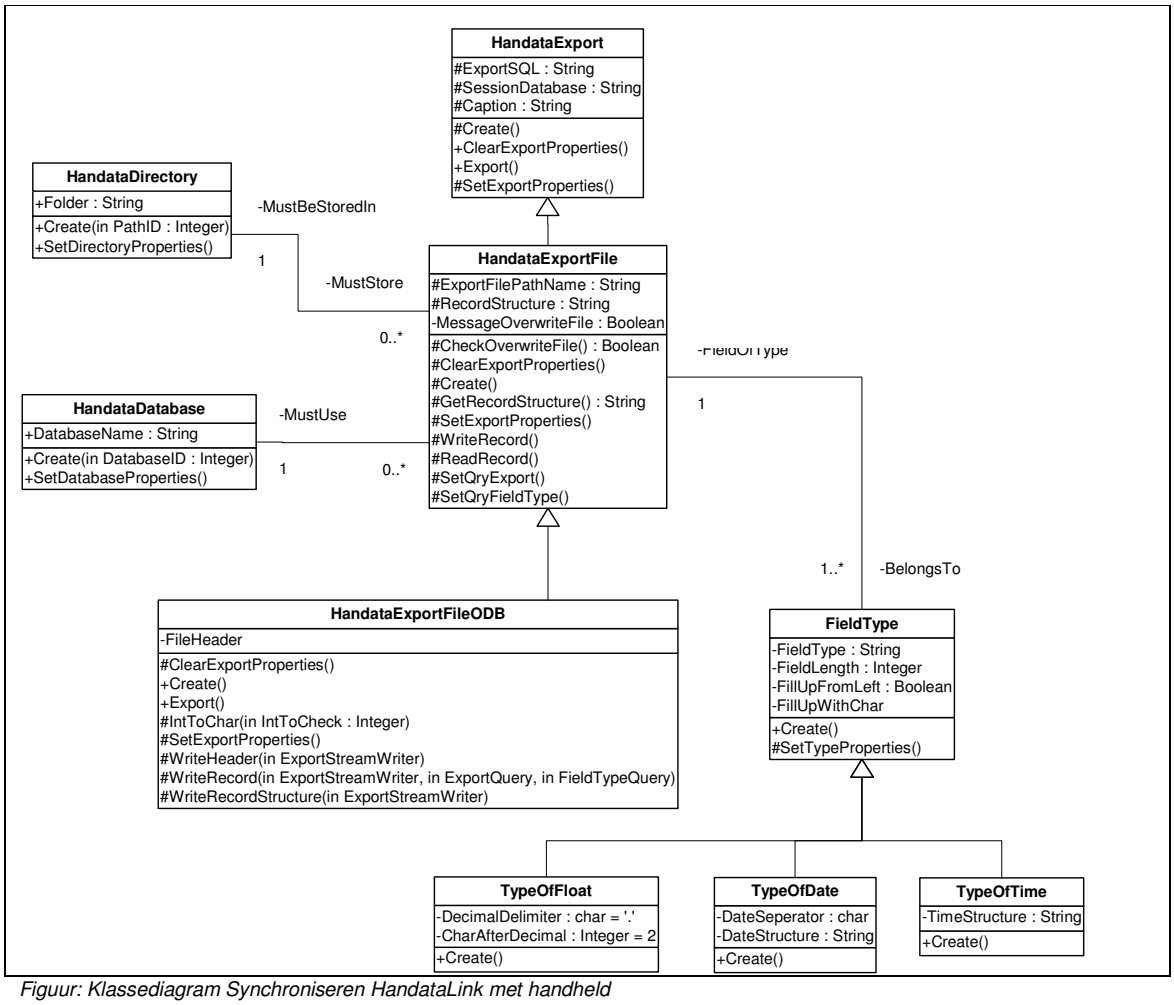
Naam	Synchroniseren HandataLink met Handheld d.m.v. ODB bestand
Samenvatting	Haalt de benodigde stamgegevens uit de juiste databasetabel en converteert deze stamgegevens naar het formaat van een ODB bestand.
Actoren	HandataLink gebruiker
Aannamen	Databasetabel bestaat en bevat gegevens voor het ODB bestand.
Beschrijving	(1) Er wordt gecontroleerd of de databasetabel aanwezig is en stamgegevens bevat voor het te genereren ODB bestand. (2) Indien het ODB bestand nog niet bestaat, wordt deze aangemaakt en geopend. Anders wordt de inhoud van het ODB bestand geopend en zullen de oude gegevens worden overschreven. (3) De standaard fileheader voor het ODB bestand wordt geschreven. (4) De recordopbouw van het bestand wordt in het ODB bestand geschreven. Op deze manier weet de handheld hoe het bestand gelezen moet worden. Het record kan opgebouwd zijn uit de volgende

	type velden: • I : Integer; • L: Long integer; • F: Floating point; • S: String; • U: Uitgevulde string (vaste lengte); (5) Per record uit de database wordt • de lengte van het record bepaald. De lengte van de velden uit het record zijn vast, nl - I : lengte is 2 - L : lengte is 4 - F : lengte is 8 - S : lengte is S.length - U : vaste lengte. • de data van het record worden geschreven naar het ODB bestand. (6) ODB bestand wordt gesloten.
Uitzonderingen	[Databasetabel bestaat niet of bevat geen gegevens]. Er wordt geen ODB bestand aangemaakt.
Resultaat	In de exportdirectory van HandataLink is een ODB bestand aanwezig dat de gegevens bevat van een tabel uit database HandataLink3000. Het ODB bestand kan worden gebruikt door een handheld van het type Psion Workabout.

Figuur: Use case beschrijving Synchroniseren HandataLink met handheld

Klassediagram

Het volgende klassediagram voor het synchroniseren van HandataLink met een handheld is opgesteld:



Figuur: Klassediagram Synchroniseren HandataLink met handheld

Ik ben als volgt tot dit klassediagram gekomen. Aangezien er gegevens uit HandataLink worden geëxporteerd, heb ik gekozen voor de naam HandataExport. HandataExport is de superklasse met de attributen die voor elke (toekomstige) subklasse bekend moeten zijn. Er moet voor alle (toekomstige) subklassen bekend zijn:

- welke gegevens moeten worden geëxporteerd (ExportSQL);
- uit welke database deze gegevens moeten worden geëxporteerd (SessionDatabase);
- een attribuut om de HandataLink gebruiker informatie te kunnen verschaffen over de actie waar HandataLink mee bezig is (Caption).

In dit geval worden er gegevens naar een bestand geëxporteerd, vandaar de naam HandataExportFile voor de subklasse. Van een bestand moet bekend zijn waar het moet worden opgeslagen (ExportFilePathName) op welke manier een te exporteren record is opgebouwd (RecordStructure) en of er melding moet worden gegeven als een bestand driegt te worden overschreven (MessageOverwriteFile).

Tenslotte gaat het hier om een ODB bestand dat moet worden geëxporteerd. En moet de fileheader bekend zijn. De fileheader is iets specifieks voor een ODB bestand.

Alle opgestelde ODB bestanden komen in dezelfde export directory van HandataLink terecht. Dat verklaart de relatie tussen HandataExportFile en HandataDirectory.

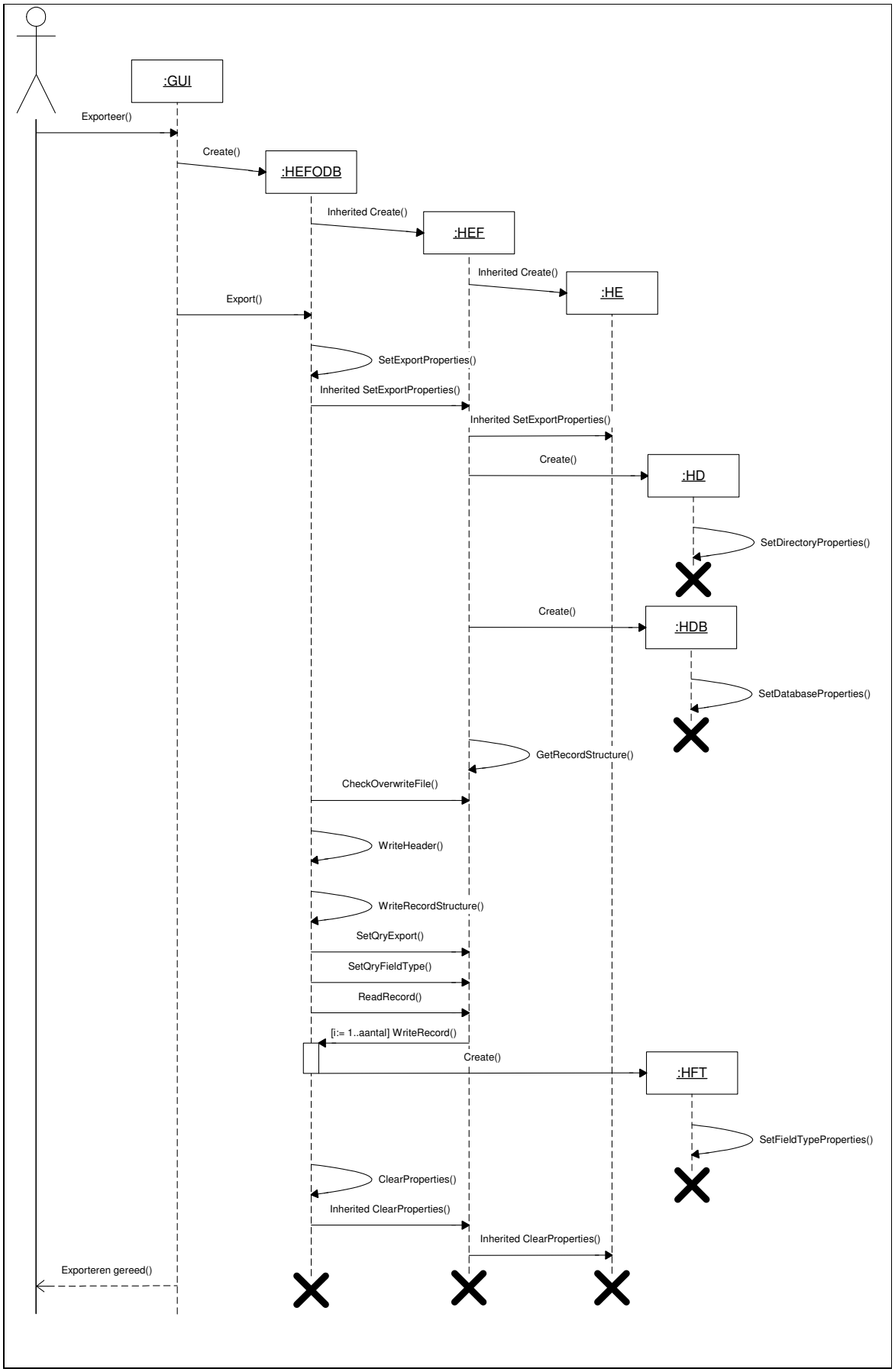
Het kan voorkomen dat de te exporteren gegevens niet afkomstig zijn uit een bestand, zoals is beschreven in de pilot importeren, maar uit een database in de back end van de HandataLink gebruiker. Er moet dus bekend zijn uit welke database de gegevens moeten worden geëxporteerd. Om deze reden is de klasse HandataDatabase toegevoegd.
Het exporteren vanuit een andere database valt buiten het bereik van de opdracht. Maar door de toevoeging van de klasse HandataDatabase wordt er alvast rekening gehouden met de toekomst.

De verschillende veldtypen worden op een verschillende manier geconverteerd tot ODB formaat. Daarom is besloten om een klasse FieldType toe te voegen. Deze klasse biedt informatie over de verschillende lengten en opmaak van de velden.

Bij dit klassediagram heb ik een modeldictionary opgesteld. Voor de beschrijving van de modeldictionary wordt verwezen naar de externe bijlage “Pilotontwikkelrapport pilot 2 Synchroniseren”.

Sequencediagram

Ik heb het volgende sequence diagram opgesteld om inzicht te krijgen in de verschillende interacties tussen de objecten. De interacties uit het sequence diagram zijn in het klassediagram opgenomen als operaties.


:GUI
Exporteer()
Create()
:HEFODB
Inherited Create()
:HEF
Inherited Create()
:HE
Export()
SetExportProperties()
Inherited SetExportProperties()
Inherited SetExportProperties()
Create()
:HD
SetDirectoryProperties()
Create()
:HDB
SetDatabaseProperties()
GetRecordStructure()
CheckOverwriteFile()
WriteHeader()
WriteRecordStructure()
SetQryExport()
SetQryFieldType()
ReadRecord()
[i:= 1..aantal] WriteRecord()
Create()
:HFT
SetFieldTypeProperties()
ClearProperties()
Inherited ClearProperties()
Inherited ClearProperties()
Exporteren gereed()

Legenda:
GUI : Grafische User Interface
HEFODB : HandataExportFileODB
HEF : HandataExportFile
HE : HandataExport
HD : HandataDirectory
HDB : HandataDatabase
HFT : HandataFieldType

Figuur: Sequencediagram Synchroniseren HandataLink met handheld

Het sequencediagram is opgesteld met behulp van het volgende scenario:

Er is bepaald dat er gegevens moeten worden geëxporteerd naar een .ODB bestand. Een instantie van THandataExportFileODB dient te worden gecreëerd. Dit gebeurt aan de hand van de constructor *Create*. Aangezien THandataExportFileODB een subklasse is van THandataExportFile, erft hij de constructor van THandataExportFile (*Inherited Create*). Om dezelfde reden erft THandataExportFile de constructor van THandataExport (*Inherited Create*)

Om de gegevens te exporteren wordt de operatie *Export* van THandataExportFileODB aangeroepen. Deze operatie override de abstracte operatie export van THandataExport. De operatie export gebruikt de tabel sExport uit de database om de juiste properties in te kunnen stellen (*Inherited SetExportProperties*). Om de juiste properties in te kunnen stellen, heeft THandataExportFile informatie nodig van THandataDirectory (*Folder*) en THandataDatabase (*DatabaseName*). Een instantie van THandataDirectory wordt gecreëerd (*Create(PathID)*). Een instantie van THandataDatabase wordt gecreëerd (*Create(DatabaseID)*). De parameters PathID en DatabaseID worden meegegeven om de correcte creatie van de twee instanties te verkrijgen.

