Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Afstudeerverslag

30 mei 2012

Auteur	A.O. Polet
Studentnummer	08046816
Opleiding	Informatica
Instelling	De Haagse Hogeschool
Begeleidend examinator	Dhr. Drs. M. Reijnhoudt
Tweede examinator	Dhr. P.R.C. Breukel
Afstudeerperiode	06-02-2012 t/m 01-06-2012

Referaat

Dit document is geschreven door Axel Polet, vierdejaars student Informatica aan de Haagse Hogeschool in Den Haag. Voor deze opleiding is een afstudeertraject van 17 weken doorlopen bij het bedrijf Artena in Den Haag. De startdatum van het project is 6 februari 2012 geweest en het project is 1 juni 2012 geëindigd.

Het afstudeerproject dient ter afronding van de opleiding Informatica aan de Haagse Hogeschool en heeft als doel de kennis en vaardigheden van de student te bewijzen.

Descriptoren:

- .NET
- Prince2
- Agile
- UML
- DBMS
- SQL Server 2008
- SSIS
- C#
- Plug-in

Voorwoord

Het afstudeerproject is uitgevoerd naar aanleiding van het afronden van mijn opleiding Informatica aan de Haagse Hogeschool. Het project is uitgevoerd bij Artena, gevestigd in Den Haag, op ontwikkelafdeling.

Het verslag is primair geschreven voor de twee examinatoren van de Haagse Hogeschool en de gecommitteerde om inzicht te geven in de door mij uitgevoerde werkzaamheden en de keuzes die ik heb gemaakt hierin. Het verslag is ook geschreven voor andere geïnteresseerden in het project. De lezers dienen kennis te hebben van Informatica.

Gedurende deze periode was Paul Bruggeman de opdrachtgever en Nico Hoekstra mijn begeleider. Ik wil Paul Bruggeman en Nico Hoekstra dan ook bedanken voor de mogelijkheid om dit project uit te voeren. Tevens wil ik Nico Hoekstra en andere betrokken ontwikkelaars bedanken voor de ondersteuning bij de uitvoer van het project.

Ik wens u veel plezier met het lezen van dit verslag.

Axel Polet
30 mei 2012

Inhoudsopgave

Referaat 2
Voorwoord 3
1. Inleiding 6
Deel I – Opstart 7
2. Organisatie 7
2.1 Bedrijfsstructuur 7
2.2 Stakeholders 8
3. Startsituatie 9
3.1 Probleembeschrijving 9
3.2 Probleemspecificatie 10
4. Verwachte situatie 10
4.1 Doelstelling 10
4.2 Verwacht resultaat 11
4.3 Producten: De Artena Data Validator 11
5. Aanpak 12
5.1 Methoden/technieken 12
5.2 Projectinrichting 14
5.3 Planning 17
5.4 Wijzigingenbeheer 18
Deel II – Werkzaamheden 19
6. Opstart & Oriëntatie 19
6.1 Systemen Artena 19
6.2 De Artena Data Validator 21
7. Iteratie 1: Conversiesysteem 22
7.1 Definitiefase 22
7.2 Ontwerpfase 28
7.3 Constructiefase 34
8. Iteratie 2: Validatie-engine 37
8.1 Definitiefase 37
8.2 Ontwerpfase 43

8.3 Constructiefase 49
9. Iteratie 3: Log systeem & Artena Data Validator 51
9.1 Definitiefase 51
9.2 Ontwerpfase 53
9.3 Constructiefase 57
9.4 Systeemtest 61
9.5 Systeemdocumentatie 63
Deel III – Evaluatie 64
10. Procesevaluatie 64
10.1 De productbeschrijving 64
10.2 Planning 64
10.3 Gekozen fasering 64
10.4 Requirements 65
10.5 Ontwerpproces 66
11. Productevaluatie 66
11.1 Artena Data Validator 66
11.2 Rule builder 67
11.3 Documenten 67
12. Beroepstaken 68
12.1 Uitvoeren analyse door definitie requirements (beroepstaak 1.4) 68
12.2 Opstellen gegevensmodel voor database (beroepstaak 2.1) 69
12.3 Ontwerpen bouwen en bevragen van een database (beroepstaak 2.2) 69
12.4 Uitvoeren gegevensconversie (beroepstaak 2.3) 70
12.5 Ontwerpen systeemdeel (beroepstaak 3.2) 70
12.6 Bouwen applicatie (beroepstaak 3.3) 71
13. Gebruikte literatuur 72
14. Woordenlijst 73
15. Figuren 74
16. Bijlagen 75

1. Inleiding

Het doel van dit verslag is om inzicht te geven in het door mij doorlopen proces tijdens de uitvoering van het afstudeerproject bij Artena. In dit verslag is een uitgebreide beschrijving van mijn werkzaamheden aanwezig. In deze beschrijving geef ik ook aan welke keuzes ik heb gemaakt en waarom.

Ik heb in 17 weken een zeer passende oplossing ontwikkeld waarmee de problemen die aanleiding vormden tot uitvoer van dit project aangepakt kunnen worden. De door mij ontwikkelde producten zijn opgeleverd aan Artena en daar goed ontvangen.

Om het gehele proces te beschrijven waarmee ik deze producten heb ontwikkeld is het document ingedeeld in drie delen: Deel I beschrijft de opstart van het project, Deel II bevat een zeer uitgebreide beschrijving van mijn werkzaamheden en Deel III bevat een evaluatie waarin ik mijn mening geef over het doorlopen proces en de producten.

Hoofdstuk 2 bevat een beschrijving van de organisatie van Artena. Hoofdstuk 3 beschrijft de startsituatie en de problemen die aanleiding vormden tot uitvoer van het project. In hoofdstuk 4 worden het verwachte resultaat en de doelstelling beschreven zoals deze aan het begin van het project waren gedefinieerd. Hoofdstuk 5 bevat de door mij gekozen aanpak van het project.

Hoofdstukken 6, 7 en 8 bevatten uitgebreide beschrijvingen van de werkzaamheden die ik heb uitgevoerd om bepaalde (tussen)producten op te leveren. Hierbij worden ook de verschillende ontwikkelfases die ik heb doorlopen uitgelicht.

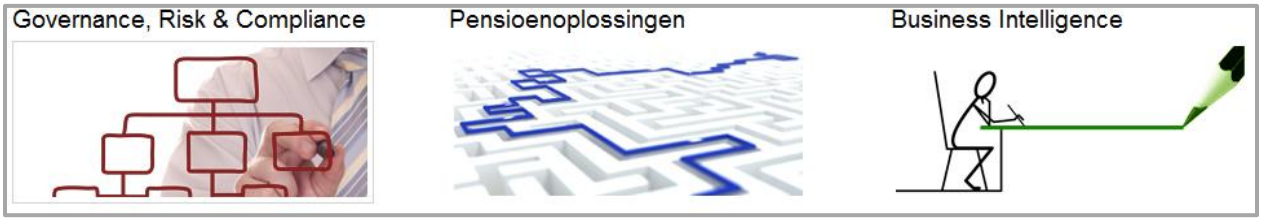
In hoofdstuk 10 wordt een evaluatie gegeven van het proces dat ik heb doorlopen. Hierin geef ik mijn mening op bepaalde procesdelen zoals de fasering en de planning. In hoofdstuk 11 evalueer ik de opgeleverde producten. In hoofdstuk 12 bewijs ik tot slot dat ik de vooraf gedefinieerde vaardigheden bezit aan de hand van de beschreven werkzaamheden en opgeleverde producten.

Deel I – Opstart

2. Organisatie

Artena is een groeiend consultancy/software bedrijf dat gericht is op het leveren van advies en IT oplossingen aan banken, pensioenfondsen, verzekeraars en dienstverlenende bedrijven. Complexe business kennis wordt daarbij gecombineerd met bedrijfskundige en financiële IT-toepassingen. Artena levert hierbij een aantal softwaretoepassingen:

1. **Financial Studio** (Business Intelligence): een uitgebreid datawarehouse dat specifiek is ontworpen voor banken. Dit datawarehouse is een volledige Microsoft toepassing die gebruik maakt van de nieuwste business intelligence componenten (SQL-Server, OLAP server) (www.finarch.com)
2. **Cerrix** (Governance, Risk & Compliance): Een administratiesysteem om operationele bedrijfsrisico's en compliance risico's te beheersen. In dit systeem kunnen bijvoorbeeld processchema's ontwikkeld worden. (www.cerrix.com)
3. **FinCircle** (Pensioenoplossingen): Een applicatiesuite voor pensioenuitvoerders. Het pakket bevat softwaremodules voor pensioensimulatie, pensioenadministratie en communicatie. Diverse pensioenfondsen en bedrijven maken gebruik van deze applicatie.

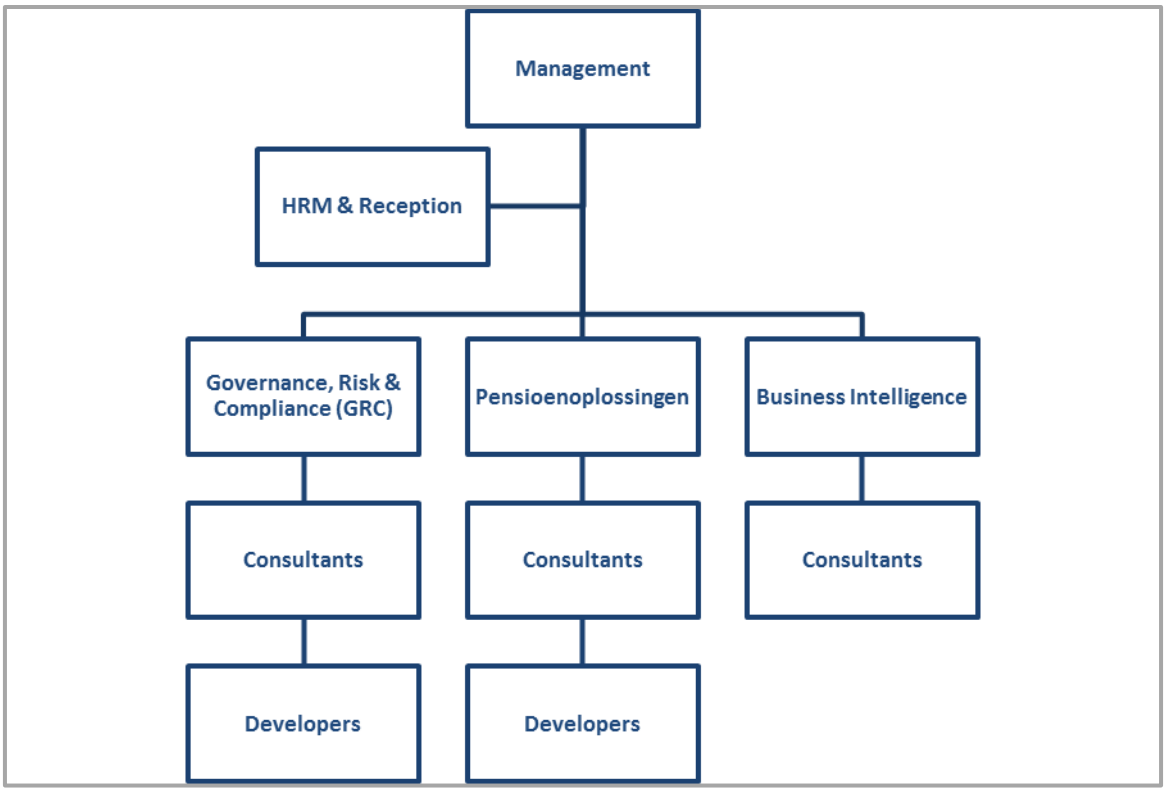


De strategie van Artena omvat het succesvol uitbouwen van de markten voor deze producten, het implementeren van de producten, het doorvoeren van specifieke aanpassingen (feature requests) en het verbeteren/uitbreiden van de producten. Enkele opdrachtgevers van Artena zijn NIBC, Van Lanschot Bankiers, Zorg & Zekerheid en de pensioenfondsen van onder andere Ahold, TNO en de ANWB.