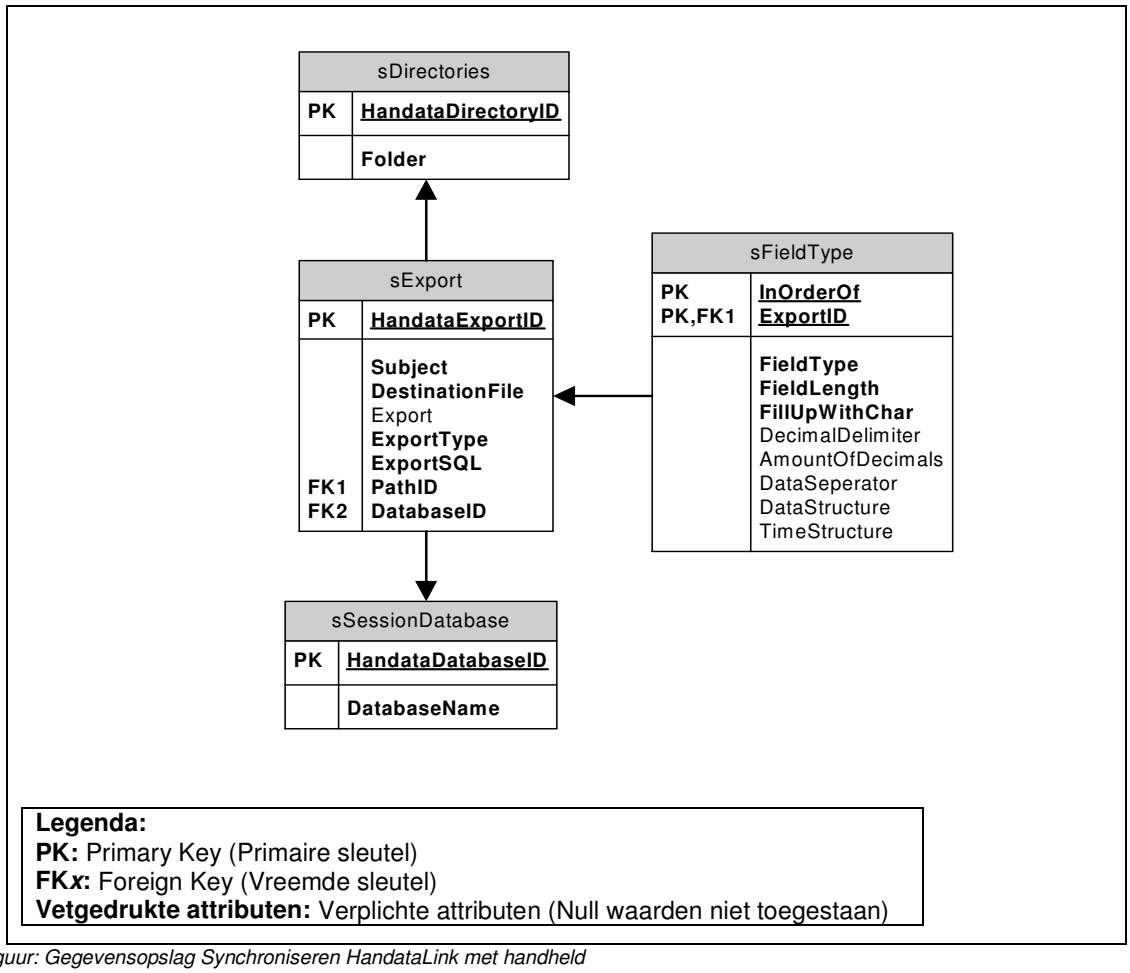
THandataExportFileODB moet weten of, indien er al een geëxporteerd .ODB bestand aanwezig is, de bestaande gegevens overschreven kunnen worden. Dit wordt gecontroleerd d.m.v. *CheckOverwriteFile*. Er wordt *True* of *False* teruggegeven. De fileheader wordt weggeschreven (*WriteFileHeader*). De queries voor het ophalen van de stamgegevens en de veldtypen worden gezet en geopend (*SetQryExport, SetQryFieldType*). De procedure ReadRecord wordt aangeroepen. Elk record dat aanwezig is wordt weggeschreven naar het ODB bestand (*[i:= 1..aantal] WriteRecord*). Om een record weg te schrijven, is er informatie nodig van het veldtype dat moet worden weggeschreven (*Create HFT*).

Wanneer het exporteren gereed is, moeten de properties op null worden gezet (*inherited ClearExportProperties*).

Figuur: Scenario Synchroniseren HandataLink met handheld

Gegevensopslag

De tabellen sExport, sSessionDatabase en sFieldType zijn toegevoegd aan de database “Handata” om het flexibele karakter van HandataLink te bewaren.



Figuur: Gegevensopslag Synchroniseren HandataLink met handheld

De tabellen zijn opgesteld aan de hand van het klassediagram. Aan de tabellen zijn primaire sleutels toegevoegd om een record te kunnen identificeren. Aan de tabel sExport zijn velden toegevoegd die de naam van het bestand (DestinationFile), het ExportType (0 = ODB bestand) en of het bestand moet worden geëxporteerd (export) weergeven.

Pilotdeel 2: Synchroniseren handheld met HandataLink

Op een gegeven moment zullen er gegevens van de WorkAbout moeten worden verstuurd naar HandataLink. Deze gegevens kunnen bijvoorbeeld betrekking hebben op, door een WorkAbout, opgestelde orders. Ook dit gebeurt d.m.v. ODB bestanden.

Structuur ODB import bestand

De inhoud van het ODB bestand dat door HandataLink wordt ingelezen is bijna hetzelfde als het ODB bestand dat door HandataLink wordt weggeschreven. Er is echter een verschil. Dat verschil betreft informatie over het record. Het ODB import bestand geeft per record het volgende aan:

- Lengte van het record
- Het recordtype
- De velden

In vergelijking met het ODB export bestand wordt nu dus aangegeven wat het type van het record is. Indien het recordtype in het bereik van 0 – 15 ligt, is het record gemerkt als

'deleted'. Dit betekent dat dit record niet ingelezen mag worden. Het is met een WorkAbout niet mogelijk om records permanent te verwijderen. Het is alleen mogelijk om aan te geven dat een record verwijderd ('deleted') is. Indien het recordtype een waarde buiten 0 – 15 heeft, moet het gewoon worden ingelezen.

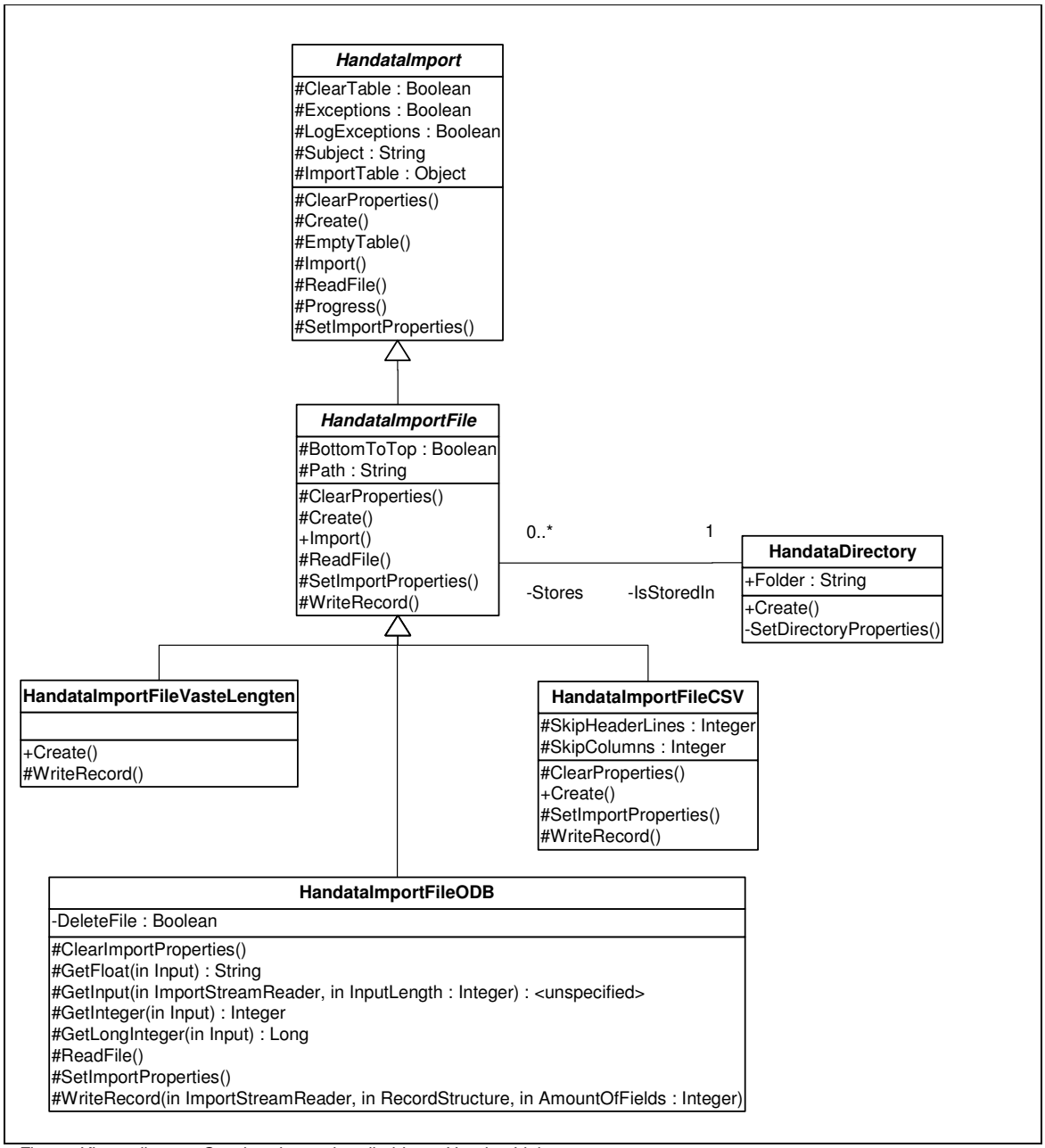
Use case synchroniseren handheld met HandataLink

Tijdens de pilotontwerp workshop is de use case uit de definitiestudie verder gedetailleerd. Dat heeft geresulteerd in de volgende use case.

Naam	Synchroniseren Handheld met HandataLink d.m.v. ODB bestand
Samenvatting	Leest een door een handheld opgesteld ODB bestand in en schrijft de gegevens naar de doeltabel in de database HandataLink3000.
Actoren	Userinterface
Aannamen	ODB bestand bestaat en de doeltabel bestaat.
Beschrijving	(1) Er wordt gecontroleerd of het in te lezen ODB bestand bestaat. (2) Vervolgens wordt gecontroleerd of de doeltabel bestaat en actief is. (3) Het bestand wordt ingelezen en indien nodig record voor record weggeschreven naar de doeltabel in de database. Dit gaat in de volgende stappen: • De OPL fileheader wordt gefilterd. • Het aantal velden in een record wordt gelezen. • De recordopbouw wordt ingelezen. • Record voor record wordt ingelezen: - Lengte van het record wordt ingelezen. - Recordtype wordt ingelezen. - Indien het record niet gemerkt is als 'deleted': ▪ Nieuw record wordt aangemaakt in de tabel. ▪ Per record worden de verschillende velden ingelezen. Deze velden zijn van type: • Integer; • Long integer; • Floating Point; • String. (4) Indien het bestand mag worden verwijderd, wordt het bestand verwijderd.
Uitzonderingen	[ODB bestand bestaat niet of de doeltabel bestaat niet.] Er wordt geen ODB bestand ingelezen.
Resultaat	De gegevens uit het ODB bestand, zijn ingelezen en opgeslagen in de corresponderende tabel in de database HandataLink3000.

Figuur: Use case beschrijving Synchroniseren handheld met HandataLink

Het volgende klassediagram is opgesteld:



Figuur: Klassediagram Synchroniseren handheld met HandataLink

De oplettende lezer merkt direct op dat dit een aangepaste versie van het klassediagram uit pilot 1 is. Aangezien er een ODB bestand moet worden geïmporteerd, is er een nieuwe subklasse van HandataImportFile ontworpen genaamd HandataImportFileODB. HandataImportFileODB heeft een boolean attribuut DeleteFile. Met DeleteFile wordt bepaald of het ODB bestand kan worden verwijderd nadat het importeren is afgerond (punt (4) uit de use case). Of een ODB bestand mag worden verwijderd is afhankelijk van de eisen van de klant (HandataLink gebruiker).

Sequencediagram synchroniseren handheld met HandataLink

Ik heb besloten om geen sequencediagram op te stellen. Het sequencediagram voor het importeren van een ODB bestand zou geen extra kennis toevoegen, aangezien dit

sequencediagram veel overeenkomsten zou hebben met het sequencediagram beschreven in § 4.1.1 betreffende de uitgevoerde activiteiten tijdens de eerste pilot.

4.2.2 Realiseren van het pilotontwerp

Per pilotdeel wordt de realisatie van het pilotontwerp beschreven.

Pilotdeel: Synchroniseren HandataLink met handheld

Ook dit maal is er tijdens de bouw begonnen met het creëren van component definities met behulp van het klassediagram. Dit geldt voor alle klassen uit het diagram met uitzondering van HandataDirectory, aangezien deze al is geïmplementeerd tijdens de vorige pilot.

Het ophalen van de te exporteren gegevens gebeurt aan de hand van verschillende queries. Deze queries worden opgeslagen in de tabel sExport in de Handata database. Het ophalen van de gegevens is zonder problemen verlopen. Het wegschrijven van de gegevens naar een ODB bestand heeft wel enkele problemen gekend. Deze problemen hadden te maken met de verschillende manier waarop WIN32 en .NET (tekst) bestanden opslaan. Hieronder volgt een beschrijving van het probleem en de gevonden oplossing.

Om het probleem te begrijpen moet eerst duidelijk zijn hoe een ODB bestand wordt opgeslagen. Een veld van het type integer, long integer of floating point moet worden omgezet naar een combinatie van ANSI karakters. Zo als al bleek, tijdens de beschrijving van de structuur van het exportbestand, hebben velden van een integer, long integer en floating point een vast formaat. Voor elk type veld bestaat een manier om de waarden om te zetten naar het aantal benodigde karakters. Een integer bijvoorbeeld, wordt op de volgende manier geconverteerd naar ANSI karakters.

```
var
  Bit1 : Integer;
  Bit2 : Integer;
  Field : String;
  Value : Integer;
begin
  …
  { Value is de originele waarde van het integer veld,
    Een integer wordt opgeslagen in twee karakters }
  Bit1 := trunc( value / 256 );
  Bit2 := Value – Bit1;
  { Integers omzetten naar bijbehorende karakters in ANSI karakterset }
  Field := Chr(Bit2) + Chr(Bit1);
  …
```

Figuur: Code integer converteren

Tijdens het programmeren heb ik tussentijds getest of de conversie goed verliep. Deze tests werden uitgevoerd met behulp van de applicatie ODBDump. Deze (WIN32) applicatie leest een ODB bestand in en converteert het naar een leesbaar .txt bestand. Tijdens het testen bleek dat sommige velden een andere waarde kregen.

Ik heb twee ODB bestanden met elkaar vergeleken. Eén opgesteld door de huidige HandataLink en de ander door de door mij geschreven code. Het viel me op dat de codering van de twee bestanden van elkaar verschilde. De codering van het originele ODB bestand is ANSI terwijl de codering van het andere ODB bestand Unicode is. Op Internet heb ik

vervolgens onderzoek gedaan naar de verschillen tussen Unicode en ANSI. Samengevat heb ik gevonden dat de karakters van ANSI en Unicode met een nummer boven 127 niet altijd met elkaar overeenkomen.

Vervolgens ben ik op zoek gegaan naar een oplossing binnen Delphi. Na enige tijd stuitte ik op de streamwriter class. De streamwriter class heeft een property encoding waarmee kan worden bepaald volgens welke encoding de byte/karakter conversie plaats moet vinden. Dit resulteerde in de volgende code voor de creatie van de streamwriter:

```
var
  ANSI : encoding;
  ExportFileStream : TFileStream;
  ExportStreamWriter : TStreamWriter;
begin
  …
  {.Net gebruikt Unicode encodering, ExportStreamWriter wordt geencodeerd }
  ANSI := Encoding.get_Default;
  ExportFileStream := FileStream.Create(ExportFilePathName, FileMode.Create) ;
  ExportStreamWriter := StreamWriter.Create(ExportFileStream, ANSI);
  …

NB : Default encoding is ISO-8859-1 karakterset
```

Figuur: Code creatie streamwriter

Met behulp van de streamwriter waren er nog een aantal velden die niet correct werden weggeschreven naar het ODB bestand. Na verder onderzoek op het Internet vond ik dat er enkele ANSI karakters niet aanwezig zijn in de ISO-8859-1 karakterset. Dit zijn de karakters met ANSI nummer:

- 128
- 130 t/m 140
- 142
- 145 t/m 156
- 158 en 159

Ik heb verder gezocht naar een oplossing binnen Delphi. Maar heb die, ondanks het gebruik van de helpfile en het Internet, niet kunnen vinden. Ik heb daarom de volgende functie IntToChar geschreven. Deze functie converteert de onbekende karakters uit de ISO-8859-1 karakterset naar de juiste ANSI karakters.

```
Function THandataExportFileODB.IntToChar(IntToCheck : Integer): Char;
{ De nummers uit IntArray zijn onbekende karakters in de ISO-8859-1 karakterset,
 deze functie converteert de onbekende karakters naar de juiste ANSI karakters }
const
  IntArray  : Array [0..26] of Integer = (128, 130 , 131, 132, 133, 134, 135, 136, 137,
    138, 139, 140, 142, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 158, 159);
  CharArray : Array [0..26] of Char    = ('€', ',', 'ƒ', '„', '…', '†', '‡', 'ˆ', '‰',
    'Š', '‹', 'Œ', 'Ž', '‘', '’', '“', '”', '•', '–', '—', '˜', '™', 'š', '›', 'œ', 'ž', 'Ÿ');
var
  CharTeller : integer;
begin
  result := chr(IntToCheck);
  For charteller := 0 to Length(IntArray) - 1 do
  begin
    {integer converteert niet correct in karakter,
     integer zelf omzetten}
    if IntToCheck = IntArray[charteller] then
```

```
    begin
      { bijvoorbeeld 128 wordt € }
      result := CharArray[charteller];
      { For loop afbreken }
      Break;
    end // if
  end; // for
end; // IntToCheck
```

Figuur: Code procedure IntToChar

Met behulp van deze functie worden alle velden correct geëxporteerd naar een ODB bestand. Doordat ik dit probleem tegen ben gekomen, heb ik geen tijd meer gehad voor het derde pilotdeel. Na het uitvoeren van het tweede pilotdeel was de geplande tijd voor deze pilot verstreken en ben ik verder gegaan met de volgende pilot. Dit heeft echter geen grote gevolgen gehad omdat het derde pilotdeel betrekking had op een comfort eis.

Pilotdeel: Synchroniseren handheld met HandataLink

Tijdens de bouw zijn er geen problemen voorgekomen. Om een ODB importbestand correct in te lezen, heb ik gebruik gemaakt van de streamreader class. De property encoding van de streamreader heb ik gezet op dezelfde encoding als de streamwriter die ik heb gebruikt bij het wegschrijven naar een ODB bestand.

```
  …
  { Bestand toewijzen, encoding moet overeenkomen met encoding in unit class_HandataExportFileODB }
  ANSI                := Encoding.Get_Default;
  ImportFileStream    := FileStream.Create(Self.Path, FileMode.Open, FileAccess.Read);
  ImportStreamReader := StreamReader.Create(ImportFileStream, ANSI);
   …
```

Figuur: Code creatie StreamReader

Om de onbekende karakters weer om te zetten in de correcte waarde heb ik de functie CharToInt geschreven. Deze functie doet het tegenovergestelde van de functie IntToChar die ik eerder heb beschreven.

Het inlezen en converteren van een veld van het type floating point wordt als volgt afgehandeld. Deze functie maakt gebruik van de functie CharToInt.

```
  …
  Input := GetInput(ImportStreamReader, 8);
  Field := GetFloat(Input);
  …

function THandataImportFileODB.GetFloat(Input : TInput): string;
var
  Bytes      : Array [0..7] of Byte;
  Value      : double;
begin
(*
  De Input is een array met (8) characters. Voor de verdere verwerking moet deze
  array worden omgezet naar een byte array en vervolgens wordt deze byte array
  geconverteerd naar een double en tenslotte teruggegeven als string.
*)
  try
    Bytes := CharToInt(Input);
    { Byte array converteren naar double en teruggeven als string }
    Value  := System.BitConverter.ToDouble(Bytes, 0);
```

```
    result := Format('%1.6f', [Value]);
  except
  // Logboek bijwerken
  end ; // try except
end ;    // GetFloat
```

Figuur: Code floating point converteren

4.2.3 Testen van de realisatie

Het testen van de twee pilotdelen is achtereenvolgens uitgevoerd. Ik ben pas met het ontwerp van het tweede pilotdeel begonnen nadat uit de tests van het eerste pilotdeel bleek dat er correcte ODB bestanden werden opgesteld en geëxporteerd.

Pilotdeel: Synchroniseren HandataLink met handheld

Zoals eerder is beschreven zijn de tests uitgevoerd met de applicatie ODBDump. De tests zijn uitgevoerd voor alle tabellen uit de HandataLink3000 database. De gegevens waren afkomstig uit bestaande testinvoer. Ik heb de tests als volgt uitgevoerd: Ik heb per test de tabelinhoud en de inhoud van het door ODBDump aangemaakte .txt bestand met elkaar vergeleken. Ik heb al beschreven dat dit in het begin niet met elkaar overeenkwam. Uiteindelijk zijn de gegevens uit de tabellen correct naar de verschillende ODB bestanden geëxporteerd.

Pilotdeel: Synchroniseren handheld met HandataLink

Het testen van deze functionaliteit heb ik gedaan met de opgestelde ODB bestanden uit het eerste pilotdeel. De gegevens uit de bestanden heb ik weer in laten lezen in de oorspronkelijke tabellen. De ingelezen gegevens en de gegevens uit de ODB bestanden kwamen overeen. Deze test was echter niet genoeg om de totale functionaliteit te kunnen testen. De ODB bestanden die zijn opgesteld voor gebruik in de WorkAbout bevatten geen veld dat het recordtype aangeeft. Deze bestanden konden dus niet worden gebruikt om te testen of een bestand, dat gemarkeerd is als ‘deleted’, niet wordt opgeslagen in de database.

Om dit wel te kunnen testen, heeft een collega een drietal ODB bestanden samengesteld op een WorkAbout. De records in de ODB bestanden bevatten informatie over opgestelde orders. In één bestand was geen enkele orderrecord gemarkeerd, in één waren ze allemaal gemarkeerd en één bestand had gemarkeerde en ongemarkeerde orderrecords. Om te kunnen testen of de orders correct worden geïmporteerd heb ik een tabel aangemaakt in de HandataLink3000 database. Deze tabel, _waOrderRegel genaamd, bevat alle velden uit het ODB bestand met uitzondering van het veld recordtype.

De tests wezen uit dat records correct ingelezen worden. Dat wil zeggen ongemarkeerde records worden ingelezen en gemarkeerde records worden genegeerd. Tevens heb ik een aantal tests uitgevoerd met betrekking tot tot het verwijderen van een ODB bestand na importeren. Ook deze tests zijn succesvol verlopen.

4.3 Activiteiten pilot Exporteren

De gegevens, die zijn opgesteld door een WorkAbout en ingelezen door HandataLink moeten uiteindelijk worden geëxporteerd voor gebruik in het back-end systeem van de klant. Tijdens de pilotontwerp workshop is besloten dat de reikwijdte van deze pilot lag op de exportfunctionaliteit van orders voor een specifieke klant. Hiertoe is besloten omdat ik in voorgaande pilots gebruik heb gemaakt van de import en ODB bestanden van deze klant tijdens de uitgevoerde tests.

Structuur exportbestand

Het te exporteren bestand heeft een standaardopmaak. Het exportbestand voor orders van de klant ziet er als volgt uit:

```
1;007051173346 ;903436;ICM;PSION ;20041109;1646;PSI
2;007051173346 ;903436;20040315;                    ;20040220;000000000   ;D31
3;007051173346 ;903436;   00896;      5
3;007051173346 ;903436;   01629;      5
3;007051173346 ;903436;   02029;      2
3;007051173346 ;903436;   02030;      2
```

Figuur: Voorbeeld exportbestand

Hierin wordt een regel voorafgegaan door een vaste waarde dat het regeltype aangeeft (1, 2, 3). Vervolgens volgen voor elke regel het ordernummer en het debiteurnummer van de opgestelde order.

Regel 1 heeft vervolgens weer twee vaste waarden (ICM en PSION), gevolgd door de datum van exporteren in het formaat jjjjmmdd en de tijd van exporteren in het formaat uumm. Tenslotte wordt regel 1 afgesloten met de vaste waarde PSI.

Regel 2 gaat verder met de leverdatum, ruimte voor een tekstuele omschrijving, de orderdatum, een vaste waarde (000000000) en tenslotte de bestelcode.

Regel 3 beschrijft de daadwerkelijk bestelde artikelen d.m.v. het artikelnummer en het bestelde aantal. Regel 3 wordt 1 of meer keer in het exportbestand weggeschreven, afhankelijk van het aantal artikelen dat is besteld.

4.3.1 Ontwerpen van de pilot

Use case

Ook bij deze pilot is de use case uit de definitiestudie aangescherpt. Dit heeft geresulteerd in de volgende use case.

Naam	Gegevens exporteren
Samenvatting	Er worden orderregels geëxporteerd vanuit HandataLink voor gebruik in backend van de klant.
Actoren	HandataLink gebruiker.
Aannamen	HandataLink is gestart.

Beschrijving	• De gebruiker maakt aan HandataLink bekend dat er orderregels moeten worden geëxporteerd naar back-end van de klant.
	• HandataLink controleert of er exportgegevens aanwezig zijn.
	• HandataLink controleert of het exportbestand al bestaat.
	  - Indien dit niet het geval is wordt een nieuw bestand gecreëerd
	  - Indien dit het geval is wordt gecontroleerd of er gegevens aan het bestand mogen worden toegevoegd.
	• De orderregels worden geëxporteerd.
	• De geëxporteerde orderregels worden uit de tabel verwijderd.

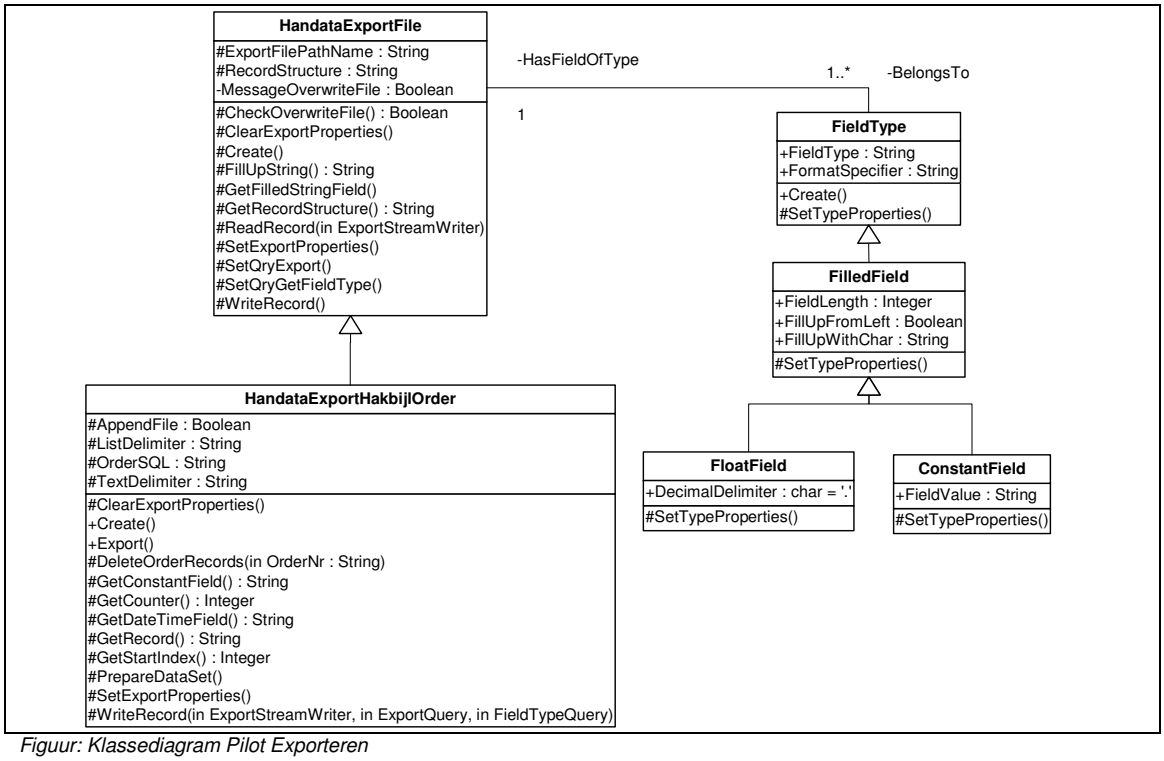
Uitzonderingen	[Geen exportgegevens] Als er na controle blijkt dat er geen exportgegevens aanwezig zijn, zal het exporteren worden afgebroken. [Niet toevoegen] Als blijkt dat er niet mag worden toegevoegd aan een

	reeds bestaand bestand, zal het exporteren worden afgebroken.
Resultaat	De exportgegevens zijn succesvol geëxporteerd vanuit HandataLink naar een vaste lengte bestand voor gebruik in de back-end van de klant.

Figuur: Use case beschrijving Gegevens exporteren

Klassediagram

Ik heb een nieuwe klasse toegevoegd aan het klassediagram dat is beschreven in pilotdeel Synchroniseren HandataLink met handheld uit pilot Synchroniseren. De nieuwe klasse is genaamd: HandataExportHakbijlOrder.



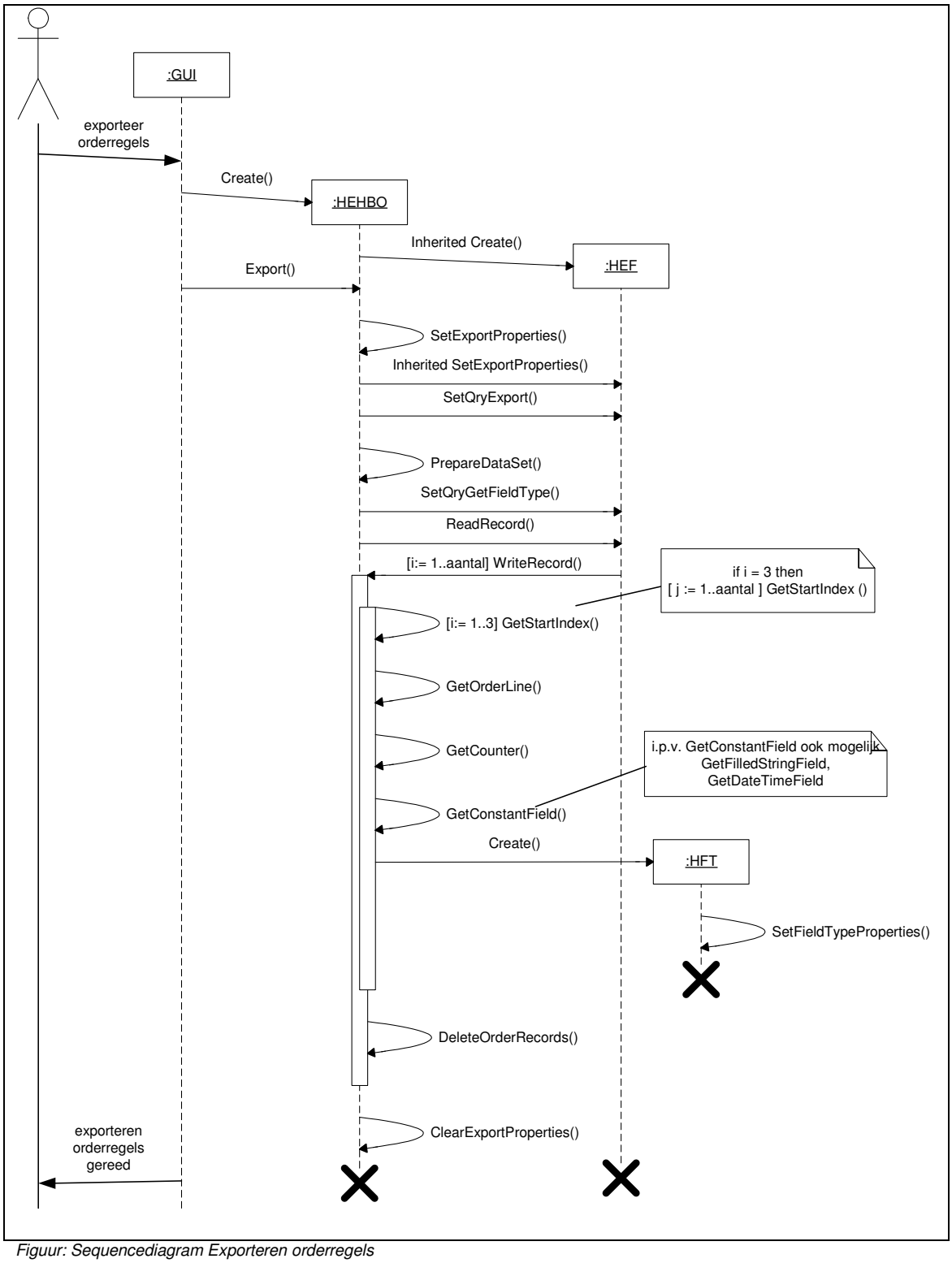
Figuur: Klassediagram Pilot Exporteren

Deze klasse is toegevoegd als een subklasse van HandataExportFile. De attributen uit deze nieuwe klasse zijn afgeleid uit de use case en de structuur van het export bestand. AppendFile is een boolean attribuut dat wordt gebruikt om te controleren of er orderregels aan een bestaand export bestand mogen worden toegevoegd. ListDelimiter beschrijft het scheidingsteken tussen de verschillende velden. OrderSQL is het SQL statement dat de juiste gegevens, in de juiste volgorde, ophaalt uit de HandataLink3000 database. De operaties zijn afkomstig uit het opgestelde sequencediagram.