2.1 Bedrijfsstructuur

Voor de onderdelen 2 en 3 (GRC en Pensioenoplossingen) worden requirements (aangeleverd door klanten of het management) omgezet naar functionele ontwerpen. Een dergelijk 'FO' wordt vervolgens geïmplementeerd door één of meerdere ontwikkelaars van Artena. In figuur 1 is de bedrijfsstructuur van Artena beschreven in een organogram.

In deze bedrijfsstructuur maak ik onderdeel uit van het ontwikkelteam voor pensioenoplossingen. In dit team is ook projectbegeleider N. Hoekstra aanwezig.



Figuur 1: Organogram Artena

2.2 Stakeholders

In deze paragraaf worden de stakeholders besproken die betrokken zijn bij de uitvoer van het project. De belangrijkste stakeholder is stakeholder 2, omdat deze stakeholder de probleemeigenaar is en aanleiding heeft gegeven tot uitvoer van het project. Onderstaande beschrijving is een samenvatting van datgene wat in het Plan van Aanpak(Bijlage A) staat vermeld.

Stakeholder 1: Paul Bruggeman, Managing Director Artena
Het belang van deze stakeholder is dat de deliverables van het project aan de vastgestelde eisen voldoen en binnen de gestelde periode (17 weken) wordt opgeleverd.

Stakeholder 2: Nico Hoekstra, Senior Developer
Deze persoon kan worden gezien als de belangrijkste stakeholder betrokken bij het project. Deze stakeholder is degene die op dit moment de handmatige controles uitvoert op de aangeleverde bestanden en de resultaten van die controles rapporteert aan de klanten. Hij is tevens degene die samen met de eerste stakeholder gezien kan worden als de opdrachtgever en probleemeigenaar.

Stakeholder 3: Consultants & Developers FinCircle/Pensioenoplossingen
Deze stakeholder is de groep van ontwikkelaars die zich bezig houden met pensioenoplossingen (FinCircle) bij Artena. Stakeholder 2 is ook onderdeel van deze groep developers, hoewel in een andere rol. Het is mogelijk dat een aantal developers in deze groep al ideeën/gedachten over het uiteindelijke resultaat van het project hebben.

Stakeholder 4: Consultants & Developers Bussiness Intelligence
Deze stakeholder is een groep van consultants en developers die zich bezig houden met Business Intelligence voor klanten van Artena. In het verleden heeft deze groep al enkele projecten uitgevoerd voor klanten die te maken hebben gehad met kwaliteit van data (en het meten ervan/validatie).Bij deze groep zal één consultant het aanspreekpunt zijn, namelijk Dhr. Drs. Ing. P. Schrier.

In het Plan van Aanpak is ook een vijfde stakeholder beschreven, namelijk de klanten van Artena. Deze stakeholder wordt in dit document niet meer beschreven omdat het product waarbij deze stakeholder eventuele belangen zou hebben niet meer ontwikkeld is (zie ook §4.3, deliverable 4)

3. Startsituatie

In dit hoofdstuk zal ik de startsituatie beschreven. Dit houdt in dat de problemen beschreven zullen worden zoals deze aan het begin van het project bij Artena gelden. De probleemstelling zal worden beschreven vanuit het oogpunt van mijn projectbegeleider en het FinCircle systeem omdat hij de problemen bij dit systeem het meest ondervind. De beschreven problemen zijn echter niet beperkt tot dit systeem maar komen ook met regelmaat voor in processen uitgevoerd door andere systemen van Artena.

3.1 Probleembeschrijving

Artena biedt sinds enige tijd de applicatie FinCircle[1] aan voor *pensioenuitvoerders*. Deze applicatie bestaat uit een administratiesysteem, een portal voor werkgevers en een portal voor werknemers. Een aantal klanten maakt alleen gebruik van het werknemersportal.

Belangrijkste functionaliteiten het werknemersportal (ook wel deelnemersportal):

- Biedt werknemers inzicht in de manier waarop ze bij de pensioenuitvoerder geregistreerd staan
- Werknemers hebben de mogelijkheid hun pensioenprognose te bekijken
- Werknemers kunnen verschillende scenario's laten doorrekenen (met behulp van rekenmodellen ontwikkeld door consultants van Artena)

Het werknemersportal maakt gebruik van bestanden die het ontwikkelteam van Artena regelmatig krijgt aangeleverd van de pensioenuitvoerders (de klantbestanden). Deze bestanden worden na aanlevering ingelezen in het werknemersportal door middel via een bepaald importproces.

Het inlezen van bestanden is momenteel *foutgevoelig* en daardoor nog erg afhankelijk van handmatige controles. Bij deze controles komen verschillende soorten fouten in de klantenbestanden naar voren die vervolgens gerapporteerd worden aan de klant zodat de klant deze fouten kan verhelpen. In §3.2 worden de belangrijkste soorten problemen uiteengezet.

[1] http://fincircle.com/Home.aspx.

3.2 Probleemspecificatie

Hieronder staan voorbeelden van de belangrijkste problemen die in de startsituatie speelden bij het inlezen van klantbestanden:

- Het bestandsformaat van het aangeleverde bestand komt niet overeen met het verwachte bestandsformaat
- Gegevensbestanden die qua datalayout niet voldoen aan wat verwacht wordt, bijvoorbeeld een verkeerd aantal kolommen, verkeerde kolomnamen en verkeerde datatypes
- Ontbrekende data: een bestand wat opeens maar de helft van het verwachte/benodigde aantal regels bevat (het verwachte aantal staat geregistreerd in het portal)
- Incorrecte *lookup values*: bijvoorbeeld Man/Vrouw in plaats van M/V
- Datumtijd/valuta notaties: verkeerde cel eigenschappen of verkeerde taalinstellingen, decimaal –en duizendtalseparators, verkeerde weergave van datum en tijd.
- Verkeerde bestandformaten: Excel 1997-2003 versus Excel 2007 of een andere/verkeerde interpretatie van het CSV formaat (wel of niet afsluiten met een puntkomma).

Naast bovenstaande (meest voorkomende) fouten kunnen er problemen ontstaan doordat de aanwezige gegevens onwaarschijnlijk zijn. Er moet dus worden gekeken naar de 'waarschijnlijkheid' van gegevens (bijvoorbeeld een salaris van een werknemer dat 1.000.000 euro is).

4. Verwachte situatie

In dit hoofdstuk bespreek ik de situatie zoals deze verwacht was aan het einde van de projectuitvoering. Hiervoor zullen de doelstelling, het verwachte resultaat en de producten die daarbij opgeleverd moeten worden, worden beschreven.

4.1 Doelstelling

De doelstelling van de opdracht kan worden beschreven in drie onderdelen:

1. Het **verhogen** van de *efficiëntie* van het proces van gegevenstransfer tussen Artena en pensioenuitvoerders voor het aanleveren, controleren en doorgeven van gegevens voor het werknemersportal (en in de toekomst voor andere data uitwisselingen)
2. Het **verbeteren** van de *kwaliteit* van aangeleverde data door het toepassen van inhoudelijke controle op aangeleverde data (zie §3.2)
3. Het **verbeteren** van de dienstverlening naar de klanten toe door de mogelijkheid te bieden aangeleverde data geautomatiseerd te kunnen verwerken waardoor data dus ook vaker aangeleverd kan worden door klanten.

4.2 Verwacht resultaat

Het uiteindelijke resultaat van deze opdracht moet zijn: Een geautomatiseerd systeem wat ondersteuning biedt voor de data uitwisseling tussen klanten van Artena en de uiteindelijke applicatie waarvoor de data bedoeld is. Dit geautomatiseerde systeem moet handmatige interventie overbodig maken en zal een verbetering opleveren in de dienstverlening naar klanten toe dankzij een betere inhoudelijke datacontrole en snellere verwerking. Verwachte functionaliteiten systeem (volledige requirements worden tijdens de opdracht vastgelegd):

1. Mogelijkheid tot 'staging[2]' van aangeleverde bestanden naar één systeem/bestandsformaat zodat de inhoud van verschillende bestanden op eenzelfde manier verwerkt kunnen worden.
2. Valideren van de aangeleverde klantbestanden door meerdere validatieregels toe te passen
3. Vastleggen van de resultaten van de uitgevoerde validaties op een klantbestand
4. Rapportage van fouten naar de klant (pensioenuitvoerder) op het moment dat problemen geconstateerd worden.
5. Flexibiliteit, vooral m.b.t. uitbreiden bestandsondersteuning en validatieregels waardoor ook een eventueel toekomstig gebruik van dit systeem bij andere systemen voor andere klanten van Artena mogelijk is.

4.3 Producten: De Artena Data Validator

Om tot het in §4.2 beschreven resultaat te komen zullen een aantal producten opgeleverd moeten worden. Deze producten kunnen worden gezien als de subsystemen het geautomatiseerde systeem, namelijk de Artena Data Validator. Elk van de subsystemen heeft een eigen set functionaliteiten die bijdragen aan de werking van de Artena Data Validator. Deze producten worden ook wel de deliverables genoemd (naar Prince2, zie §5.1.1). Een volledige beschrijving van de producten inclusief een aantal daarvoor bedoelde modellen uit Prince2(§5.1.1) is vastgelegd in het Plan van Aanpak (bijlage A).

Deliverable 1: Conversiesysteem

Deze deliverable beschrijft het eerste subsysteem in de Artena Data Validator. Dit systeem is verantwoordelijk voor het uitlezen van aanwezige data in een aangeleverd klantbestand om vervolgens deze data op de juiste manier op te slaan zodat de data gevalideerd kan worden. Het uitlezen van deze data wordt 'conversie' genoemd maar manipulaties op de data zullen geminimaliseerd zijn, zodat het bestand in originele staat gebruikt kan worden door de twee deliverable.