Tevens is in het klassediagram te zien dat de klasse HandataFieldType en zijn subklassen zijn gewijzigd in de klasse FieldType met subklassen. Het attribuut FormatSpecifier bepaalt nu in welk formaat naar het veld moet worden weggeschreven. Bijvoorbeeld ddmmyyyy voor een Datum. 14 januari 2005 wordt dan weggeschreven als 14012005.

Sequencediagram

Het sequencediagram voor het exporteren van orderregels heeft enige overeenkomsten met het sequencediagram voor het exporteren van ODB bestanden. Toch zijn er enkele duidelijke verschillen. Daarom volgt hier het sequencediagram dat ik heb opgesteld voor het exporteren van orderregels:



Figuur: Sequencediagram Exporteren orderregels

De interacties die tussen de objecten HandataExportFile en HandataExport heb ik niet beschreven in het bovenstaande sequencediagram omdat die gelijk zijn aan de interacties in het eerder opgestelde sequencediagram in Pilot 2 Synchroniseren.

Het bijbehorende scenario is als volgt:

De gebruiker heeft aangegeven dat de orderregels moeten worden geëxporteerd naar een bestand voor gebruik in de back-end van de gebruiker.

Een instantie van HandataExportHakbijlOrder wordt gecreëerd. Dit gebeurt aan de hand van de constructor *create*. Aangezien HandataExportHakbijlOrder een subklasse is van HandataExportFile dient er tevens een instantie van HandataExportFile te worden gecreëerd (*Inherited create*).

Om de gegevens te exporteren wordt de operatie *Export* van HandataExportHakbijlOrder aangeroepen. Deze operatie override de abstracte operatie export van ThandataExport. De operatie export roept onder andere de operatie SetExportProperties aan. Deze operatie stelt de juiste properties in aan de hand van de waarden uit de tabel sExport. Om de juiste properties van de superklassen in te kunnen stellen, worden de operaties SetExportProperties van de superklassen aangeroepen (*Inherited SetExportProperties*).

De query die wordt gebruikt om de te exporteren gegevens te verkrijgen, dient te worden ingesteld (*SetQryExport*). De datum en tijd van exporteren wordt toegevoegd aan de orderregels (*PrepareDataset*).

Om dadelijk de juiste instantie van HandataFieldType te creëren wordt de betreffende query correct ingesteld (*SetQryGetFieldType*).

De orderrecords wordt ingelezen (*ReadRecord*). Elk Orderrecord dat aanwezig is, moet weggeschreven worden naar het export bestand (*i:= 1..aantal] WriteRecord*). Aangezien een orderrecord is opgedeeld in drie delen (Zie structuur Export bestand) wordt voor elk deel de orderregel bepaald en weggeschreven. (*[i:= 1..3] GetStartIndex, GetOrderLine, GetCounter, GetConstantField*). Hierbij moet opgemerkt dat i = 3 wordt uitgevoerd voor alle bestelde artikelen. (Dit is niet in het sequence diagram opgenomen).

Wanneer een order is weggeschreven, worden de bijbehorende records uit de database verwijderd (*DeleteOrderRecords*).

Wanneer het exporteren gereed is, moeten de properties op null worden gezet, dit geld tevens voor de properties van de superklassen. Dit gebeurd aan de hand van *ClearProperties*.

Tenslotte wordt de melding gegeven dat het exporteren van orderregels gereed is.

Figuur: Scenario Exporteren orderregels

Gegevensopslag

De aanpassingen aan het klassediagram hebben ook voor aanpassingen aan de database “Handata” gezorgd. In de tabel sExport heb ik de benodigde velden toegevoegd voor de opslag van het object HandataExportHakbijlOrder. Ik heb een record met de benodigde gegevens toegevoegd, zodat de orderregels kunnen worden geëxporteerd.

De tabel voor de klasse HandataFieldType heb ik uit de database verwijderd en daar is een nieuwe tabel, corresponderend met de klasse FieldType, voor in de plaats gekomen. Voor de volledige tabelopmaak verwijs ik naar de externe bijlage “Pilotontwikkelrapport Pilot 3 Exporteren”.

4.3.2 Realisatie van het pilotontwerp

De realisatie van het pilotontwerp bestond uit het coderen van de nieuwe klassen HandataExportHakbijlOrder en FieldType met zijn subklassen.

De query die alle orderregels van één record ophaalt is als volgt:

```
SELECT OrderNr,KlantCD, VerzendDatum, VerzendTijd, OrderNr, KlantCD, Leverweek,Tekst,
OrderDatum, BestelCD, OrderNr, KlantCD, ArtikelCD, Aantal
FROM _waOrderRegel
WHERE OrderNr = :OrderNr
ORDER BY VerzendDatum DESC
```

Figuur: Query ophalen exportgegevens

De opmaak van het resultaat van deze query komt niet overeen met de structuur van het exportbestand. Onderstaande code beschrijft hoe de verschillende orderregels worden weggeschreven in het exportbestand.

```
...
   { OrderRegel 1 bepalen en vervolgens schrijven }
   StartIndex := 1;
   OrderRegel := GetOrderLine(StartIndex);
   ExportStreamWriter.WriteLine(OrderRegel);

   { OrderRegel 2 bepalen en vervolgens schrijven }
   StartIndex := GetStartIndex(StartIndex);
   OrderRegel := GetOrderLine(StartIndex);
   ExportStreamWriter.WriteLine(OrderRegel);

   ...

  { Alle records langsgaan om de verschillende OrderRegel 3
    te kunnen bepalen en schrijven }
   StartIndex := GetStartIndex(StartIndex);
   While not dmHandata.qryGetHakbijlOrderRegel.Eof do
   begin
     { OrderRegel 3 bepalen en vervolgens schrijven }
     OrderRegel := GetOrderLine(StartIndex);
     ExportStreamWriter.WriteLine(OrderRegel);
     dmHandata.qryGetHakbijlOrderRegel.Next;
   end;
...
```

Figuur: Code Opstellen orderregels

Ik ben geen problemen tegen gekomen tijdens het programmeren van deze pilot.

4.3.3 Testen van de realisatie

Ik heb tijdens het testen van de realisatie op drie dingen moeten letten. Ten eerste moeten de orderregels die zijn weggeschreven naar het bestand overeenkomen met de orderregels in tabel _waOrderRegel uit de HandataLink3000 database. Ten tweede moet de structuur van het opgestelde bestand overeenkomen met de structuur van het export bestand, beschreven in paragraaf 2.3. Als laatste mogen orderregels alleen aan het bestaande export bestand worden toegevoegd als de property AppendFile de waarde ‘True’ heeft.

Uit de, door mij uitgevoerde, tests bleek dat het bovenstaande correct werd uitgevoerd. Uit de tests bleek ook dat de orders van nieuwe klanten ook werden geëxporteerd. Dit is echter niet wenselijk aangezien HandataLink in de toekomst een functionaliteit ‘Orders nieuwe klanten beheren’ zal krijgen. Met deze functionaliteit zal het onder andere mogelijk zijn om de

klantcode van nieuwe klanten te wijzigen, zodat deze overeenkomt met de klantcode in het back-end systeem. Een nieuwe klant wordt bijvoorbeeld aangegeven met klantcode 000000.

Ik moest dus een oplossing vinden waardoor de orders met klantcode 000000 niet geëxporteerd worden. Hier heb ik twee mogelijke oplossingen voor gevonden.

- De eerste oplossing was om de gebruikte query (zie § 4.3.2) aan te passen zodat nieuwe klanten uit het resultaat van de query worden gefilterd zodat orders van nieuwe klanten niet worden geëxporteerd.
- De tweede oplossing was om tijdens het inlezen van het order bestand, aangemaakt door de WorkAbout, de orders van nieuwe klanten in te lezen in een aparte tabel.

In overleg met de opdrachtgever is voor de tweede oplossing gekozen. Het zou handiger zijn als de toekomstige functionaliteit voor het beheren van nieuwe klanten zijn eigen dataset zou hebben. Tevens werd verwacht dat het inlezen van het ODB bestand niet aanzienlijk langer zal duren als de orders voor nieuwe klanten direct in een aparte tabel worden geplaatst.

Ik heb een nieuwe subklasse van HandataImportFileODB gecodeerd. Deze nieuwe klasse, HandataImportHakbijlOrder, heeft een nieuwe property NewClientCode. Door de volgende code wordt een nieuwe klant nu opgeslagen in de nieuwe tabel _waOrderRegelNieuweKlant. Ik heb voor een nieuwe klasse gekozen omdat niet alle in te lezen ODB bestanden orderregels bevatten. Op deze manier kan de klasse HandataImportFileODB nog gebruikt worden om bestanden met andere gegevens (inventarisatie etc.) in te lezen.

```
procedure THandataImportHakbijlOrder.WriteRecord;
var
  i : integer;
begin
  if Trim(ImportTable.FieldByName('KlantCD').AsString) = NewClientCode then
  begin
    { Record hoort bij een Nieuwe klant en wordt opgeslagen in _waOrdersNieuweKlant}
    dmHandata.tblOrdersNewClients.Append;
    for i := 0 to ImportTable.FieldCount - 1 do
    { Waarden van de velden overzetten }
    begin
      dmHandata.tblOrdersNewClients.Fields.Fields[i].Value :=
        ImportTable.Fields.Fields[i].Value;
    end;
    { Record 'posten' in tabel _waOrdersNieuweKlant}
    dmHandata.tblOrdersNewClients.Post;
    dmHandata.tblOrdersNewClients.Refresh;
    { Niet toevoegen in tabel _waOrderRegel }
    ImportTable.Cancel;
  end
  else
  begin
    { Record hoort bij een bekende klant en wordt opgeslagen in _waOrderRegel}
    ImportTable.Post;
    ImportTable.Refresh;
  end;
end;
```

Figuur: Code procedure WriteRecord

4.4 Activiteiten pilot Handheld instellingen

Om HandataLink te kunnen laten communiceren met een handheld, dient de handheld bekent te zijn binnen HandataLink. Op deze manier kunnen er geen ‘vreemde’ handhelds gegevens versturen naar of ontvangen van HandataLink. De reikwijdte van deze pilot is

vastgesteld op het toevoegen, wijzigen en verwijderen van een handheld. Het gaat hier om een specifieke handheld namelijk een Psion WorkAbout. Tevens gaat het hier om een specifiek HandataLink systeem, namelijk HandataLink OrderSysteem. Er dient een GUI te worden ontworpen en gerealiseerd waarin de beschreven functionaliteit kan worden toegepast.

Deze pilot is opgesplitst in de volgende pilotdelen:
- Beheren handheldinstellingen;
- GUI beheren handheldinstellingen.

4.4.1 Ontwerpen van de pilot

Deze paragraaf beschrijf de activiteiten die ik heb uitgevoerd tijdens de ontwerpfase. Als eerste worden de activiteiten beschreven die zijn uitgevoerd ten behoeve van het ontwerp van pilotdeel Beheren handheldinstellingen. Vervolgens worden de activiteiten voor het GUI ontwerp beschreven.

Pilotdeel: Beheren handheldinstellingen

Tijdens de pilotontwerp workshop zijn de basis systeemeisen gedetailleerd uitgewerkt in use cases. Voor het toevoegen van een handheld bijvoorbeeld, had de opdrachtgever nog enkele aandachtspunten met betrekking tot:
- Wanneer een handheld is toegevoegd moet er een ODB bestand worden opgesteld voor gebruik in de WorkAbout.
- Standaardwaarden: Dikwijls heeft een klant een HandataLink systeem in combinatie met één type handheld. Wanneer nu gegevens van een nieuwe handheld toegevoegd worden, dienen deze standaardwaarden door HandataLink te worden ingevoerd;
- Controle op de invoer van:
  - Verplichte gegevens;
  - Serienummer;
  - Numerieke gegevens;
  - Uniciteit van gegevens.

Structuur ODB bestand

Het vanuit een HandataLink OrderSysteem te exporteren bestand voor gebruik in de WorkAbout is in ODB formaat. Het bestand moet uit drie regels bestaan die gegevens bevatten die noodzakelijk zijn voor de correcte werking van een WorkAbout. Het bestand bestaat uit drie regels omdat een WorkAbout een regel langer dan 255 karakters niet kan verwerken. Deze informatie is afkomstig van de opdrachtgever. De inhoud van dit bestand is als volgt:
- Regel 1, bevat de velden:
  - 1 (Regelaanduiding)
  - WANr
  - Wachtwoord
  - LegeRegels
  - Afdrukken
  - Taalcode
  - Vertegenwoordiger
  - OrderKopRegel1
  - OrderKopRegel2
- Regel 2 bevat de velden:

    - 2 (Regelaanduiding)
    - WANr
    - OrderKopRegel3
    - OrderKopRegel4
    - OrderKopRegel5
- Regel 3 bevat de velden:
    - 3 (Regelaanduiding)
    - WANr
    - OrderVoetRegel1
    - OrderVoetRegel2
    - InbelNr
    - Baudrate
    - Poort

De naamgeving van het bestand is afgeleid van het serienummer van de WorkAbout. Het serienummer van een WorkAbout bestaat altijd uit 11 karakters. De naam van het bestand bestaat uit de eerste acht karakters van het serienummer, gevolgd door een punt en tenslotte de laatste drie karakters van het serienummer. Aangezien een serienummer uniek is, is de naamgeving van het bestand ook uniek.

Use cases

De use case handheldinstellingen beheren uit de definitiestudie is opgesplitst in de volgende use cases:

- Use case Toevoegen handheld;
- Use case Tonen handheldgegevens;
- Use case Wijzigen handheld;
- Use case Verwijderen handheld.

Deze use cases zijn opgesteld tijdens de pilotontwerp workshop.