Deliverable 2: Validatie-engine

De validatie-engine is het belangrijkste subsysteem van de Artena Data Validator en verantwoordelijk voor het valideren van de inhoud van de uitvoer van het conversiesysteem (de inhoud van de klantbestanden). Een validatie(routine) kan bestaan uit meerdere soorten validaties om de mogelijke problemen (beschreven in §3.2) te achterhalen. Het is belangrijk dat dit systeem een grote mate van flexibiliteit bevat zodat validatietypen toegevoegd, gewijzigd of verwijderd kunnen worden aan een validatieroutine. Het vastleggen van de gevonden problemen is ook onderdeel van de validatie-engine.

[2] http://en.wikipedia.org/wiki/Staging_(data)

Deliverable 3: Log systeem

Het logsysteem van de Artena Validator is verantwoordelijk voor het loggen van systeemfouten die zich kunnen voordoen tijdens conversie -en validatieprocessen. Met behulp van deze logs moet het mogelijk zijn om de problemen die zich voordoen tijdens een conversie –of validatieproces te verhelpen. Mogelijke systeemfouten zijn bijvoorbeeld verbindingsfouten (met een database).

Deliverable 4: Monitoring-tool

De monitoring-tool waarmee rapportages van de uitgevoerde validaties gemaakt zouden worden stond in de planning voor de laatste periode van het project (zie ook §5.3). De ontwikkeling hiervan is echter niet meer uitgevoerd om een aantal redenen (zie ook de planning):

- *De ingeplande tijd van de derde iteratie (voor iteratiesbeschrijving zie §5.2.1) was niet toereikend voor de afronding van de programmatuur voor de Artena Data Validator*
- *De systeemtest(zie §5.1.2) van de Artena Data Validator moest nog worden afgerond*
- *Het constructierapport van de Artena Data Validator moest nog worden opgesteld*
- *De monitoring-tool was van ondergeschikt belang aan de juiste (en verbeterde) implementatie van de Artena Data Validator*
- *De implementatie van de monitoring-tool zou heel erg kunnen verschillen per klant en om dat te onderzoeken, analyseren of ontwerpen was (veel) meer tijd nodig dan de planning toeliet.*

Deliverable 5: Systeemdocumentatie

De systeemdocumentatie bevat een handleiding van het gemaakte systeem (de Artena Data Validator). Daarnaast wordt een technische specificatie toegevoegd waarin alle features tot in detail zijn beschreven. Hierbij geldt wederom dat overleg gepleegd moet worden voor de daadwerkelijke eisen. In de planning wordt echter wel rekening gehouden met dit product.

5. Aanpak

Dit hoofdstuk beschrijft de gekozen aanpak die is gebruikt voor het uitvoeren van het project bij Artena. Deze aanpak is ook vastgelegd in het Plan van Aanpak en is een aantal keren aangepast op basis van nieuwe ervaringen en feedback (zie revisielijst in Bijlage A).

5.1 Methoden/technieken

Voor het project zijn een aantal methoden en technieken gebruikt om de uitvoering te ondersteunen. Sommige methoden/technieken zijn gebruikt voor de planning van het project en andere methoden zijn gebruikt voor de ontwikkeling van de software.

5.1.1 Projectmanagement

PRINCE2. Prince2 projectmanagement beschrijft een wijze waarop een project opgezet en beheerd kan worden. Van deze methode zal ik alleen gebruik maken van de manier waarop de producten worden ingedeeld en beschreven. Hiervoor worden deliverables, productdecompositie diagrammen en

productstroom diagrammen gebruikt. PRINCE2 omvat natuurlijk veel meer dan alleen methoden om producten te beschrijven, de reden dat ik alleen dat specifieke onderdeel gebruik en bijvoorbeeld geen methode om wijzigingen te beheren is dat het project (en het projectteam) niet groot genoeg is voor de toepassing van uitgebreid projectmanagement en projectorganisatie. Daarnaast betreft het project de ontwikkeling van een nieuw stuk software. De literatuur behorende bij PRINCE2 is *[Janssen, P. & Van Craen, J.P.] (2011).*

5.1.2 Softwareontwikkeling

Agile Software Development. Een wijze van indelen van een softwareontwikkelingtraject. Agile is een heel brede definitie van softwareontwikkeling, maar komt het meeste overeen van de wijze waarop ik het afstudeerproject heb uitgevoerd. Deze ontwikkelmethode gaat uit van een iteratieve en incrementele ontwikkeling van software waarbij documentatie geminimaliseerd wordt. Voor meer informatie over de precieze aanpak van het project, zie §5.2.

Unified Modelling Language (UML). UML is gebruikt voor het beschrijven van functionele eisen (UML use cases), het vastleggen van de structuur van de software (UML klassendiagrammen) en voor het vastleggen van de stappen die tijdens een proces binnen het systeem worden uitgevoerd (UML sequentiediagrammen, UML activiteit diagrammen). Voor deze methode zal ik gebruik maken van de literatuur *[Warmer, J. & Kleppe, A.] (2007).*

(Enhanced) Entity Modelling (EER). EER diagrammen worden gebruikt om een conceptueel ontwerp van een database op te stellen. In een EER diagram worden entiteiten, attributen en relaties tussen de entiteiten gebruikt om een datamodel vast te leggen. Vanuit een dergelijk diagram kan een relationeel model worden opgesteld. Literatuur: [Tré, G.] (2007).

Relationeel Representatie Model & Relationeel Implementatie Model (RRM & RIM). Deze modellen worden gebruikt om een relationeel model voor een database op te stellen en kunnen worden afgeleid van een EER diagram.

OO en Design Patterns. Met deze 'technieken' kan software op een juiste wijze geïmplementeerd worden zodat deze zeer uitbreidbaar en onderhoudbaar wordt. Bij Object Oriëntatie moet gedacht worden aan het toepassen van bijvoorbeeld *polymorphisme*. Indien nodig zullen sommige systeemdelen zijn ontwerpen met behulp van design patterns. Een design pattern is een gestandaardiseerde oplossing voor een (veelvoorkomend) probleem bij ontwerp van software. Literatuur: [Shalloway, A & Trott, J.R.](2005).

TMap. Testmethodiek waarin verschillende soorten tests uitgevoerd kunnen worden. Voor dit project is alleen van deze methode gebruik gemaakt om een 'systeemtest' op te stellen, omdat het bevestigen van de aanwezigheid van de gewenste functionaliteiten de hoogste prioriteit had. Performancetests of usabilitytests waren nog niet van belang tijdens dit project. Naast de systeemtest zijn ook unit-tests uitgevoerd waarin specifieke functies van het systeem getest zijn (zie §7.1.3), maar hierbij zijn geen specifieke testmethoden zoals beschreven in TMap toegepast., maar heb ik als ontwikkelaar testgevallen opgesteld(zie §7.1.3). Literatuur: [Pol, M. & Teunissen, R. & Van Veen, E.] (2002).

5.1.3 LINQ-to-SQL in .NET

Zoals in het deel II te lezen is, is het ontwerp van de programmatuur gebaseerd op het ontwerp van de database. In plaats van eerst klassendiagrammen te ontwikkelen en van daaruit een database te implementeren heb ik in dit project eerst de database ontwikkeld en van daaruit de programmatuur gebouwd. Ik heb hiervoor gekozen omdat de wijze waarop alle benodigde gegevens voor een conversie en een validatie opgeslagen zou worden centraal stond. Zo maakt het systeem bijvoorbeeld gebruik van een uitgebreid templatestructuur om klantbestanden te beschrijven (zie §7.2.1).

Om vanuit die database gebruik te maken van de vastgestelde(database)entiteiten heb ik gekozen om een O/R mapper toe te passen. Daarmee kunnen databasetabellen omgezet worden naar objecten. De O/R mapper die ik heb gekozen is LINQ-to-SQL[3] die wordt meegeleverd in het .NET framework van Microsoft(zie ook §7.2.5). Voor elke tabel is een object aangemaakt waarin de kolommen (attributen) en de relaties vanuit de database zijn vastgelegd.

De klassendiagrammen die ik wel zelf heb ontworpen (met behulp van UML) bevatten eigenschappen en gedrag van de objecten die het conversie –en validatieproces controleren.

5.2 Projectinrichting

In deze paragraaf beschrijf ik de wijze waarop ik het project heb ingericht waaronder de fasering, activiteiten en planning van de softwareontwikkeling. De aanpak die ik hierin beschrijf is gebaseerd op de laatste versie van het Plan van Aanpak (bijlage A).

5.2.1 Fasering

Zoals beschreven in §5.1.2 komt de toegepaste ontwikkelmethode het meeste overeen met de Agile ontwikkelmethode. Dit houdt vooral in dat de software iteratief en incrementeel is ontwikkeld. Het aantal iteraties is echter beperkt gehouden om het project beter beheersbaar te maken(zie ook §10.3). In totaal zijn er drie iteraties uitgevoerd in dit project waarin de Artena Data Validator stuk voor stuk(=incrementeel) ontwikkeld is. In elke iteratie zijn verschillende activiteiten uitgevoerd afhankelijk van de planning, de feedback van stakeholders en door ontstane problemen tijdens de uitvoer van eerdere iteraties.

In de eerste twee iteraties ontwikkel in twee subsystemen van de Artena Data Validator en in de laatste iteratie breng ik deze subsystemen samen.

- Iteratie 1: Conversiesysteem
- Iteratie 2: Validatie-engine (en conversiesysteem)
- Iteratie 3: Artena Data Validator (conversiesysteem, validatie-engine & log systeem)

Het is duidelijk te zien dat de eerste en de tweede iteratie betrekking hebben op twee subsystemen van de Artena Data Validator: het conversiesysteem en de validatie-engine. Beide systemen voegen verschillende functionaliteiten toe aan de Artena Data Validator en hebben tegelijkertijd grote

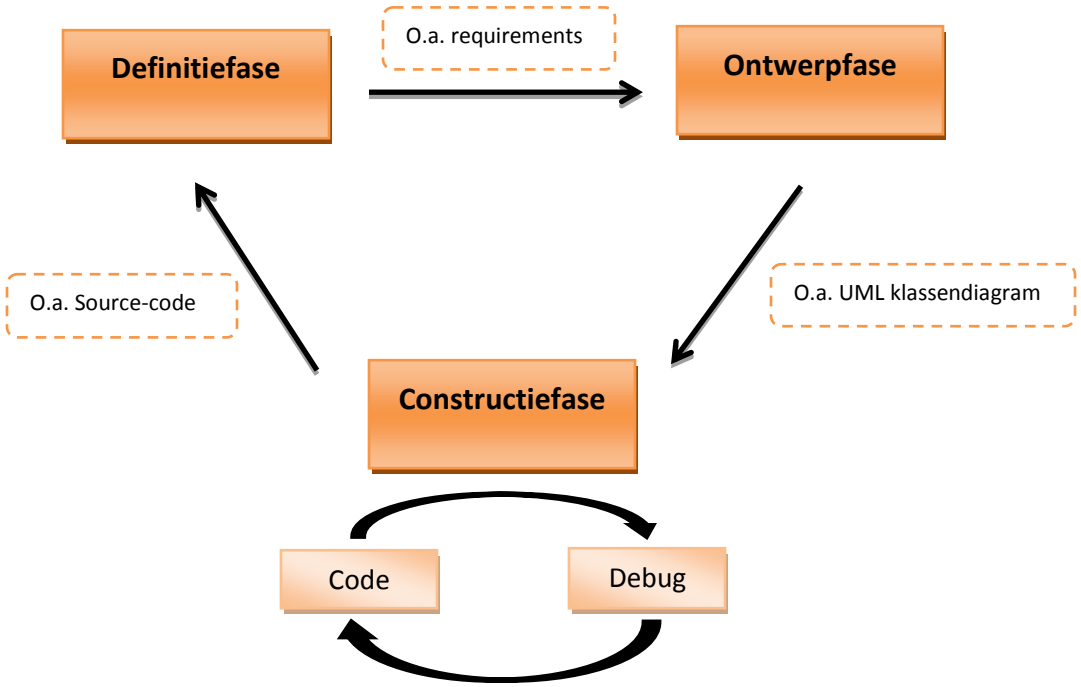
[3] http://weblogs.asp.net/scottgu/archive/2007/05/19/using-linq-to-sql-part-1.aspx

samenhang. Zo is bijvoorbeeld een deel van de invoer van de validatie-engine afhankelijk van de uitvoer van het conversiesysteem.

In de laatste iteratie is een extra 'systeem' toegevoegd, het log systeem, en zijn alle drie de systemen samengevoegd tot de Artena Data Validator. Het log systeem is meer een registratiemiddel dan een systeem en wordt gebruikt door de twee andere (sub)systemen van de Artena Data Validator om systeemfouten (mochten die zich voordoen) vast te leggen.

In elke iteratie zijn dus extra functionaliteiten gemaakt (incrementeel) en zijn de nodige aanpassingen doorgevoerd op al bestaande functionaliteiten (iteratief). Op deze manier is het systeem stuk voor stuk opgebouwd en kunnen veranderingen die zich voordoen aan eisen en/of ontwerpdelen worden opgevangen. In elke iteratie was dus ook de mogelijkheid aanwezig om wijzigingen uit te voeren op eerdere iteraties.

Figuur 3 toont het faseringsmodel waar ik mijzelf aan heb gehouden binnen één iteratie.



Figuur 2: Iteratiemodel

Het model toont een (vrij standaard) ontwikkelcyclus die ik toegepast heb tijdens de ontwikkeling van de producten. Van deze cyclus zijn in elke iteratie enkele of alle activiteiten uitgevoerd, hiervoor zie de planning in §5.3.

Bij elke iteratie zijn twee documenten opgesteld, een iteratieplanning –en een iteratieresultaat document. In het iteratieplanning document staan alle activiteiten die ik voor de betreffende iteratie verwachtte uit te voeren. In het iteratieresultaat document staat het resultaat van de iteratie waaronder behaalde en niet-behaalde resultaten. Al deze documenten zijn samengebundeld in één iteratiedocument (bijlage B).

5.2.2 Fases & Activiteiten

Definitiefase (Onderzoek & Analyse)

Deze fase heeft betrekking op de het uitvoeren van onderzoek en analyse. Activiteiten die binnen deze fase worden uitgevoerd:

- Oriëntatie op bestaande systemen van Artena
- Een user story waarin de problemen die de aanleiding tot uitvoer van het project vormen uiteengezet worden.
- Definiëren van requirements door gesprekken te voeren met de beschreven stakeholders
- Specificeren van functionele eisen in UML use case diagrammen.

Bovenstaande gegevens zullen worden samengevoegd in een *Analyserapport*.

Ontwerpfase

Deze fase omvat het opstellen van ontwerpdelen die uiteindelijk in het ontwerprapport terecht komen. In het ontwerp wordt aangetoond dat de, in het analyserapport vastgestelde, eisen zijn meegenomen in het ontwerp door vanuit de gebruikte ontwerpmodellen terug te verwijzen naar de requirements. Bij het databaseontwerp wordt gebruik gemaakt van 4 modellen:

- (Enhanced) Entity Relationship Diagram(*ERD*) voor het ontwerp van een conceptueel gegevensmodel waarin bijvoorbeeld validatieregels worden opgeslagen.
- Relationeel Representatie Model (*RRM*) voor de relationele definitie van het gegevensmodel.
- Relationeel Implementatie Model (*RIM*) voor een specifiek DBMS.
- Database diagram (Visio 2010 model) voor de grafische weergave van het ontwerp.

Bij het ontwerpen van de programmatuur wordt gebruik gemaakt van onderstaande modellen:

- UML klassendiagram om de structuur van de programmatuur te beschrijven.
- UML sequence diagram om het gedrag van de programmatuur te beschrijven.

Het resultaat van deze fase is een *Ontwerprapport* dat bovenstaande gegevens bevat.

Constructiefase

Deze fase bestaat uit de implementatie van de ontwikkelde ontwerpen uit de ontwerpfase. In het geval van de ontwikkeling van een datastructuur zal een (deel van een) database gebouwd worden in het gebruikte Database Management System: SQL Server 2008.

In het geval van de implementatie van de programmatuur zal gecodeerd en gedebugd worden. Dit houdt in dat de klassen beschreven in de klassendiagrammen van het ontwerp geïmplementeerd worden met behulp van Microsoft Visual Studio 2010.

Tijdens deze fase kunnen ook een aantal modellen opgesteld worden die gedetailleerde (technische) beschrijvingen van de applicatie (en code) bevatten. Dit zijn bijvoorbeeld UML klassendiagrammen op applicatieniveau of UML activiteit diagrammen. Hiervan kan een *Constructierapport* worden gemaakt.

5.3 Planning

Met behulp van de gestelde tijdsduur van het project (17 weken), de productbeschrijving, productdecompositie, productstroomschema en fasering is een planning opgesteld. De tabel hieronder toont de volledige planning (afkomstig uit het Plan van Aanpak, Bijlage A).

Activiteit	Betrokken deliverables	Startdatum	Einddatum	Week-nummer
Voorbereiding				
Plan van aanpak		06-02-2012	10-02-2012	1
Requirements conversiesysteem (bestandslezers)	1-Conversiesysteem	06-02-2012	10-02-2012	1
Vergaren kennis infrastructuur systemen Artena		06-02-2012	10-02-2012	1
Iteratie 1: Conversiesysteem				
Requirements Onderzoek bestandslezers	1-Conversiesysteem	13-02-2012	17-02-2012	2
Requirements Ontwerp	1-Conversiesysteem	20-02-2012	24-02-2012	3
Ontwerp Constructie	1-Conversiesysteem	27-02-2012	09-03-2012	4-5
Iteratie 2: Validatie-engine				
Onderzoek & Analyse: Definities validatieregels Definities resultaten Validatieflow	1-Conversiesysteem , 2-Validatie-engine	12-03-2012	16-03-2012	6
Requirements	1-Conversiesysteem , 2-Validatie-engine	19-03-2012	23-03-2012	7
Ontwerp Implementatie validatieroutine	1-Conversiesysteem, 2-Validatie-engine	26-03-2012	06-04-2012	8-9
Iteratie 3: Logging & Afronding ADV				
Requirements & Ontwerp o.b.v. feedback iteratie 2	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem	09-04-2012	13-04-2012	10
Requirements & Ontwerp log systeem Constructie	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem	16-04-2012	27-04-2012	11-12
Iteratie 4: Monitoring & Systeemtest				
Onderzoek & Analyse, Requirements	*4-Monitoring tool*	30-04-2012	04-05-2012	13

Ontwerp *Constructie* Systeemtest ADV	1- Conversiesysteem, 2-Validatie-engine, 3-Log systeem	07-05-2012	18-05-2012	14-15
Uitloop/Afronding				
Uitloop	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem, 4-Monitoring-tool	21-05-2012	25-05-2012	16
Nazorg	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem, 4-Monitoring-tool	28-05-2012	01-06-2012	17

In bovenstaande planning is product 4 ‘doorgestreept’ omdat deze niet meer is meegenomen in het project (zie ook §4.3). Over alle wijzigingen (er zijn meerdere versies van het Plan van Aanpak) die zijn gedaan aan de projectinrichting en de planning is een uitgebreide beschrijving aanwezig in de procesevaluatie van hoofdstuk 10.

5.4 Wijzigingenbeheer

Een ander belangrijk onderdeel van de projectinrichting is het uitvoeren van wijzigingsbeheer op de opgestelde producten en systeemdelen. Door het uitgebreid bijhouden van de wijzigingen kan een goed inzicht worden gecreëerd in het ‘wat’ en ‘waarom’ tijdens de ontwikkeling van bepaalde documenten zoals requirements en ontwerpen.

Voor alle documenten en onderdelen betrokken bij het uitgevoerde project is versiebeheer uitgevoerd met behulp van TortoiseSVN versiebeheer. Voor de documenten is ook extra versiebeheer uitgevoerd door aan het begin van een document een revisietabel vast te leggen.

De tabel hieronder toont de structuur van het versiebeheer gebruikt tijdens het project:

Eigenschap	**Beschrijving**
0.1	Dit is de startversie van elk document. Dit nummer geeft aan dat het een eerste conceptversie van een document betreft
0.11	Indien zich een kleine wijziging voordoet aan bestaande data dan zal het versienummer met 0.01 worden verhoogd.
0.2	Indien zich een grote wijziging voordoet, waarbij significante veranderingen optreden of waarbij een aantal onderdelen wordt toegevoegd zal het versienummer met 0.1 worden verhoogd.
1.0	Wanneer een eerste versie klaar is voor oplevering, dan zal deze het versienummer 1.0 meekrijgen. Deze versie bevat dan in ieder geval alle onderdelen die het product moet bevatten. Er kunnen echter nog wel grote wijzigingen of toevoegingen worden gedaan na deze versie (wanneer dat nodig is).

Deel II – Werkzaamheden

6. Opstart & Oriëntatie

In dit hoofdstuk beschrijf ik de gedane werkzaamheden tijdens de opstart van het project en de oriëntatie binnen Artena. Voorafgaand aan de opstart van het project waren veel aspecten van het project vastgelegd in het Plan van Aanpak. Het was duidelijk welke deliverables opgeleverd moesten worden aan Artena en de globale functionaliteiten van de deliverables waren vastgelegd in het afstudeerplan.

In de loop van het project bleek dat de indeling van de deliverables en hun onderlinge samenhang onvoldoende was vastgelegd, dit is uitgebreid beschreven in de procesevaluatie.

6.1 Systemen Artena

Zoals beschreven in hoofdstuk 2 biedt Artena een aantal systemen/services aan hun klanten. Bij deze systemen worden grote hoeveelheden data gebruikt. De gegevens worden vaak geüpdatet en zijn aan veel veranderingen onderhevig.