Hieronder wordt de use case voor het wijzigen van een handheld beschreven:

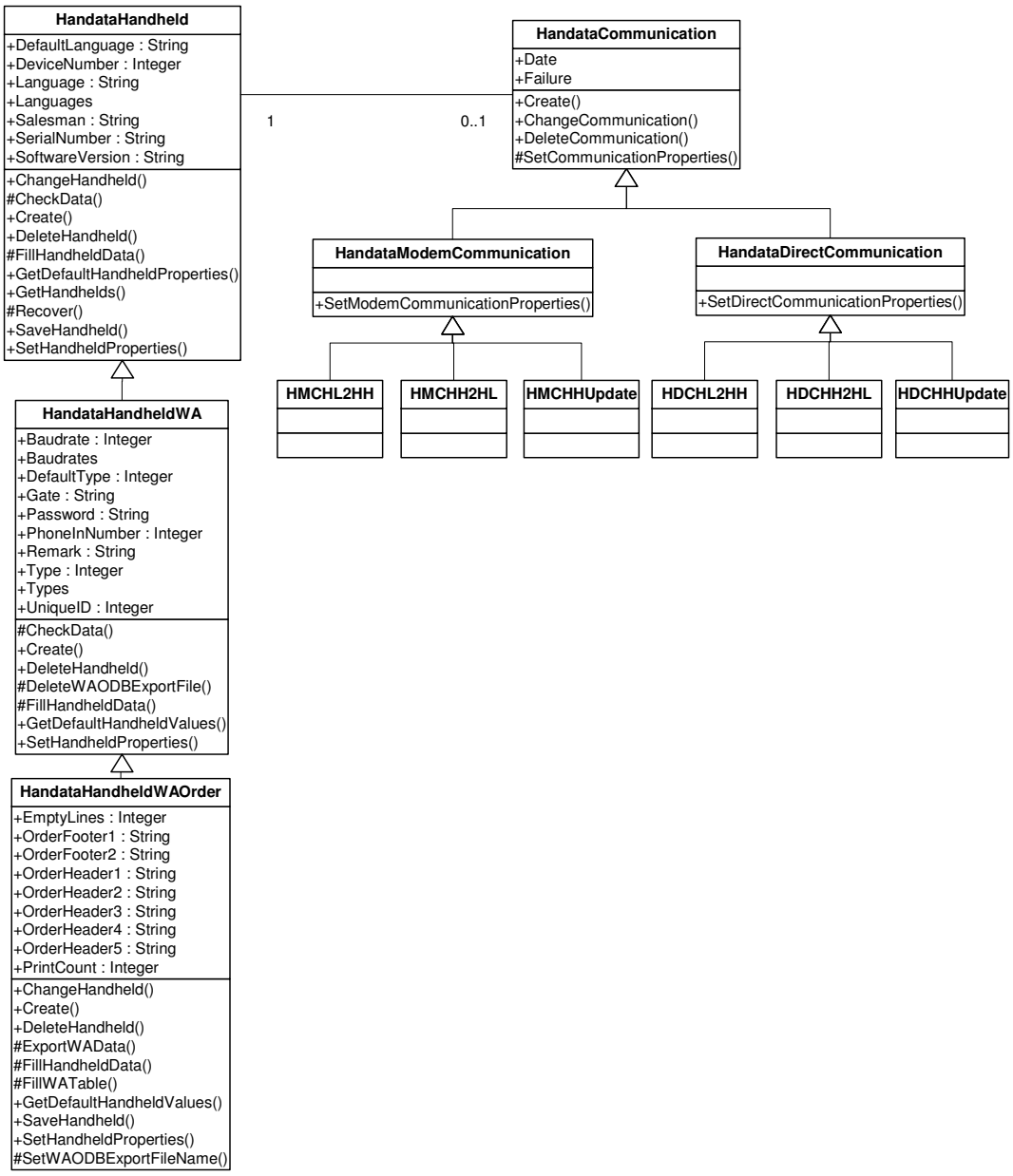
Naam	Handheld wijzigen
Samenvatting	Gegevens van een geselecteerde handheld worden gewijzigd.
Actoren	HandataLink gebruiker
Aannamen	HandataLink is gestart. Optie handheldinstellingen is geselecteerd. Er is een handheld geselecteerd en de bijbehorende gegevens worden getoond.
Beschrijving	(1) De gebruiker maakt kenbaar dat de gegevens van een handheld moeten worden gewijzigd. (2) De gebruiker wijzigt de gegevens van een handheld. (3) De gebruiker selecteert de button Opslaan / OK. (4) HandataLink controleert of de verplichte gegevens ingevoerd zijn. De verplichte gegevens zijn: • WorkAboutnummer • Taalcode • Vertegenwoordiger • Serienummer • Wachtwoord • Type WorkAbout • Unieke ID (5) HandataLink controleert of het serienummer uit 11 karakters bestaat. HandataLink controleert of de numerieke velden daadwerkelijk numerieke waarden bevatten. (6) HandataLink controleert of de unieke velden unieke waarden bevatten. (7) De gegevens zijn gewijzigd. (8) De gemaakte wijzigingen worden doorgevoerd in het ODB bestand.

Uitzonderingen	[Niet alle verplichte gegevens zijn ingevuld] HandataLink toont aan de gebruiker dat niet alle verplichte velden zijn ingevuld. Gegevens zijn niet gewijzigd. Er is geen ODB bestand aangemaakt. [Serienummer bestaat niet uit 11 karakters] HandataLink toont een melding dat het serienummer niet uit 11 karakters bestaat. Gegevens zijn niet gewijzigd. Er is geen ODB bestand aangemaakt. [Numerieke waarden incorrect] HandataLink toont een melding dat de velden die een numerieke waarde moeten bevatten, geen numerieke waarde bevatten. Gegevens zijn niet gewijzigd. Er is geen ODB bestand aangemaakt. [Geen unieke waarden] HandataLink toont aan de gebruiker dat de gegevens niet uniek zijn. Gegevens zijn niet gewijzigd. Er is geen ODB bestand aangemaakt. [Button annuleren geselecteerd] De gebruiker annuleert het wijzigen. De eventuele gemaakte wijzigingen worden niet opgeslagen.
Resultaat	De wijzigingen van de handheldgegevens zijn succesvol opgeslagen. Er is een ODB bestand aangemaakt met de gewijzigde gegevens.

Figuur: Use case beschrijving Handheld wijzigen

Klassediagram

Aan de hand van het op te stellen ODB bestand, de use cases en het later opgestelde sequencediagram heb ik het uiteindelijke klassediagram op kunnen stellen. Het klassediagram voor het beheren van handheldinstellingen ziet er als volgt uit:



Legenda:
HMCHL2HH = HandataModemCommunicationHandataLinkToHandheld
HMCHH2HH = HandataModemCommuncationHandheldToHandataLink
HMCHHUpdate = HandataModemCommunicationHandheldUpdate
HDCHL2HH = HandataDirectCommunicationHandataLinkToHandheld
HDCHH2HH = HandataDirectCommuncationHandheldToHandataLink
HDCHHUpdate = HandataDirectCommunicationHandheldUpdate

Figuur: Klassediagram Pilot Beheren handheldinstellingen

De attributen van de klasse HandataHandheld en zijn subklassen zijn afgeleid van het op te stellen ODB bestand. De klasse HandataHandheld beschrijft de attributen die voor alle typen handheld benodigd zijn. De klasse HandataHandheldWA beschrijft de attributen die nodig zijn voor een handheld van het type Psion WorkAbout. De klasse

HandataHandheldWAOrder tenslotte beschrijft de attributen die een Psion WorkAbout nodig heeft om te kunnen communiceren met een HandataLink OrderSysteem.

De klasse HandataCommunication en zijn subklassen zijn toegevoegd om informatie te kunnen verschaffen over de verschillende communicatiesessies tussen HandataLink en, in dit geval, een WorkAbout. Dit was een extra wens van de opdrachtgever die ik had uitgevoerd nadat beiden pilotdelen waren uitgevoerd. De opdrachtgever wilde informatie over 6 soorten communicatiesessies namelijk:

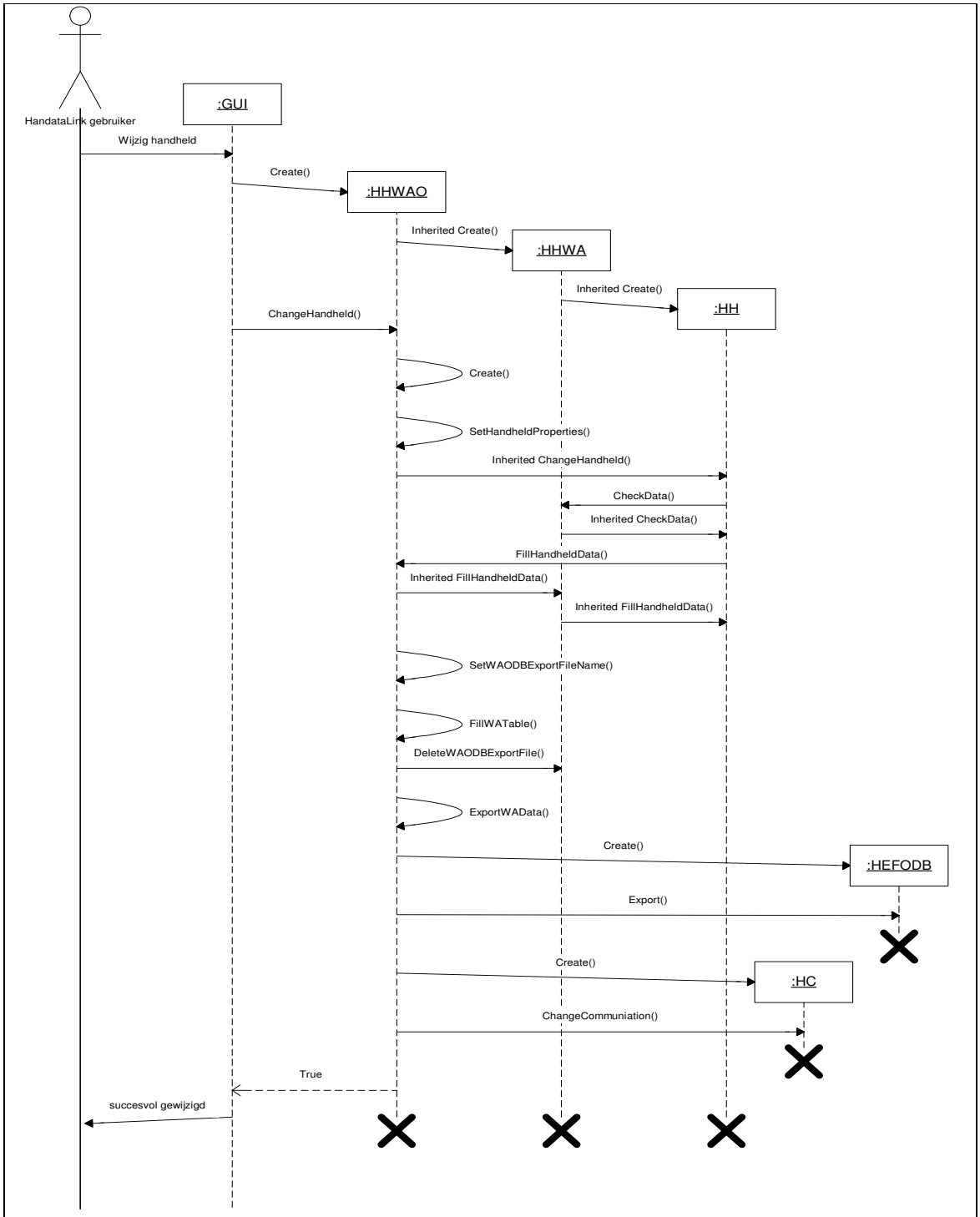
- voor directe communicatie:
  - tussen HandataLink en handheld
  - tussen handheld en HandataLink
  - updates van handheld software
- voor modemcommunicatie:
  - tussen HandataLink en handheld
  - tussen handheld en HandataLink
  - updates van handheld software

De informatie moet betrekking hebben op de datum en tijd van de communicatiesessie en of de communicatiesessie correct is verlopen. Vandaar de attributen Date en Failure.

Bij dit klassediagram heb ik een modeldictionary opgesteld. Voor de modeldictionary verwijs ik naar de externe bijlage Pilotontwikkelrapport pilot Beheren handheldinstellingen.

Sequencediagram

Ik heb twee sequencediagrammen opgesteld. Eén die het tonen van handheldgegevens beschrijft en één die het wijzigen van handheld gegevens beschrijft. Ik heb besloten om geen sequencediagrammen bij de twee overige use cases te maken, omdat die veel overeenkomsten vertonen met het sequencediagram voor het wijzigen van handheldgegevens. Hier volgt het sequencediagram voor het wijzigen van handheldgegevens:



Legenda:
GUI : Grafische User Interface
HHWAO : HandataHandheldWAOrder
HHWA : HandataHandheldWA
HH : HandataHandheld
HEFODB : HandataExportFileODB
HC : HandataCommunication

Figuur: Sequence diagram Wijzigen handheldgegevens

Bij dit sequence diagram heb ik het volgende scenario opgesteld:

De gebruiker heeft via de GUI bepaald dat de gemaakte wijzigingen moeten worden opgeslagen. Een instantie van HandataHandheldWAOrder (HHWAO) wordt gecreëerd (*Create*). Aangezien HHWAO een subklasse is van HandataHandeldWA (HHWA) dient er tevens een instantie van HHWA te worden gecreëerd (*Inherited*). Dit geld ook voor HandataHandheld. De waarden van de properties worden gezet in de GUI.

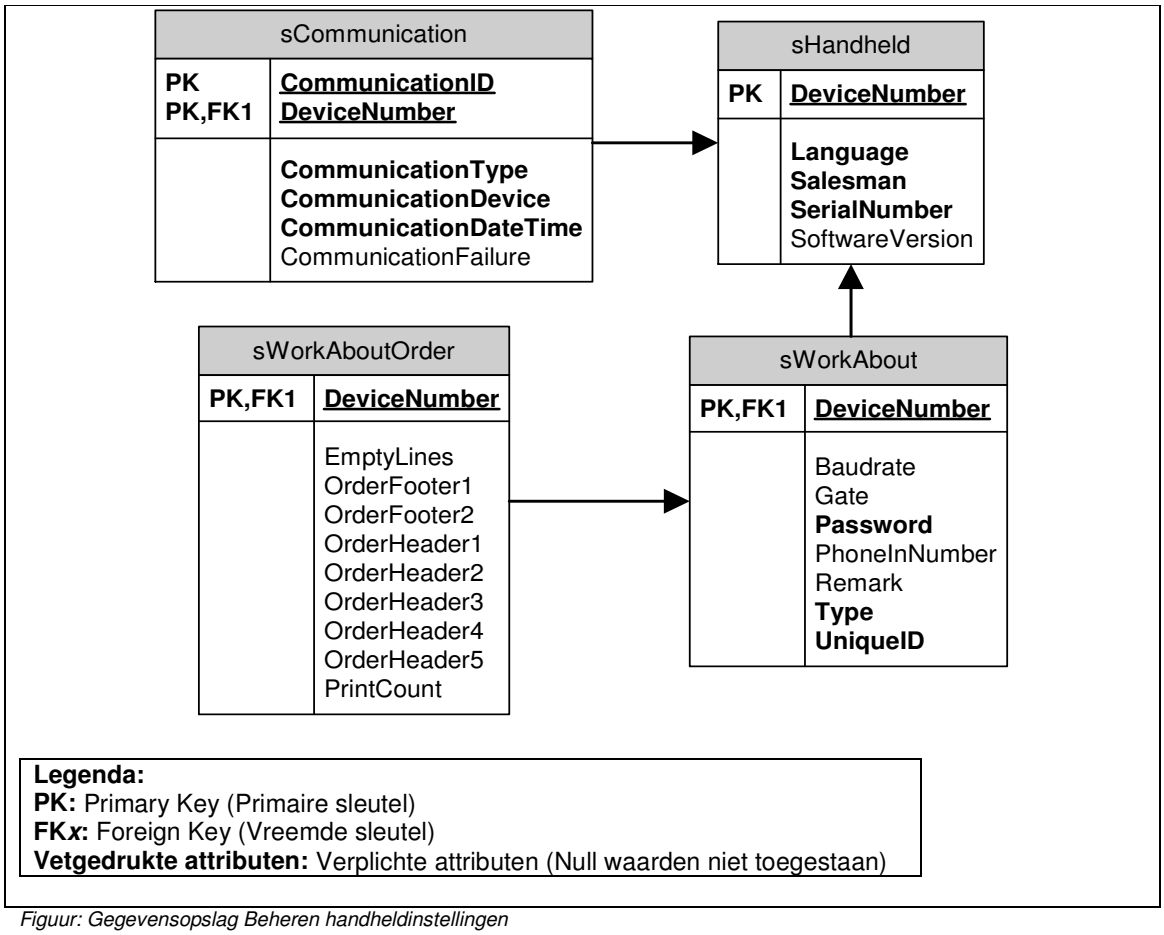
De operatie ChangeHandheld wordt aangeroepen om de waarden op te slaan.
Voordat de wijzigingen daadwerkelijk worden opgeslagen, wordt er een nieuwe instantie van HHWAO gecreëerd (*Create*). Deze instantie onthoudt de originele waarden (SetHandheldProperties) voor het geval de originele waarden moeten worden hersteld. Vervolgens wordt de operatie ChangeHandheld van de superklasse HH aangeroepen. Er worden enkele controles op de gegevens uitgevoerd (*CheckData / Inherited CheckData*). Indien er geen fouten aanwezig zijn, kunnen de gegevens worden weggeschreven naar de database. Dit gebeurt aan de hand van de operatie *FillHandheldData.* Indien dit allemaal zonder fouten is opgeslagen, wordt de naam van het ODB exportbestand gezet (*SetWAODBExportFileName*). De te exporteren gegevens worden in de Exporttabel opgeslagen (*FillWATable*). Het oude ODB exportbestand wordt, indien aanwezig, verwijderd (*DeleteWAOdbExportFile*). Vervolgens worden de zojuist gewijzigde gegevens geëxporteerd naar een ODB bestand. Hiervoor dient een instantie van HandataExportFileODD te worden gecreëerd (*Create*).