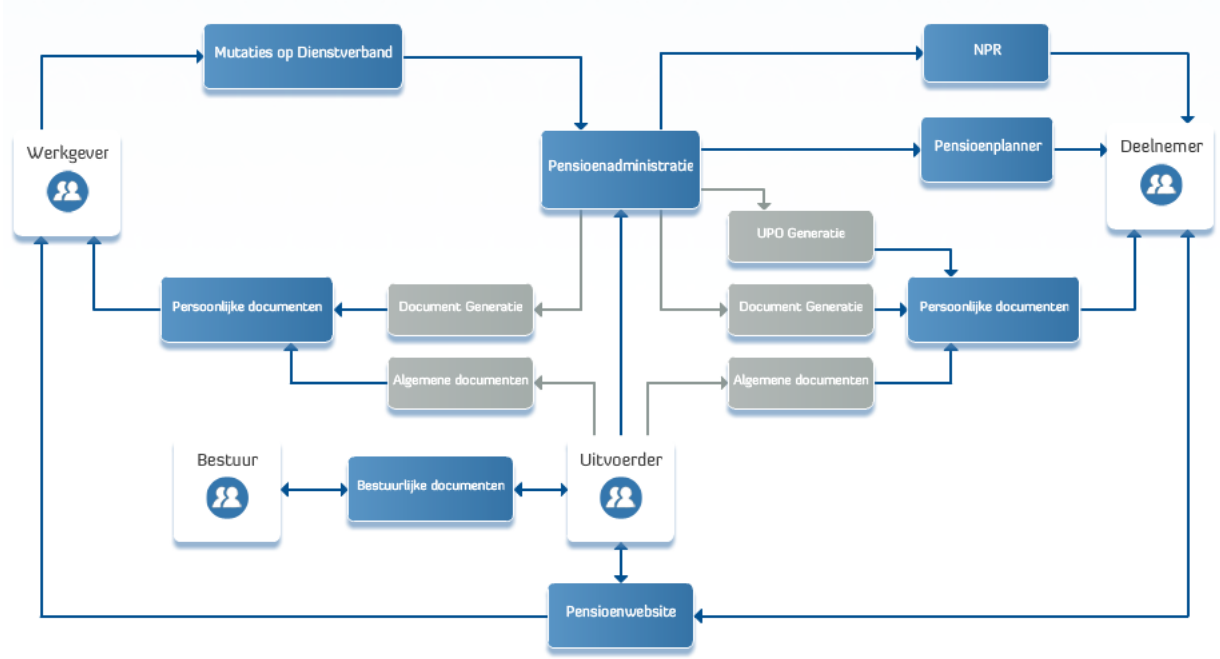
Wat er gebeurt met de data zodra die worden gebruikt in een systeem van Artena valt buiten de scope van dit project. Zoals aangegeven in de probleembeschrijving zijn de steeds weer voorkomende fouten bij het importeren van data in een systeem van Artena de aanleiding voor de uitvoering van dit project.

Een belangrijk systeem (waar stakeholder 2 vaak mee bezig is) is FinCircle, een volledige pensioenoplossing voor werknemers (deelnemers), werkgevers en pensioenuitvoerders (bijvoorbeeld fondsen). De primaire processen va FinCircle zijn getoond in figuur 3, waarvan de belangrijkste systemen:

Deelnemerportal (pensioenplanner)
Gebruikt door pensioendeelnemers (werknemers) aangesloten bij verschillende pensioenfondsen. Het portal biedt toegang tot een pensioenplanner waarin gegevens van het pensioenadministratiesysteem weergeven worden.

Werkgeverportal (mutaties op dienstverband)
Via dit portal kunnen werkgevers mutaties op dienstverband van hun werknemers invoeren in het pensioenadministratiesysteem.

Pensioenadministratie
Het centrale pensioenadministratiesysteem van Artena waarin gegevens van verschillende bronnen samenkomen en verwerkt worden. Onder andere de *pensioenuitvoerder* levert klantbestanden aan bedoeld voor dit systeem.

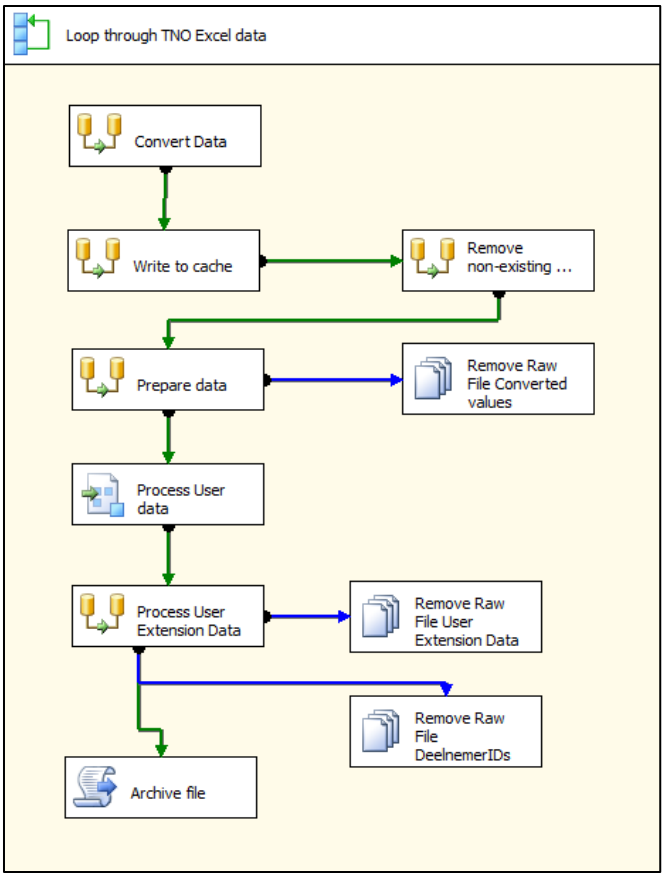


Figuur 3: Softwareprocessen FinCircle

Het systeem dat ontwikkeld is tijdens het project is van toepassing op de validatie van klantbestanden alvorens een import van deze gegevens wordt uitgevoerd in een systeem van Artena. Het importeren van data is goed te verduidelijken aan de hand van het volgende voorbeeld:

Eén van de klanten is het pensioenfonds van TNO, elke deelnemer (werknemer) die aangesloten is bij dit pensioenfonds kan via het deelnemersportal van TNO zijn/haar pensioengegevens inzien. Deze (deelnemer)gegevens worden periodiek aangeleverd door TNO in de vorm van een klantbestand (een Excel97 bestand).

Om deze gegevens in het systeem te verwerken moeten deze gegevens geïmporteerd worden. Deze importprocessen zijn meestal geïmplementeerd als SSIS packages. Figuur 4 toont een package waarmee een TNO deelnemer bestand geïmporteerd wordt in het systeem van



Figuur 4: TNO import package

Artena. Microsoft SQL Server Integration Services[4] is een platform waarmee data-integratie –en transformatieoplossingen gebouwd kunnen worden.

Tijdens de bewerkingen die in figuur 4 te zien zijn, kunnen de beschreven problemen (zie de probleembeschrijving, §3.1) zorgen voor fouten waardoor het importproces mislukt. Om de problemen zoals deze beschreven waren door stakeholder 2 verder te specificeren heb ik een user story gemaakt waarin de problemen aan de hand van een voorbeeld specifieker zijn beschreven. Dit document is bijgevoegd in bijlage C.

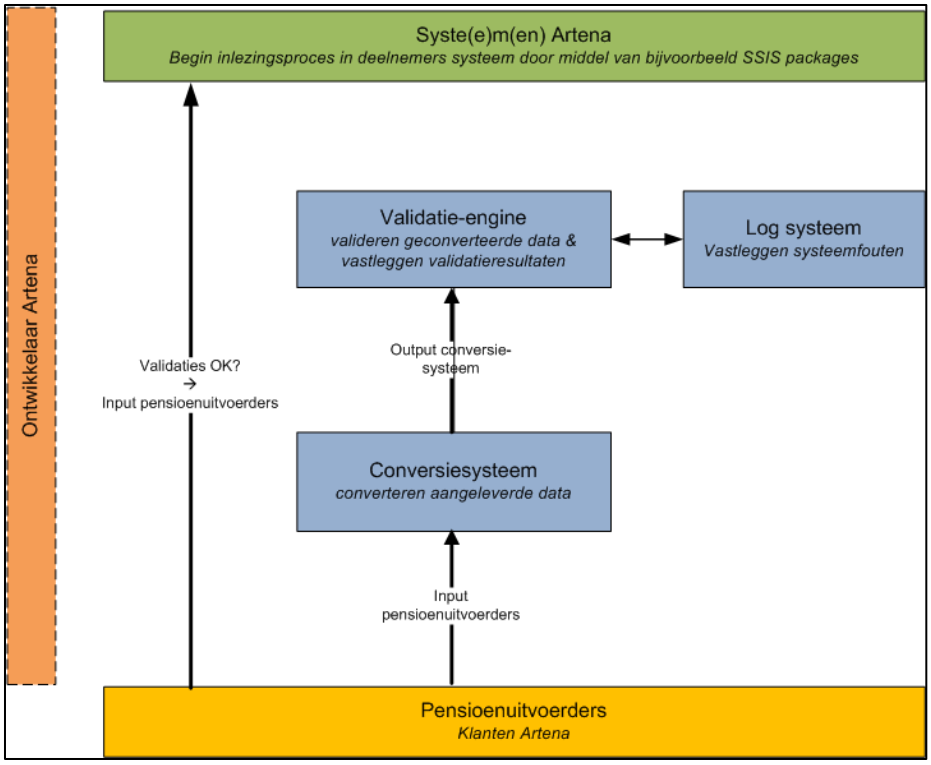
6.2 De Artena Data Validator

De drie deliverables; het conversiesysteem, de validatie-engine en het logsysteem maken elk onderdeel uit van één systeem, de Artena Data Validator. Elk van de drie deliverables vervult een andere rol binnen de Artena Data Validator.

Alvorens deze rollen gespecificeerd worden is het belangrijk te weten waar de applicatie binnen de andere systemen/applicaties van Artena thuishoort. Met deze vraag heb ik mijzelf tijdens de oriëntatie beziggehouden. Het resultaat daarvan is een document (zie bijlage D) waarin onder andere de 'locatie' van de Artena Data Validator beschreven is. De Artena Data Validator positioneert zich aan het begin van een importproces (of welk ander dataproces dan ook) om de data die geïmporteerd moet worden in een systeem van Artena te controleren. In figuur 5 wordt de plaatsing van de Artena Data Validator inclusief zijn subsystemen weergeven.

Eerst zullen de aangeleverde bestanden worden verwerkt door het conversiesysteem en vervolgens door de validatie-engine.

Indien de validaties geen foutieve resultaten opleveren dan kan het aangeleverde klantbestand worden verwerkt in het systeem van Artena.



Figuur 5: Plaatsing Artena Data Validator

[4] http://msdn.microsoft.com/en-us/library/ms141026.aspx

7. Iteratie 1: Conversiesysteem

In dit hoofdstuk beschrijf ik de ontwikkeling van het conversiesysteem, het eerste subsysteem van de Artena Data Validator. Het conversiesysteem bevat functionaliteiten waarmee de gegevens uit de aangeleverde bestanden uitgelezen kunnen worden en daarmee worden klaargezet voor verwerking door de validatie-engine (hoofdstuk 8)

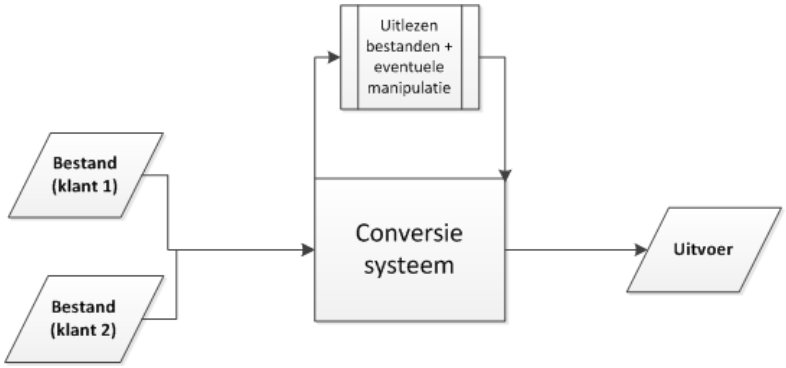
In §7.1 beschrijf ik mijn werkzaamheden uitgevoerd tijdens de definitiefase van deze iteratie, waaronder de definitie van de requirements. In §7.2 beschrijf ik de ontwerpfase waarin ik de database en programmatuur van het conversiesysteem heb vastgelegd. In §7.3 beschrijf ik tot slot kort de constructie van het conversiesysteem.

7.1 Definitiefase

7.1.1 Requirements

Aangezien nog geen specifieke functionele eisen waren vastgelegd voor het conversiesysteem in het afstudeerplan en plan van aanpak ben ik deze als eerste gaan vastleggen in een requirements document. De eisen zijn meerdere keren gewijzigd op basis van observaties, input van stakeholders en feedback van de begeleidende examinator.

Uit gesprekken met de stakeholders (vooral stakeholder 2) kwam naar voren dat de aangeleverde bestanden eerst naar één formaat 'geconverteerd' moest worden alvorens een validatieroutine op de bestandsregels in deze bestanden uitgevoerd zou worden. Op deze manier zou de invoer van de validatie-engine voor alle te ondersteunen bestandsformaten hetzelfde zijn, zie figuur 6.



Figuur 6: Invoer/uitvoer conversiesysteem

Deze verwerkingsstap wordt ook wel *staging*[5] genoemd, maar tijdens het gehele project is gesproken van een conversiesysteem omdat naast de omzetting van bestanden naar één formaat ook enkele manipulaties worden uitgevoerd op de data. Deze manipulaties hebben vooral betrekking op het formaat van de datums en decimale getallen van velden in Excel bestanden.

De analyse die is uitgevoerd voor het conversiesysteem door het definiëren van de requirements heeft ertoe geleid dat de gewenste functionaliteiten van het conversiesysteem duidelijk zijn geworden voor zowel mijzelf als de stakeholders. De gewenste flexibiliteit is daarbij afgebakend door het bepalen van een bestandsstructuur die het conversiesysteem verwacht wil het de bestanden kunnen converteren.

[5] http://en.wikipedia.org/wiki/Staging_(data)

Een ander belangrijk aspect dat naar voren kwam was dat de wijze waarop de verschillende bestanden uitgelezen zouden worden nog niet vaststond. Er waren meerdere mogelijkheden die elk hun voor –en nadelen hadden. Om de juiste wijze te kiezen heb ik een onderzoek moeten uitvoeren nadat de requirements waren vastgesteld. De resultaten hiervan zijn beschreven in §7.1.3.

Onderstaande tabel toont de laatste versie van de functionele requirements van het conversiesysteem, deze tabel is afkomstig uit de laatste versie van het analyserapport uit de eerste iteratie, zie bijlage E.

LEGENDA

Nr.	= nummer van de eis
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Eis	= de eis zelf

Nr.	St.	Pr.	Eis
F1	S1,S2	1	Invoer van conversiesysteem: • Eén of meerdere klantbestanden **{vervolg in F2, F3, F4}** • Eén of meerdere verwijzingen naar de templates waarin de benodigde gegevens voor een conversie van een klantbestand staan **{vervolg in F5}**
F2	S2	1	Soorten klantbestanden te ondersteunen door conversiesysteem: • Excel (xls of xlsx) • CSV Het aantal ondersteunde bestandsformaten is flexibel **{zie NF3}**
F3	S3	2	Het conversiesysteem moet de conversie van XML klantbestanden uit kunnen voeren wanneer deze genoeg gegevens bevatten en complex genoeg zijn zodat kan worden voldaan aan de uitvoer van het conversiesysteem **{vervolg in F6}.**

F4	S2	1	Structuur klantbestanden voor invoer conversiesysteem:
			• Een bestand bestaat uit één of meerdere tabellen
			  ○ Een tabel bestaat uit één of meerdere kopregels plus nul of meerdere rijen
			• Structuur rij in een tabel:
			  ○ (xls/xlsx) nul of meerdere velden
			  ○ (csv) nul of meerdere velden gescheiden door een scheidingsteken
			• Een veld heeft een datatype:
			  ○ Datum of getal of double of string

F5	S2	1	Gegevens benodigd in een template om conversie uit te voeren: • Aantal tabellen aanwezig in bestand (minimaal één) • Bestandstype (xls of xlsx of csv)

F6	S2	1	Uitvoer van een conversieproces (de import):
			• Database met daarin:
			  ○ Naam klantbestand
			  ○ Nul of meerdere rijen uit klantbestand **{vervolg in F7}**

F7	S2	1	Kenmerken van een rij in uitvoer van een conversie: • Nummer van de tabel • Nummer van de rij • Nul of meerdere velden gescheiden door een komma (volgens de RFC-4180 standaard[6])
F8	S2	1	Formaten van uitvoervelden van xls/xlsx conversies moeten voldoen aan onderstaande definities, zodat dit gelijk zal zijn:

[6] http://tools.ietf.org/html/rfc4180

			1. Datum – 'dd-mm-yyyy hh:mm:ss' 2. Integer – 'xxxxxxx'(geen duizendtal separatoren) 3. Double – 'xx.xx' (geen duizendtal separatoren & gescheiden door punt i.p.v. komma)
F9	S2, S3	2	Het conversiesysteem moet in staat zijn conversies voor bestanden van verschillende klanten parallel aan elkaar uit te voeren.
F10	S2	1	Indien zich fouten voordoen tijdens een conversieproces (=één run van een conversie) moet het conversiesysteem de gegevens/kenmerken van deze fout kunnen vastleggen/loggen.
F11	S2	2	Het conversiesysteem moet in staat zijn het bestandsformaat van het klantbestand automatisch te bepalen aan de hand van de inhoud van de meegeleverde template (of extensie) wanneer een klantbestand is geselecteerd.
F12	S2	3	Het opstartmechanisme van het conversiesysteem moet gebaseerd zijn op een systeem dat een nieuw aangeleverd klantbestand detecteert.

In de tabel hieronder staan een aantal niet-functionele eisen waarbij gefocust is op een aantal kwaliteitskenmerken afkomstig van ISO-9120. De door mij gebruikte categorieën van dit model zijn *functionality compliance*(1) en *changeability*(2):

Nr.	St.	Pr.	Cat.	Eis
NF1	S2	1	1	De uitvoer van een conversie voldoet zoals in **{F7}** beschreven aan de RFC-4180[7][8] standaard voor Comma Separated Values. Dit houdt in dat: • Individuele velden in een rij zijn omgeven door dubbele quotes (''item'') • Individuele velden zijn van elkaar gescheiden door een komma (,) • Het laatste veld wordt niet afgesloten met een komma
NF2	S2	1	1	Het conversiesysteem moet de ASCII standaard[9] volledig ondersteunen. Dit houdt in dat het systeem de tekens uit deze karakterset kan herkennen en verwerken.
NF3	S2	1	3	De flexibiliteit van het conversiesysteem moet zodanig zijn dat een ondersteuning van een nieuw bestandsformaat geen wijzigingen aan de code van het conversieproces te weeg moet brengen, de wijze waarop het conversieproces omgaat met verschillende soorten bestanden zal dus generiek moeten zijn.

Bovenstaande requirements zijn geprioriteerd met behulp van de MoSCoW methode. In de vierde kolom staat een getal (1-4) waaraan de prioriteit kan worden afgeleid:

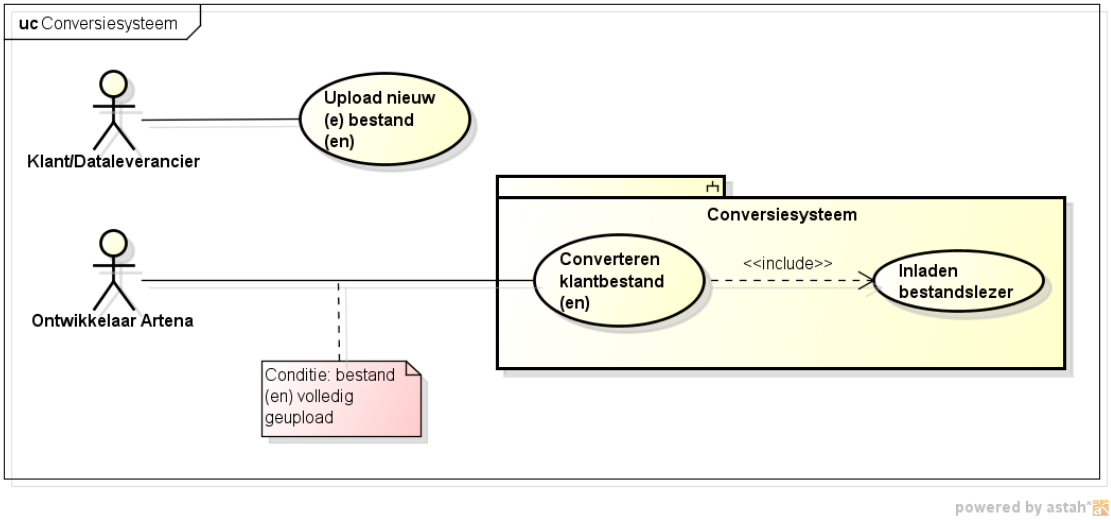
1. **Must have:** deze eisen worden in ieder geval binnen de gemaakte timebox/planning verwerkt.
2. **Should have**: Deze eisen hoeven verplicht binnen de planning verwerkt te worden.
3. **Could have:** een dergelijke eis kan worden meegenomen als tijd en kosten dat toelaten.
4. **Won't have**: deze eisen zullen niet worden verwerkt in het te implementeren systeem.

De beschreven prioriteiten van de requirements zijn tot stand gekomen door met de stakeholder(s) één voor één door de lijst met eisen heen te lopen en te vragen hoe belangrijk de eis was. Uit dit soort vragen kwam bijvoorbeeld naar voren dat de implementatie van XML ondersteuning niet zo heel belangrijk was in dit stadium, ondersteuning voor Excel en CSV bestanden waren belangrijker.