Tevens worden eventuele communicatiesessies mee gewijzigd (*ChangeCommunition*). Om deze operatie aan te roepen wordt er natuurlijk eerst een instantie van Communication gecreëerd (*Create*). Indien dit alles zonder fouten is verlopen, wordt True geretourneerd aan de GUI. Op deze manier kan er in de GUI verder worden gewerkt.

Figuur: Scenario Wijzigen handheldgegevens

Gegevensopslag

Onderstaande tabellen zijn toegevoegd aan de database “Handata”. Deze tabellen zijn afgeleid van het eerder beschreven klassediagram. Voor de klasse HandataHandheld en zijn subklassen is besloten om voor iedere klasse een aparte tabel te creëren. Op deze manier is de tabel sHandheld te hergebruiken wanneer er nieuwe subklassen van HandataHandheld worden ontworpen.



Figuur: Gegevensopslag Beheren handheldinstellingen

De attributen DefaultLanguage en Languages worden niet opgeslagen. De waarden van deze attributen worden in code opgeslagen aangezien het niet aannemelijk is dat deze zullen wijzigen wanneer HandataLink bij een klant is ingevoerd.

In de tabel sCommunication is een veld CommunicationDevice opgenomen om te kunnen bepalen of de gegevens bij een communicatiesessie via modem of via directe verbinding horen. Het veld CommunicationType is toegevoegd om te bepalen of het gaat om een communicatie:

- van HandataLink naar handheld;
- van handheld naar HandataLink;
- in verband met een software update voor de handheld.

Pilotdeel: GUI beheren handheldinstellingen

Tijdens de pilot ontwerp workshop heb ik met de opdrachtgever overlegd over de grafische user interface. Hierbij heb ik gevraagd of aan de huidige interface vast moet worden gehouden of dat er een nieuw GUI ontwerp moest worden opgesteld. In overleg met de opdrachtgever is besloten dat ik enkele schetsen zou maken, waarbij ik gebruik moest maken van verschillende Delphi componenten om te kunnen navigeren tussen de verschillende in HandataLink bekende handhelds. Zodat er uit verschillende schetsten een ontwerp gekozen kon worden.

Besloten is dat ik deze schetsen zou maken nadat het ontwerp voor het beheren van de handheldinstellingen was goedgekeurd door de opdrachtgever. Hiertoe is besloten omdat pas na het ontwerp bekend is welke gegevens er in de GUI moeten worden getoond.

Aan de hand van het aantal te tonen gegevens, afgeleid van het ontwerp van het pilotdeel Beheren handheldinstellingen, is besloten om binnen het scherm gebruik te maken van tabbladen. Op deze manier kunnen gegevens die logischerwijs bij elkaar horen, bij elkaar worden getoond en wordt het scherm niet te druk. Er is besloten om het scherm een rustig en overzichtelijk uiterlijk te kunnen geven.

De verschillende Delphi componenten die ik in de schetsen gebruikt heb zijn:

- ComboBox
- TreeView
- Grid i.c.m. Navigatiebalk

De voorkeur van de opdrachtgever ging uit naar een GUI met een ComboBox component. Deze schets heb ik uitgewerkt tijdens de realisatiefase.

4.4.2 Realiseren van het pilotontwerp

Het realiseren van het pilotontwerp heb ik, ten opzichte van het ontwerp, in omgekeerde volgorde gedaan. Ik ben begonnen met het implementeren van de user interface voordat ik aan de implementatie van het pilotontwerp ging werken. Dit heb ik gedaan om vooraf tot een goede overeenstemming met de opdrachtgever met betrekking tot de GUI te komen en omdat het testen van de verschillende functionaliteiten niet gaat zonder een user interface.

Pilotdeel: GUI beheren handheldinstellingen

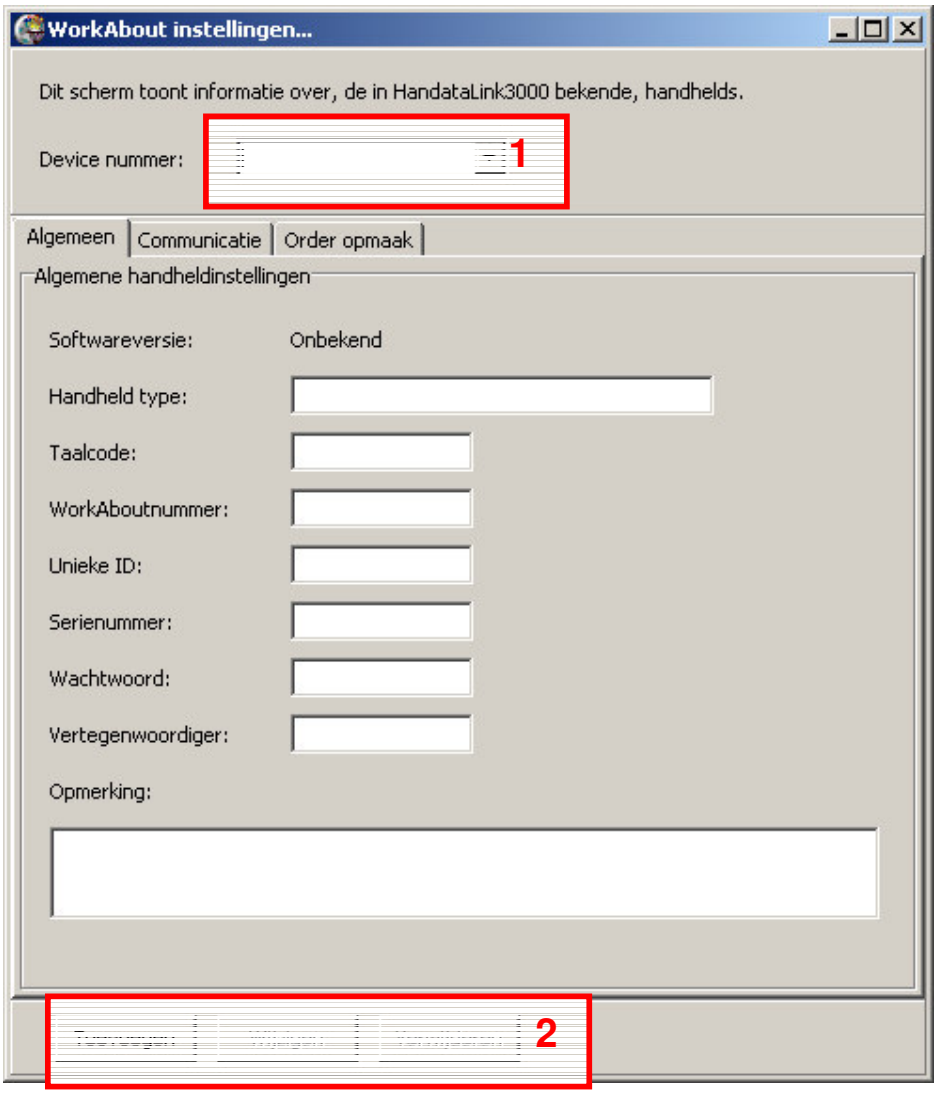
Tijdens het bouwen van de GUI merkte ik een minpunt aan het geheugengebruik van Delphi 8 op. Het zetten van componenten (bijv. labels en editboxen) op een Delphi form duurde lang. Ik heb in het verleden met Delphi 6 gewerkt en hierbij duurde een dergelijke actie minder dan een seconde. Nu was dat al snel tien seconden. Ook het selecteren van enkele componenten om ze uit te lijnen duurde al snel een halve minuut. Dit probleem heb ik voorgelegd aan de opdrachtgever. Aanvankelijk was het niet precies duidelijk wat het probleem was. Toevallig ging ik die week samen met de opdrachtgever naar een Delphi workshop. Daar hebben we het probleem voorgelegd. En het bleek dat het geheugengebruik in Delphi 8 nog niet geoptimaliseerd was door Borland. Ons werd daar ook verteld dat dit in Delphi 2005 is verbeterd. Deze versie van Delphi was wel door Handata besteld, maar was nog niet geleverd. Ik ben dus gewoon doorgegaan met de bouw van de GUI in Delphi 8.

Het scherm handheldinstellingen toont de verschillende gegevens van in HandataLink bekende handhelds. Het is mogelijk om gegevens van een nieuwe handheld toe te voegen, of gegevens van een bekende handheld te wijzigen of te verwijderen.

Het scherm is opgebouwd uit drie delen. Een navigatiegedeelte waarin een handheld kan worden geselecteerd. Een informatiegedeelte waarin de gegevens van de geselecteerde handheld worden getoond. In dit gedeelte worden tevens gegevens toegevoegd of gewijzigd. En tenslotte een gedeelte waarin wordt bepaald of er gegevens toegevoegd, gewijzigd of moeten worden verwijderd.

Het informatiegedeelte is opgebouwd uit drie tabbladen. Deze tabbladen tonen respectievelijk de algemene instellingen, informatie over eventuele communicatiesessies die de handheld met HandataLink heeft gehad en tenslotte de orderopmaak. Er zijn enkele

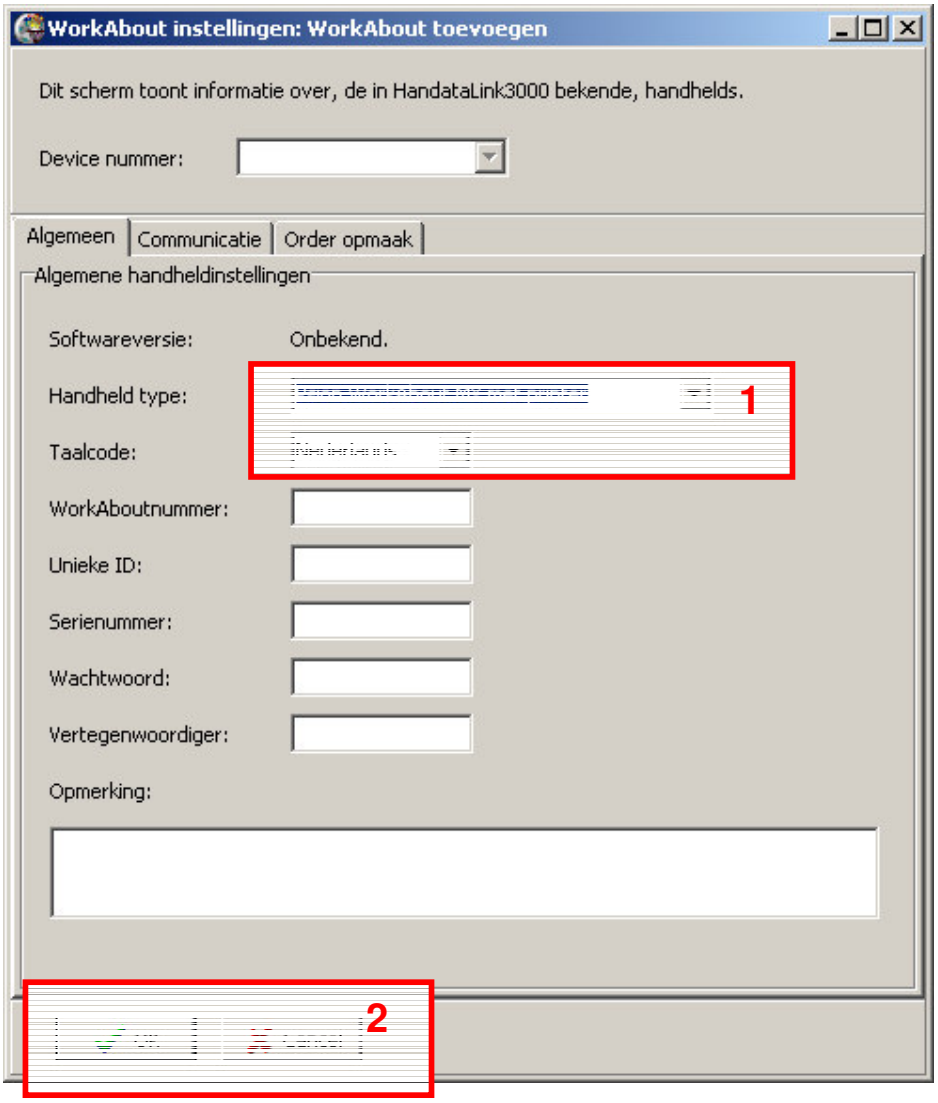
verschillen tussen de tabbladen voor het tonen van de gegevens en die voor het toevoegen / wijzigen van gegevens. Aan de hand van twee screenshots zal ik deze verschillen beschrijven. Als eerste volgt hier een screenshot van het scherm dat de algemene handheldinstellingen toont.



Figuur : Screenshot Tonen handheldinstellingen

Het selecteren van een handheld gebeurt in de combobox in het navigatiegedeelte (1) . Voor de tabbladen die de gegevens tonen geldt dat de buttons wijzigen en verwijderen (2) alleen kunnen worden geselecteerd wanneer er een handheld is geselecteerd. Tevens kunnen de getoonde gegevens op dit moment niet worden gewijzigd.

Nu volgt het scherm dat het toevoegen of wijzigen van handheldgegevens afhandelt:



Figuur: Screenshot Toevoegen / Wijzigen handheldinstellingen

Wanneer er voor de optie toevoegen of wijzigen wordt gekozen, veranderen er enkele zaken binnen het scherm. Ten eerste kan er geen handheld meer worden geselecteerd. Wanneer er voor de optie toevoegen wordt gekozen, worden er enkele standaardwaarden ingevuld (1). Wanneer de optie wijzigen is geselecteerd, worden de originele gegevens getoond.

Tevens zijn de buttons Toevoegen, Wijzigen en Verwijderen vervangen door de buttons OK en Cancel. Afhankelijk van de geselecteerde keuze worden met de button OK de gegevens opgeslagen of gewijzigd. Met de button Cancel wordt teruggegaan naar het tonen van de oorspronkelijke handheldgegevens.

Voor de overige screenshots wordt verwezen naar de externe bijlage “Pilotontwikkelrapport Pilot 4 Handheldinstellingen.

Pilotdeel: Beheren handheldinstellingen

Tijdens het realiseren van dit pilotdeel, werd Delphi 2005 geleverd. Ik heb dit geïnstalleerd en het werken ging een stuk sneller dan met Delphi 8. Voordat ik aan de realisatie ben begonnen heb ik getest of de code van Delphi 8 nog correct werkte, door de tests van de

voorgaande pilots opnieuw uit te voeren. Alles bleek nog correct te werken en ik kon verder gaan met de realisatie van het pilotdeel Beheren handheldinstellingen.

Tijdens de realisatie van dit pilotdeel ben ik als volgt te werk gegaan. Om te beginnen heb ik enkele testwaarden in de tabellen sHandheld, sWorkAbout en sWorkAboutOrder geplaatst. Deze testwaarden representeerden verschillende handhelds.

Vervolgens ben ik begonnen met de daadwerkelijke realisatie van dit pilotdeel. Als eerste heb ik de functionaliteit van het tonen van handhelds geprogrammeerd. Wanneer het scherm getoond wordt, wordt de combobox gevuld met de in de database bekende handhelds.