[7] http://www.testconsultancy-groep.nl/index.php/downloads - zie ISO-9126 toegelicht
[8] http://tools.ietf.org/html/rfc4180
[9] http://ascii-table.com/ansi-codes.php

7.1.2 Use cases

Na de opstelling van de requirements zijn de functionele eisen uitgewerkt in een UML use case diagram en daarbij behorende use case beschrijvingen. Met behulp van de opgestelde use cases werden delen van het proces waarin de eisen verwerkt zouden duidelijk. Figuur 10 toont de laatste versie van het use case diagram voor het conversiesysteem. Dit diagram is afkomstig uit het analyserapport (bijlage E).



Figuur 7: Use case diagram conversiesysteem

De use case beschrijving hieronder is van toepassing op de use case 'converteren klantbestand(en)'. Deze use case is van toepassing op de functionele eisen F1-F8 (eisen met betrekking tot de invoer en uitvoer van het systeem) en F10 (eis met betrekking tot de systeemfouten).

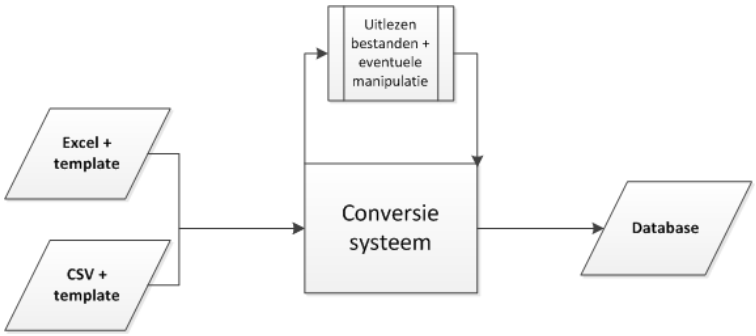
Naam	Converteren klantbestand(en)
Aannamen	De ontwikkelaar van Artena weet de locatie van het/de klantbestand(en) en de template(s) die daarbij hoort(horen)
Beschrijving	(1) De ontwikkelaar selecteert de juiste bestand(en) en de juiste template(s) (2) De ontwikkelaar start de conversie m.b.v. een button in een interface **[I]** (3) Het conversiesysteem selecteert de juiste bestandslezer voor het bestand - **Use case 'Inladen bestandslezer'** (4) De bestandslezer leest de regels van het bestand in en voert afhankelijk van het bestandsformaat de noodzakelijke manipulaties uit. (5) Het conversiesysteem schrijft de geconverteerde bestandsregels weg naar de database is het gewenste formaat **[II]**
Uitzondering	**[I]** Het systeem herkent de ingevoerde template niet, waardoor het conversieproces wordt afgebroken en een systeemfout wordt getoond aan de ontwikkelaar **[II]** Databasefout (*kan geen verbinding maken met database*), waardoor het conversieproces wordt afgebroken en een systeemfout wordt getoond aan de ontwikkelaar
Resultaat	De data aanwezig in het geselecteerde bestand is geconverteerd naar de betreffende database.

Met behulp van deze –en andere use case beschrijvingen was het mogelijk om tijdens de analyse een globaal conversieproces te identificeren.

7.1.3 Onderzoek 'bestandsconversie'

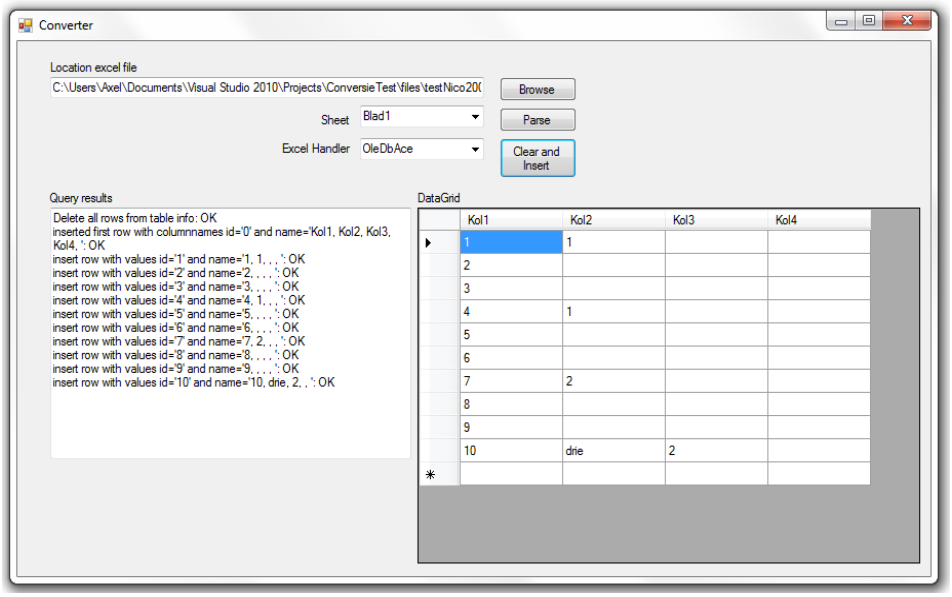
In figuur 6 is het idee achter het conversiesysteem weergegeven. Na het vastleggen van de requirements kunnen we de invoer en uitvoer van het conversiesysteem anders weergeven, zie figuur 8.

Alvorens een ontwerp uitgedacht kon worden om het conversiesysteem te implementeren moest worden uitgezocht op welke manier de in de requirements beschreven invoer omgezet kon worden naar de beschreven uitvoer. Cruciaal hierbij is de eerste stap in het 'conversieproces', namelijk het uitlezen van de aangeleverde bestanden. Voor het uitlezen van deze bestanden (vooral de Excel bestanden) bleken veel mogelijkheden te zijn, die elk hun voor –en nadelen hadden.



Figuur 8: Invoer/uitvoer conversiesysteem - herzien

Om uit te zoeken wat de beste oplossing was voor het uitlezen van de Excel en CSV bestanden heb ik een onderzoek uitgevoerd naar de verschillende mogelijkheden, het volledige onderzoek inclusief resultaten en conclusie is te vinden in bijlage F.



Figuur 9: Gebruikersinterface testapplicatie

Voor dit onderzoek had ik een testapplicatie geschreven waarin een bestand geselecteerd en uitgelezen kon worden. De uitgelezen gegevens konden vervolgens worden weergegeven in de applicatie (figuur 9).

Excel bestanden

De twee belangrijkste eisen voor het uitlezen/converteren van Excel bestanden zijn:

1. De conversie van Excel bestanden moet alle velden ongeacht hun datatype converteren.
2. Het formaat van datums en (decimale) getallen moet hetzelfde zijn ongeacht het bronformaat (Excel 97-2003 of Excel 2007)

Uiteindelijk zijn in het onderzoek vier methoden voor uitlezen van Excel bestanden onderzocht en getest:

1. **OLEDB Jet Engine 4.0**
2. **OLEDB ACE Engine 12.0**
3. **Office Interop Excel Object**
4. **Excel Data Reader for .NET**

Methoden 1,2 en 3 zijn ontwikkeld door Microsoft en zijn te gebruiken in het *.NET framework* van Microsoft. Methode 4 is een open-source bibliotheek ontwikkeld in C# als alternatief voor de oplossingen van Microsoft.

De door Microsoft ontwikkelde methoden om Excel bestanden werkten naar behoren op een paar serieuze nadelen na. De eerste drie methoden zijn vooral afgekeurd op onderstaande punten:

- Geen ondersteuning voor Excel 2007 bestanden (methode 1)
- Datatypering van een gehele kolom geschiedde op basis van de analyse van de eerste 8 rijen in die kolom, waardoor waarden die niet voldeden aan deze datatypering niet werden meegenomen. (methode 1 en 2)
- Slechte performance (methode 3)

Methode 4 bleek een serieuze kandidaat te zijn voor het uitlezen van de Excel bestanden omdat deze methode alle velden uit een Excel bestand als één datatype behandelde: *System.String*. Hierdoor werden alle waarden ongeacht hun datatype uitgelezen. De open-source bibliotheek van methode 4 bleek wel meerdere bugs te bevatten, maar die konden opgelost worden door het gebruik van een aantal beschikbaar gestelde patches.

Naast de vier beschreven methoden is ook nog gekeken naar het gebruik van SQL Server Integration Services(zie §6.1) voor het uitlezen van de bestanden, echter bleek SSIS ook gebruik te maken van de methoden 1 en 2 om dergelijke bestanden uit te lezen. Daarom werd SSIS ook niet als bruikbaar beschouwd.

Conclusie van onderzoek: Voor het uitlezen van bestanden zal de Excel Data Reader for .NET worden gebruikt als de uit te voeren unit-tests (waarmee de functionaliteit van de bestandslezer getest worden) een positief resultaat hebben. Deze unit-tests(zie Bijlage G) toonden geen problemen aan bij de toepassing van de open-source bibliotheek.

Figuur 10: Excel Data Reader for .NET

Comma Separated Values bestanden

Naast Excel bestanden moesten ook CSV bestanden uitgelezen kunnen worden. Omdat het duidelijk was dat er een C# implementatie geschreven zou worden voor het uitlezen van Excel bestanden zou het een goede ontwerpkeuze zijn om het uitlezen van CSV bestanden door dezelfde applicatie af te handelen. Er ontstaat dan één conversiesysteem als onderdeel van de Artena Data Validator. Het gebruik van SSIS voor CSV bestanden was daarmee dan ook uitgesloten.

Voor het onderzoek heb ik een eigen implementatie geschreven om een CSV bestand uit te lezen. Van belang hierbij was dat het CSV formaat kon verschillen. In de implementatie heb ik het uitlezen twee verschillende soorten CSV bestanden moeten verwerken:

1. CSV volgens RFC-4180: *“aaa”,”bbb”,”ccc”*
2. CSV van een klant van Artena: *aaa;bbb;ccc*

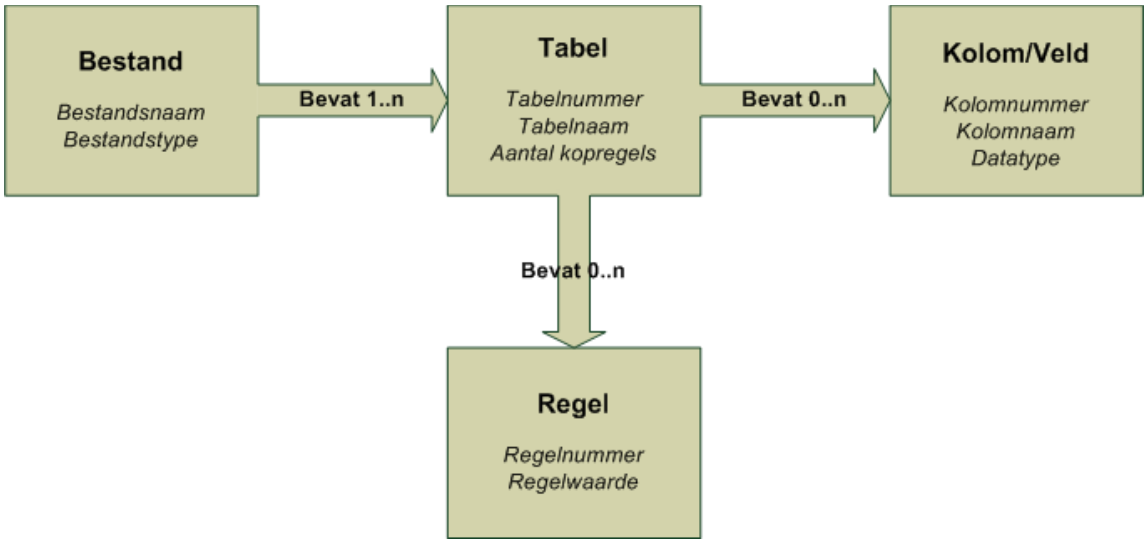
Het bleek relatief makkelijk om met het .NET framework een dergelijk uitleessysteem te implementeren in de testapplicatie. De unit-test die aan het einde is uitgevoerd (bijlage H) toonde aan dat alle waarden correct werden uitgelezen.