Figuur: Screenshot Selecteren handheld

Wanneer een handheld uit de combobox wordt geselecteerd, worden de bijbehorende gegevens getoond. Dit gebeurt aan de hand van de volgende code:

```
procedure Tform_HandheldInstellingen.cbDeviceNrChange(Sender: TObject);
begin
  { Er wordt een andere handheld geselecteerd, dus andere gegevens tonen }
  FillFormHandheldInstellingen;
  { Aangezien er een geldige handheld geselecteerd is, kunnen de gegevens
    veranderd of verwijderd worden }
  btnChangeHandheld.Enabled := True;
  btnDeleteHandheld.Enabled := True;
end;
```

Figuur; Code procedure cbDeviceNrChange

Dit heeft geen problemen veroorzaakt. Vervolgens heb ik de functionaliteiten toevoegen, wijzigen en verwijderen van een handheld geprogrammeerd. Hierbij heb ik aanvankelijk geen rekening gehouden met de verschillende controles, zoals beschreven in de use cases, die moeten worden uitgevoerd. Ook het opstellen van het ODB bestand heb ik in deze fase achterwege gelaten. Dit heeft als gevolg gehad dat het snel mogelijk was om handhelds te kunnen toevoegen, wijzigen of verwijderen. Daarna heb ik de verschillende controles ingebouwd. De volgende code beschrijft de controle op de lengte en uniciteit van het serienummer:

```
  ...
  { Controleren of het serialnummer 11 karakters heeft }
  if Length(Self.SerialNumber) <> 11 then
  begin
    ShowMessage(rsCheckSerialNumber);
    result := False;
    { We hoeven niet verder }
    exit;
  end;

  { Controleren of het serienummer uniek is }
  ...
    { Voor elk aanwezig record controleren of het serienummer overeenkomt
```

```
    met een reeds bestaand serienummer }
  For i := 0 to Resource.ResourceDataSource.DataSet.RecordCount - 1 do
  begin
    if Self.SerialNumber =
Resource.ResourceDataSource.DataSet.FieldByName('SerialNumber').AsString then
    { Het serienummer is niet uniek }
    begin
      ShowMessage(rsSerialNumberNotUnique);
      result := False;
      { We hoeven niet verder te zoeken }
      exit;
    end;
    { Volgende record }
    Resource.ResourceDataSource.DataSet.Next;
  end;
 ...
```

Figuur: Code Controle lengte en uniciteit serienummer

Bovenstaande code wordt gebruikt bij de controle op toe te voegen en te wijzigen handheldgegevens. Dit geldt tevens voor de andere controles.

Toen alle controles correct waren geïmplementeerd heb ik de code geschreven die het bijbehorende ODB bestand samenstelt en exporteert wanneer er handheldgegevens worden toegevoegd. Hierbij maak ik gebruik van de eerder gecodeerde klasse THandataExportFileODB. Wanneer er een handheld wordt verwijderd, wordt het eventueel bestaande ODB bestand ook verwijderd.

4.4.3 Testen van de realisatie

Het testen van de realisatie heeft alleen betrekking gehad op het pilotdeel Beheren handheldinstellingen. Aan de hand van de opgestelde use cases heb ik tests opgesteld. Deze tests hadden betrekking op:

- het toevoegen van een handheld;
- het wijzigen van handheldgegevens;
- het verwijderen van een handheld;
- correctheid van het serienummer;
- numerieke waarden;
- compleetheid van gegevens;
- uniciteit van gegevens;
- het te exporteren ODB bestand.

De tests zijn op verschillende momenten tijdens de realisatie uitgevoerd. Nadat een functionaliteit naar mijn idee gereed was, heb ik de bijbehorende test uitgevoerd. Niet alle tests werden direct correct uitgevoerd. Ik heb de code aangepast totdat wel alle tests correct werden uitgevoerd. Hieronder volgt het resultaat van de test die ik heb gebruikt bij het controleren op uniciteit van de gegevens:

Er zijn enkele velden die uniek dienen te zijn. Dit zijn de volgende velden:

- WorkAboutnummer
- Serienummer
- Unieke ID

De volgende handheldgegevens vormen de basis voor deze test:

- Handheld type: Psion WorkAbout MX
- Taalcode: Nederlands
- Workaboutnummer: 5
- Unieke ID: 5
- Serienummer: 56789012345
- Wachtwoord: WA5
- Vertegenwoordiger: RP5

Vervolgens wordt er achtereenvolgens getest met bovenstaande gegevens waarin de volgende gegevens om beurten zijn aangepast:

- WorkAboutnummer: 1
- UniekeID: 1
- Serienummer: 12345678901

Verwachte uitvoer	Correcte uitvoer?
- Het WorkAboutnummer is niet uniek. - Hiervan wordt een melding gemaakt. - De gegevens worden niet opgeslagen. - Er wordt geen ODB bestand aangemaakt. - HandataLink wacht nog steeds op nieuwe invoer	√
- Het unieke ID is niet uniek. - Hiervan wordt een melding gemaakt. - De gegevens worden niet opgeslagen. - Er wordt geen ODB bestand aangemaakt. - HandataLink wacht nog steeds op nieuwe invoer	√
- Het serienummer is niet uniek. - Hiervan wordt een melding gemaakt. - De gegevens worden niet opgeslagen. - Er wordt geen ODB bestand aangemaakt. - HandataLink wacht nog steeds op nieuwe invoer	√

Figuur: Test Uniciteit handheldgegevens

4.5 Onderzoek naar Borland StarTeam

Aangezien de pilot GUI niet is uitgevoerd, viel er een gat in mijn planning. De opdrachtgever vond het belangrijk om te kijken naar de overdracht van de verschillende geprogrammeerde units. Delphi maakt tijdens het compileren van de code van elke unit verschillende tijdelijke bestanden. Hij stelde voor dat ik ging uitzoeken welke bestanden noodzakelijk zijn om te bewaren en welke kunnen worden verwijderd.

Ik heb de opdrachtgever toen gevraagd of ik niet verder moest met de pilotontwikkeling. De opdrachtgever vond dat ik de doelstelling van het project al had gehaald en vond de pilotdelen uit de Pilot Overig niet noodzakelijk om uit te voeren. Ik ben op zijn voorstel ingegaan. Ik heb de opdrachtgever toen gewezen op het versiebeheer programma StarTeam van Borland dat standaard wordt geleverd bij Delphi 2005. We hebben toen afgesproken dat ik onderzoek naar StarTeam zou doen en zou bekijken of StarTeam een applicatie is die binnen Handata kan worden gebruikt.

Ik heb de verschillende meegeleverde handleidingen doorgenomen en gewerkt met de applicatie. Aan de hand daarvan heb ik een document opgesteld dat de globale werking van StarTeam beschrijft en mijn idee hoe deze binnen Handata kan worden gebruikt. Nadat ik dit onderzoek had gedaan ben ik het zelf ook gaan gebruiken. Ik heb de verschillende door mij opgestelde documenten en geprogrammeerde units onder het beheer van StarTeam geplaatst. Dit kan worden gezien als de overdracht van de verschillende producten.

4.5.1 Globale functionaliteiten StarTeam

StarTeam bestaat uit twee verschillende applicaties, een StarTeam Server en een StarTeam Client. Deze Client is na installatie geïntegreerd in Delphi 2005. De server wordt gebruikt voor de centrale opslag van projecten. De client wordt gebruikt voor toegang tot de opgeslagen projecten.

Met StarTeam is het onder andere mogelijk om:

- alle project gerelateerde bestanden op dezelfde plaats op te slaan;
- via LAN, WAN of Internet toegang te verkrijgen tot deze informatie;
- het project teamwork te verbeteren;
- bij te houden wie waar en wanneer aan heeft gewerkt;

Om het ontwikkelproces te ondersteunen, biedt StarTeam naast de mogelijkheid tot versiebeheer, vijf geïntegreerde functionaliteiten, te weten het beheren van:

- Change Requests
- Requirements
- Tasks
- Topics
- Audit Log

Voor een uitleg van deze functionaliteiten verwijs ik naar de externe bijlage “Werken met StarTeam”.

4.5.2 Toepasbaarheid binnen Handata

De bijkomende functionaliteiten (het beheren van change requests, requirements, tasks en topics), zijn voor Handata niet echt van toepassing. Aangezien er binnen Handata niet met grote projectgroepen wordt gewerkt. Er zullen namelijk maximaal twee personen aan hetzelfde project werken (Dit geldt voor een project waarin een HandataLink wordt ontwikkeld). Dit betekent dat de communicatielijnen tussen projectleden kort zijn. Ik kan me voorstellen dat een bedrijf dat werkt met grote projectgroepen wel baat heeft bij de extra functionaliteiten, wellicht omdat de projectgroepen zijn verspreid over verschillende afdelingen of zelfs vestigingen.

De functionaliteit van versiebeheer kan wel van toepassing zijn. Er wordt bij Handata namelijk geen gebruik gemaakt van een versiebeheer applicatie. De activiteiten met betrekking tot het ontwikkelen van HandataLink worden op dit moment door één persoon gedaan. In de toekomst gaat dit veranderen. Het kan dan voorkomen dat meerdere personen aan hetzelfde project bezig zijn. Het gebruiken van een versiebeheer applicatie is dan geen overbodige luxe. Door StarTeam op een correcte manier te gebruiken zal het niet mogelijk zijn dat wijzigingen van het ene projectlid teniet worden gedaan door een ander projectlid.

Het gebruik van StarTeam vraagt wel om een iets andere manier van werken dan op het moment het geval is. Ten eerste zal er een server administrator moeten worden benoemd. De taken van de administrator zullen onder andere bestaan uit het configureren van de StarTeam Server en het toevoegen van gebruikers en gebruikersgroepen met de correcte toegangsrechten.

De gebruikers zullen zichzelf aan moeten leren om met de StarTeam Client te werken en moeten er ook daadwerkelijk mee gaan werken. Een versiebeheer programma is alleen

zinvol als er op de correcte manier mee wordt gewerkt. StarTeam wordt geleverd met een uitgebreide gebruikershandleiding. Hierin worden onder andere de basis principes van het werken met de StarTeam Client uitgelegd.

Concluderend is StarTeam een uitgebreide applicatie waarvan niet alle functionaliteiten even bruikbaar zijn binnen Handata. De functionaliteit van het versiebeheer is echter geschikt voor gebruikt binnen Handata.

4.6 Activiteiten pilot Overig

In het pilotplan worden bij pilot overig de volgende systeemeisen beschreven:

- Bestandslocaties kunnen worden gewijzigd;
- Back-up bestanden kunnen worden geïmporteerd;
- Het bijhouden van een logboek;
- Modeminstellingen beheren;
- Orders nieuwe klanten beheren.

In overleg met de opdrachtgever is gekozen om de functionaliteit van het herstellen (importeren) van een back-up als eerst uit te voeren. Aangezien het opstellen van een back-up bestand (§ 4.2.2) nog niet was gerealiseerd, moest dit eerst gebeuren.

4.6.1 Ontwerpen van de pilot

Aangezien de korte tijd die nog beschikbaar was, is er tijdens de workshop alleen gesproken over het aanmaken en inlezen van een back-up bestand.

Use cases

De workshop heeft geleid tot twee use cases. Eén met betrekking tot het aanmaken van het back-up bestand, de ander met betrekking tot het inlezen van het back-up bestand. Hier volgt de use case voor het aanmaken van een back-up bestand.

Naam	Aanmaken back-up bestand.
Samenvatting	Er is een back-up bestand aangemaakt.
Actoren	HandataLink
Aannamen	HandataLink is gestart. HandataLink staat op het punt om een ODB bestand te importeren.
Beschrijving	(1) Het soort ODB importbestand wordt bepaald aan de hand van de naam van het bestand. - OD = Ordergegevens (2) Er wordt een back-up bestand aangemaakt in de back-up directory van HandataLink. De naam van het bestand is als volgt opgezet: - Onderwerp - Datum (ddmmjj) - Tijd (uummss) - Extensie .bak Onderwerp, datum en tijd worden gescheiden door een underscore. (3) Er wordt een fileheader aan het back-up bestand toegevoegd ter herkenning van het type importbestand. De fileheader wordt afgesloten met #13#10. - OD = 90 (4) De inhoud van het ODB importbestand wordt toegevoegd aan het back-up bestand (5) Het back-up bestand wordt afgesloten

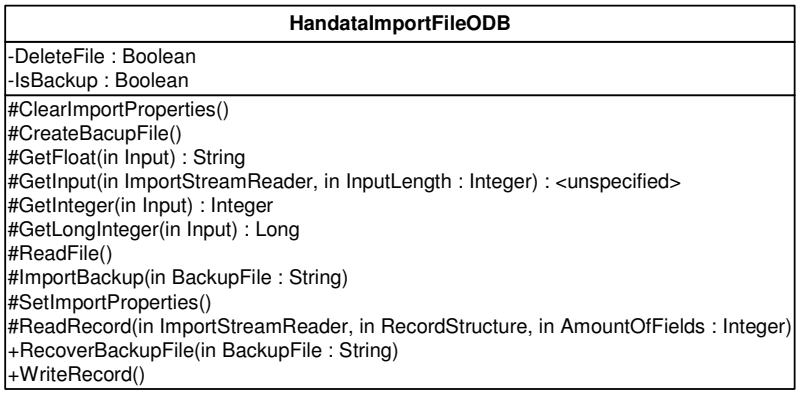
Uitzonderingen	[Kan bestand niet wegschrijven] Er wordt een melding gegeven aan de gebruiker.
Resultaat	Er is succesvol een back-up bestand aangemaakt in de back-up directory van HandataLink.

Figuur: Use case beschrijving Aanmaken back-up bestand

De fileheader die aan het bestand wordt toegevoegd is bedoeld om te kunnen bepalen in welke tabel de gegevens moeten worden ingelezen. Op dit moment wordt er alleen een back-up bestand met orderregels ingelezen. Maar er bestaan ook back-up bestanden met andere gegevens. Met de toevoeging van de fileheader kunnen verschillende back-up bestanden in de correcte tabel worden ingelezen, volgens één ontwerp.

Klassediagram

Het klassediagram dat betrekking heeft op het importeren van gegevens is licht aangepast. De klasse HandataImportFileODB heeft één nieuw attribuut: IsBackup. Dit attribuut is toegevoegd om te kunnen bepalen of een ODB bestand een back-up bestand is of niet. Er zijn drie nieuwe operaties: ImportBackup, CreateBackupFile en RecoverBackupFile.



Figuur: Klassediagram Pilot Overig

Ik heb hier geen sequencediagram van opgesteld, omdat het sequencediagram voor het importeren uit § 4.1.1 in combinatie met de nieuwe use cases genoeg informatie boden. De volgende tekst kan gezien worden als toevoeging op het scenario van het importeren van gegevens.

Wanneer een back-up bestand moet worden ingelezen (IsBackupFile = True), wordt de operatie RecoverBackupFile aangeroepen. RecoverBackupFile filtert de fileheader voordat de operatie ImportBackup wordt aangeroepen. Deze operatie zet de property Path op de juiste waarde. Zodat de locatie van het te importeren bestand bekend is. Vervolgens wordt de operatie Import aangeroepen, die het importeren afhandelt.

Wanneer een ‘normaal’ ODB bestand wordt ingelezen, wordt, voordat het bestand wordt ingelezen, de operatie CreateBackupFile aangeroepen. Het back-up bestand wordt nu aangemaakt. Vervolgens wordt het ODB bestand ingelezen.

4.6.2 Realiseren van het pilotontwerp

De realisatie van het pilotontwerp is een uitbreiding op de reeds geïmplementeerde klasse THandataImportFileODB.

Back-up bestand aanmaken

De procedure ReadFile heeft een nieuwe controle gekregen, die bepaalt of er een back-up bestand moet worden aangemaakt.

```
…
   if not IsBackup then
   { Als we een regulier bestand inlezen moet daar een backup van komen }
   begin
     { Backup bestand aanmaken }
     Self.CreateBackupFile;
   end;
…
```

Figuur: Code controleren van type bestand

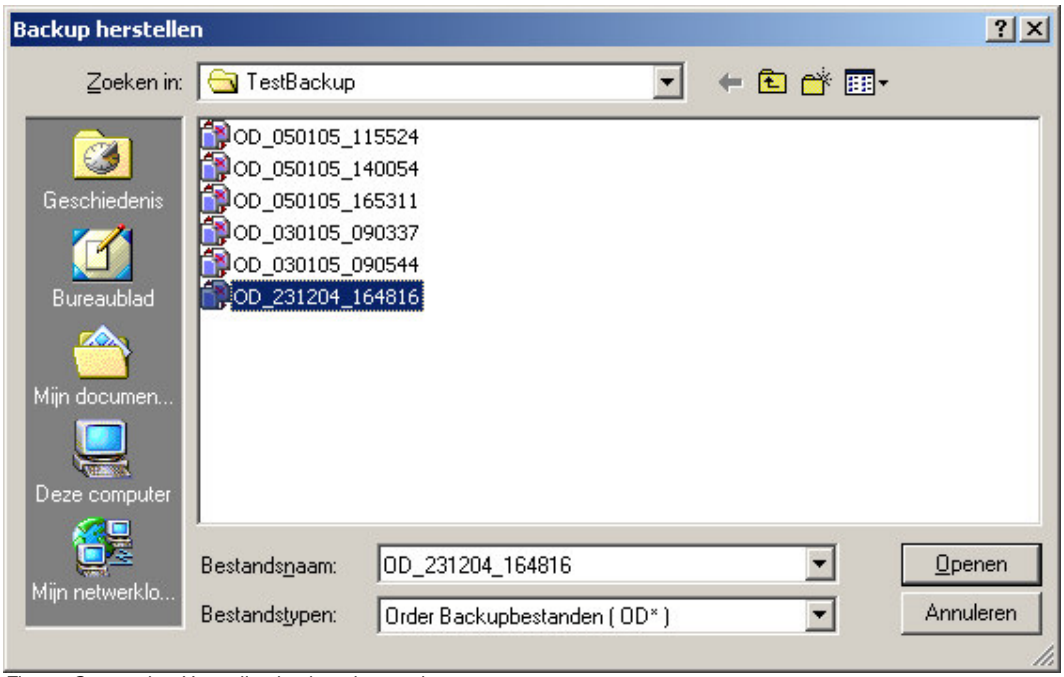
De procedure CreateBackupFile doet een aantal dingen. Ten eerste wordt bepaald om welke gegevens (op dit moment alleen orders) het gaat. Vervolgens wordt er een nieuw, leeg tekstbestand aangemaakt. Aan dit bestand wordt de fileheader toegevoegd. Tenslotte worden de gegevens van het originele bestand weggeschreven naar het nieuwe bestand. Dit resulteert in onderstaande code:

```
procedure THandataImportFileODB.CreateBackupFile;
begin
  { Juiste encodering zetten }
  …
  { Bestandsnaam bepalen }
  …
  { soort ODB bestand bepalen }
  if UpperCase(FileName.Substring(0, 2))  = rsOrderBackup then
  { OD }
  begin
    BackupFileName := UpperCase(FileName.SubString(0, 2));
    FileHeader := '90' + #13#10;
  end
  { Backup Directory bepalen }
  …
  { Backup bestand aanmaken }
  BackupToFileStream   := FileStream.Create(Folder + '\' + BackupFileName, FileMode.CreateNew,
       FileAccess.Write);
  BackupStreamWriter := StreamWriter.Create(BackupToFileStream, ANSI);

 { De backup gegevens komen uit het te importeren ODB bestand }
  BackupFromFileStream := FileStream.Create(Self.Path, FileMode.Open, FileAccess.Read);
  BackupStreamReader   := StreamReader.Create(BackupFromFileStream, ANSI);

  { FileHeader toevoegen }
  BackupStreamWriter.Write(FileHeader);

  { Inhoud ODB bestand toevoegen }
  BackupStreamWriter.Write(BackupStreamReader.ReadToEnd);

  { Netjes afsluiten }
   …
end;
```

Figuur: Code Aanmaken back-up bestand

Back-up bestand herstellen

Binnen delphi bestaat een standaard component TOpenDialog. Met dit component is het heel gemakkelijk om een bestand te selecteren. Door een filter te zetten zijn alleen de back-up bestanden te zien. Dit component heb ik gebruikt om een back-up bestand te selecteren.



Figuur: Screenshot Herstellen back-up bestand

Wanneer er nu een bestand geselecteerd wordt, begint het importeren van het back-up bestand.

```
if BackupFileOpenDialog.Execute then
begin
  …
  Backup.RecoverBackup(BackupFileOpenDialog.FileName);
  …
end; // if BackupFileOpenDialog.execute
```

Figuur: Code Aanroep herstellen back-up bestand

Het back-up bestand wordt nu geïmporteerd. Afhankelijk van het ImportType dat is meegegeven aan de fileheader.

```
  …
  { Afhankelijk van soort backup bestand de objecten voor afhandeling creeeren }
  Case ImportType of
    90:
    begin
      { OD* Order backup bestand (Hakbijl) }
      BackupOrder := THandataImportHakbijlOrder.Create(True);
      BackupOrder.ImportBackup(BackupFile);
      BackupOrder := nil;
    end;
  …
procedure THandataImportFileODB.ImportBackup(BackupFile: String);
begin
  Self.Path := BackupFile;
```

```
  Self.Import;
end;
```

Figuur: Code Aanroep importeren back-up bestand

4.6.3 Testen realisatie

Bij het testen heb ik gebruik gemaakt van de tests, die ik heb beschreven tijdens de pilot Synchroniseren. Uit deze tests bleek dat het aanmaken en herstellen van back-up bestanden correct werden uitgevoerd.

Deel III:

5 Evaluatie

In het laatste hoofdstuk van dit verslag evalueer ik mijn afstudeerperiode. Hiermee krijgt de lezer inzicht in wat ik zelf van het opgeleverde werk vind. Als eerste evalueer ik de producten die zijn opgeleverd. Vervolgens zal ik het door mij gevolgde proces evalueren.

5.1 Productevaluatie

In deze paragraaf evalueer ik de door mij opgeleverde producten. Hiermee moet duidelijk worden in hoeverre de opgeleverde producten aan de gestelde eisen voldoen.

5.1.1 Plan van aanpak algemeen

Het plan van aanpak is aan het begin van het project door de opdrachtgever goedgekeurd en is de gehele periode gebruikt bij het bewaken van de voortgang. Hierdoor is dit product een leidraad geworden van mijn afstudeeropdracht. Ik ben van mening dat dit plan van aanpak inhoudelijk heeft voldaan om de opdracht tot een goed einde te brengen.

5.1.2 Rapport definitiestudie

Het rapport definitiestudie is opgesteld om de opdracht vast te stellen en een eerste aanzet tot een oplossing te geven. De betrokkenheid van de opdrachtgever was in deze fase van groot belang. Daarom heb ik de pilotplan workshop gehouden. Met de informatie die hieruit voortkwam heb ik naar mijn mening een goed definitiestudie rapport op kunnen leveren.

Enkele onderdelen uit het rapport zijn in de beginfase vaak aangepast. Dit geldt vooral voor het klassediagram uit de definitiestudie. Ik heb besloten om na de definitiestudie geen wijzigingen meer in het rapport aan te brengen. Hierdoor is het rapport niet up-to-date. Het rapport bevat een systeemconcept waar ik mee verder kon en het heeft daarom zijn doel bereikt. De pilotontwikkelrapporten vullen het systeemconcept aan. Wanneer het rapport definitiestudie wordt gelezen in combinatie met de verschillende pilotontwikkelrapporten wordt er wel een up-to-date beeld gevormd.

Een onderdeel waar ik minder tevreden over ben is de pilotindeling. De vijfde pilot die ik wilde ontwikkelen was de pilot GUI. Deze pilot is uiteindelijk niet ontwikkeld omdat er weinig te ontwikkelen viel. De eerste drie pilots worden op de achtergrond uitgevoerd en hebben zodoende weinig belang bij een GUI en de pilot Beheren handheldinstellingen had al een GUI. Het is dus een slecht gekozen pilot geweest.

5.1.3 Pilotontwikkelrapporten

Het opstellen van de eerste ontwikkelrapporten vond ik relatief lastig. Vooral het vinden van een correct klassediagram dat in de toekomst kan worden hergebruikt vond ik niet gemakkelijk. Toch is het in samenwerking met de opdrachtgever gelukt om een goed

klassediagram op te stellen. Ook de opgestelde use cases en sequencediagrammen hebben mij voldoende geholpen bij de bouw van de verschillende units.

Ik heb getracht om de opzet van de rapporten steeds hetzelfde te laten zijn. Zo zal het voor een toekomstige ontwikkelaar mogelijk zijn om de opgestelde rapporteren goed te kunnen begrijpen. De overdraagbaarheid van de rapporten wordt hier, naar mijn inzien, mee vergroot.

5.1.4 Geprogrammeerde units

Het doel van de afstudeeropdracht was om aan de hand van de verschillende pilots, verschillende functionaliteiten van een HandataLink Order Systeem te verkrijgen die functioneren in een nieuwe, object georiënteerde omgeving. De objectgeoriënteerde opzet van HandataLink zal in ieder geval het volgende moeten bevatten:

- importfunctionaliteit: het inlezen van stamgegevens vanuit de back-end van de klant;
- exportfunctionaliteit: gegevens die afkomstig zijn van de handheld moeten kunnen worden geëxporteerd voor gebruik in de back-end van de klant;
- synchronisatiefunctionaliteit met een handheld: data van HandataLink beschikbaar maken voor gebruik in de handheld en omgekeerd.
- functionaliteit voor het beheren van de handheldinstellingen, het betreft hier het:
    - toevoegen
    - wijzigen
    - verwijderen
- GUI waarin bovenstaande functionaliteiten kunnen worden aangeroepen;

De door mijn geschreven code heeft ervoor gezorgd dat de eerste vier punten zijn behaald. Ook aan het vijfde punt is voldaan. Voor de import-, synchronisatie- en exportfunctionaliteit is een testapplicatie geschreven waarin de functionaliteiten kunnen worden aangeroepen. De code hiervan kan worden hergebruikt. De functionaliteiten voor het beheren van de handheldinstellingen heeft een eigen GUI waarin handheldgegevens kunnen worden toegevoegd, gewijzigd en verwijderd. Omdat de verschillende klassen in verschillende units zijn geprogrammeerd, zijn de verschillende klassen opnieuw te gebruiken in nieuwe projecten.

Wanneer wordt gekeken naar de systeemeisen kan ik zeggen dat 100% van de basiseisen is behaald. De comforteisen zijn voor de helft behaald en de luxe eisen zijn wegens tijdgebrek niet gehaald. Ik ben dan ook tevreden met de opgeleverde producten. Ook de opdrachtgever is tevreden, dit blijkt uit het feit dat Handata mij een baan heeft aangeboden.

5.2 Procesevaluatie

Het eindproduct is middels een proces tot stand gekomen. Hieronder beschrijf ik mijn mening over het gevolgde proces.

5.2.1 Communicatie met opdrachtgever

Tijdens het project heb ik veelvuldig contact gehad met de opdrachtgever. Tijdens de pilotplan workshop uit de definitiestudie hebben we flink kunnen brainstormen over het systeem en kwamen we al snel op een systeemconcept uit. Tijdens de workshops uit de pilotontwikkelfase hebben we dit systeemconcept verder uitgewerkt en werd er regelmatig

naar de voortgang van mijn project gevraagd. Met vragen of opmerkingen kon ik altijd bij de opdrachtgever terecht. Ik heb dit als prettig ervaren.

5.2.2 Ontwikkelmethode en strategie

Tijdens het project heb ik gebruik gemaakt van de ontwikkelmethode IAD. Dit was de eerste keer dat ik alleen met IAD aan de slag ben gegaan. Om deze reden heb ik, vooral aan het begin van het project, moeten uitzoeken welke activiteiten wel en welke minder of niet geschikt waren om uit te voeren. Dit was in het begin nogal lastig in te schatten, maar des te verder ik kwam in het traject des te beter het ging. Zo heb ik de verschillende activiteiten uit de definitiestudie en de eerste pilotontwikkeling nog per activiteit proberen in te plannen. Ik ben daar vanaf gestapt en heb alleen de verschillende pilotdelen ingepland. Ik vond het namelijk een lastige en te tijdrovende klus om alle activiteiten apart in te plannen. Ik heb timeboxing toegepast op een hele pilot. Wanneer de ingeplande tijd voor een pilot was verstreken ben ik verder gegaan met de volgende pilot.

Ook over de keuze van de ontwikkelstrategie, incrementeel ontwikkelen, ben ik tevreden. Door eerst de systeemeisen volledig in kaart te brengen en deze af te beelden in een systeemconcept kreeg ik al snel een beeld van het te bouwen systeem. Ook het steeds opnieuw verder ontwerpen en bouwen van een deel van het systeem is mij erg goed bevallen. Het ontwikkelen wordt hierdoor afwisselend. Op deze manier ben je niet een lange tijd of met het ontwerp of met de bouw bezig.

Tot de invoering van het systeem ben ik niet gekomen. Dit was van te voren ook afgesproken. Ik heb de belangrijkste functionaliteiten van HandataLink ontworpen, gebouwd en opgeleverd aan de opdrachtgever.

5.2.3 Gehanteerde technieken

Tijdens het project heb ik gebruik gemaakt van de UML modelleringtechniek om invulling te kunnen geven aan IAD. Hieronder wordt deze geëvalueerd:

Ik heb al eerder in projectvorm met UML gewerkt. Dit was gedurende mijn stage. De kennis van UML was alleen wel een beetje weggezakt. Tijdens het opstellen van de verschillende UML diagrammen heb ik gebruik moeten maken van het boek “Praktisch UML 2de Editie”. Uiteindelijk denk ik dat het me gelukt is om correcte diagrammen op te stellen. Ik vind de keuze voor UML een goede geweest. De verschillende use cases kon ik gebruiken bij het opstellen van de testcases. De klasse- en sequencediagrammen waren heel bruikbaar tijdens het programmeren in Delphi omdat Delphi een objectgeoriënteerde ontwikkelomgeving is. De verschillende klassediagrammen konden gemakkelijk worden geïmplementeerd en de sequencediagrammen kon ik goed gebruiken bij het declareren van de verschillende procedures en functies. Waarbij de scenario’s de globale inhoud van de procedures en functies beschreven.

5.2.4 Planning

Ik ben redelijk tevreden over het verloop van het project. De basissysteemeisen en enkele comforteisen heb ik binnen de beschikbare tijd uit kunnen voeren. Ik heb echter wel te weinig rekening gehouden met het schrijven van mijn scriptie. Hierdoor heb ik te veel tijd ingepland voor de pilotontwikkeling en onvoldoende voor het schrijven van de scriptie. Hierdoor heb ik de pilotdelen uit de laatste pilot ‘Overig’ niet allemaal uit kunnen voeren. De laatste twee

weken heb ik namelijk de prioriteit bij het schrijven van mijn scriptie gelegd. Dit is in overleg met de opdrachtgever besloten.

Verklarende woordenlijst

- ANSI: American National Standards Institute.
- ASCII: American Standard Code for Information InterChange.
- CSV: Comma Separated Values. Formaat voor tekstbestanden.
- GUI: Grafische User Interface.
- HHS: Haagse Hogeschool.
- IAD: Iterative Application Development.
- IP-03: Naam van een module van de opleiding IVIT aan de HHS.
- IVIT: Informatie Voorziening en Informatie Technologie
- ODB bestand: Database bestand voor een Psion WorkAbout.
- ODBDump: Applicatie die ODB bestanden omzet in leesbare .txt bestanden.
- OPL: Organiser Progamming Language. Programmeertaal voor Psion handhelds.
- Psion: Psion Teklogix, Leverancier van handhelds.
- SDM I: System Development Method versie 1.
- SO-07: Naam van een module van de opleiding IVIT aan de HHS.
- SO-08: Naam van een module van de opleiding IVIT aan de HHS.
- UML: Unified Modelling Language.
- VCL: Visual Component Library.
- WorkAbout: Type handheld van Psion.

Bijlagenlijst

- Plan van aanpak algemeen
- Rapport definitiestudie
- Pilotontwikkelrapport Pilot 1 Importeren
- Pilotontwikkelrapport Pilot 2 Synchroniseren
- Pilotontwikkelrapport Pilot 3 Exporteren
- Pilotontwikkelrapport Pilot 4 Beheren handheldinstellingen
- Werken met StarTeam
- Pilotontwikkelrapport Pilot 5 Overig

Literatuurlijst

- Pacheco, X., *Delphi for .NET Developer's Guide*, Sams, 2004, (ISBN 0 672 32443 1)

- Tolido, R.J.H., *IAD Het evolutionair ontwikkelen van informatiesystemen*, Academic Service, 1996 (ISBN 90 395 0401 6)

- Wamer, J., Kleppe, A., *Praktisch UML 2e editie,* Addison Wesley, 2001 (ISBN 90 430 0494 4)

- Verschillende internetpagina's