7.2 Ontwerpfase

In deze fase beschrijf ik de manier waarop ik vanuit de vastgestelde eisen en de onderzoeksresultaten het ontwerp van het conversiesysteem heb opgesteld. Het volledige rapport van het conversiesysteem is bijgevoegd in bijlage I. Belangrijk bij het opstellen van het ontwerp is dat ik de datastructuur als basis heb genomen voor het systeem om er vervolgens een O/R mapper op toe te passen (zie ook §5.1.3).

7.2.1 Templates

Zoals beschreven in functionele eis **F1** heeft het conversiesysteem naast de aangeleverde klantbestanden ook templates nodig om de bestanden uit te lezen. Een template beschrijft de verwachte structuur van een bestand. Voordat ik de templatestructuur kon ontwerpen heb ik vastgesteld welke bestandstructuur ik verwacht in een aangeleverd bestand (figuur 11).



Figuur 11: Bestandstructuur

Zoals beschreven in de requirements (**F6 en F7**) bestaat de uitvoer van het conversiesysteem (= de inhoud van het klantbestand) uit een ‘database’ met daarin een naam van het klantbestand en de bestandsregels. Elke bestandsregel bestaat uit een regelnummer en een verwijzing naar de tabel en het bestand waar hij bij hoort.

Met behulp van de structuur van figuur 11 en de requirements met betrekking tot de uitvoer heb ik kunnen vaststellen hoe die uitvoer in relatie staat met zijn template om eventuele verdere verwerking (bijvoorbeeld een validatie) te ondersteunen.

1. Het aangeleverde klantbestand moet gekoppeld zijn aan een bestand template
2. De uitgelezen bestandsregels moeten gekoppeld zijn tabel template (die weer gekoppeld is aan een bestand template)

De regels hierboven beschreven regels zijn toegepast op het databaseontwerp beschreven in §7.2.2.

7.2.2 Databaseontwerp

Met behulp van de gegevens uit §7.2.1 was het mogelijk om een datastructuur op te stellen waarin de templates en de uitvoer van een conversie samen konden worden vastgelegd in het gebruikte DBMS (SQL Server 2008). Voor het databaseontwerp heb ik de methode gebruikt zoals beschreven in §5.2.2:

1. Opstellen (Enhanced) Entity Relationship diagram
2. Opstellen Relationeel Representatie model
3. Opstellen Relationeel Implementatie model

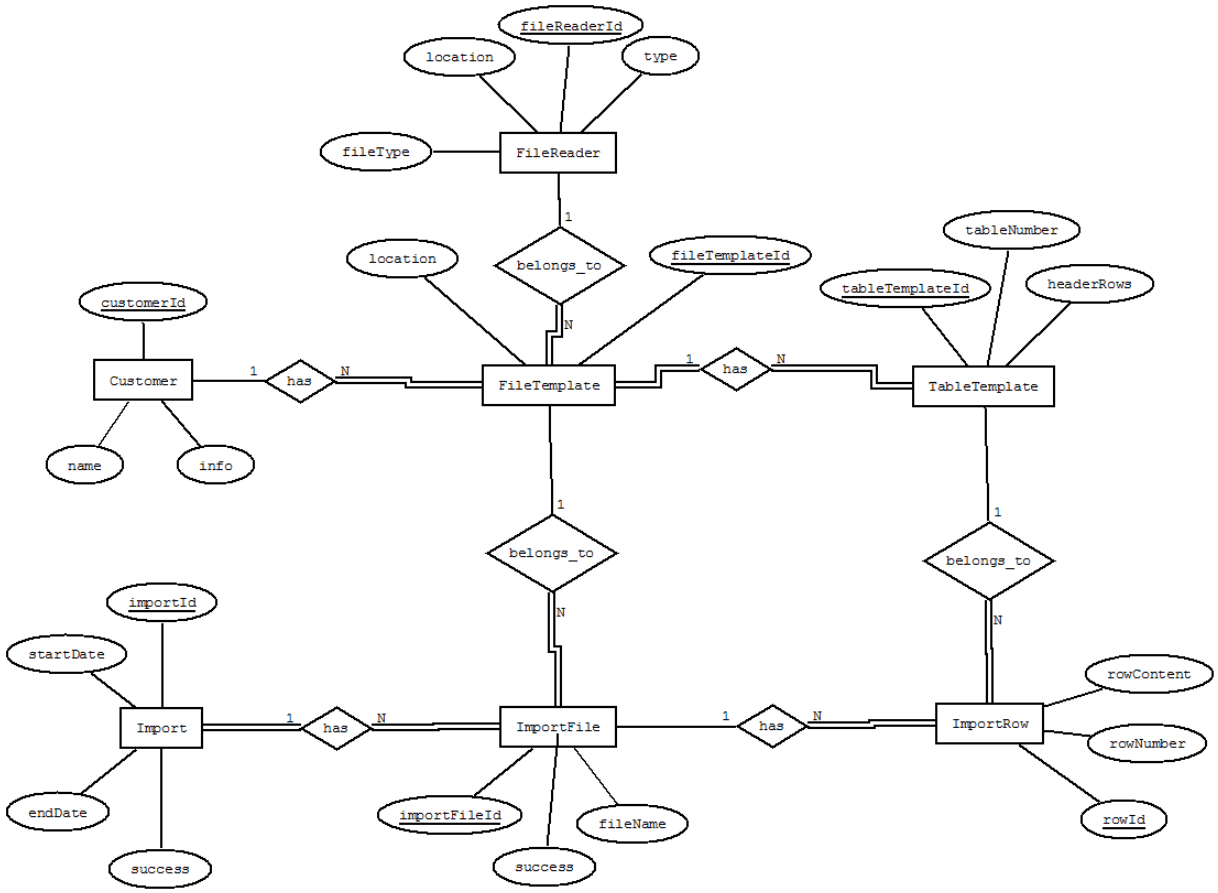
EER diagram

Figuur 12 toont de het EER diagram van het conversiesysteem. Het conversiesysteem kan vanuit deze database templates ophalen en kan geconverteerde bestandsregels wegschrijven naar de database.

Een template bestaat uit één FileTemplate en **één** of meerdere TableTemplates. De uitvoer van het conversiesysteem is gescheiden in Imports, ImportFiles en ImportRows, zodat één import uit één of meerdere klantbestanden kan bestaan en één importbestand uit meerdere importregels.

Het is duidelijk te zien dat twee entiteittypen (*FileTemplate*) en (*ImportRow*) afhankelijk zijn van twee of meer andere entiteittypen. Volgens de gebruikte methode (Literatuur: [Tré, G.] (2007)) kunnen deze entiteittypen daarom worden gedefinieerd als zwak. Dit is in het EER diagram niet gedaan omdat:

1. Artena heeft als eis voor het ontwerp van een database dat er altijd surrogaatsleutels worden gebruikt als identificerende attributen van een entiteit. Geen enkele tabel zal dus samengestelde primaire sleutels bevatten, zoals dat bij zwakke entiteittypen wel het geval is.
2. Een *ImportRow* kan alleen worden toegevoegd aan de database wanneer de bijbehorende TableTemplate en ImportFile ook aanwezig zijn in de database. Er zijn meerdere ImportRows met dezelfde TableTemplate en ImportFile mogelijk.
3. De verplichte relatie vanuit een *FileTemplate* met de *TableTemplate* is aangegeven in het ERD door de totale participatie. In praktijk is deze verplichte relatie geïmplementeerd door bij elke FileTemplate **minimaal één** TableTemplate toe te voegen.



Figuur 12: ERD conversiesysteem

In deze datastructuur zijn de functionele eisen m.b.t. invoer en uitvoer verwerkt, zoals functionele eis F7 die beschrijft dat een uitvoerrij het tabelnummer, rijnummer en velden moet bevatten. Dit komt overeen met de tabel ImportRow. De inhoud van een bestandsregel (de velden in een rij) wordt altijd weggeschreven in standaard CSV formaat (niet-functionele eis **NF1**) in de kolom RowContent:

- " waarde1","waarde2","waardeN"

Relationeel Representatie Model

Vanuit het ERD is een Relationeel Representatie Model opgesteld, hieronder een deel daarvan.

FileTemplate(FileTemplateId, FileLocation, *CustomerId*, *FileReaderId*)
CustomerId is foreign key, references CustomerId in Customer, NOT NULL
FileReaderId is foreign key, references FileReaderId in FileReader, NOT NULL

TableTemplate(TableTemplateId, TableNumber, IgnoreRows, *FileTemplateId)*
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NOT NULL

RIM & Triggers

Bij het opstellen van het RIM kwam ik erachter dat de implementatie van regels voor referentiële

integriteit (ON UPDATE/ON DELETE) niet op de 'normale' manier gedaan kon worden omdat het DBMS geen *multiple cascade paths*[10] ondersteunt.

Bij bijvoorbeeld de verwijdering van een FileTemplate worden zowel de TableTemplate(**TT**) als de ImportFile(**IF**) die daarmee een relatie heeft verwijderd. Bij zowel de TT als de IF zou dan een ON DELETE CASCADE worden aangeroepen zodat de ImportRows behorende bij de TT en de IF verwijderd zullen worden. In deze situatie kan een ImportRow dus via twee *cascade paths* verwijderd worden en dat wordt niet ondersteund door het SQL Server DBMS zodat bijvoorbeeld *deadlocks* worden voorkomen.

Dit probleem komt volgens de ontwikkelaars van de stakeholder 2 en stakeholder 3 vaker voor in de systemen van Artena en ontstaat bij de wat complexere databases. Om deze integriteit toch te waarborgen moest gebruikt gemaakt worden van Triggers om de referentiële integriteit in de relaties te waarborgen. Hieronder de trigger voor een FileTemplate:

```
-- =============================================
-- Author:          Axel Polet
-- Create date: 23-04-2012
-- Description:     This trigger is used to enforce referential integrity (ON DELETE
CASCADE) between FileTemplate and TableTemplate & between FileTemplate & ImportFile
-- =============================================
ALTER TRIGGER [dbo].[Delete_File_Template]
   ON  [ConversionImport].[dbo].[FileTemplate]
   INSTEAD OF DELETE
AS
BEGIN

       IF @@ROWCOUNT = 0 RETURN

       -- Delete importFile entries linked to the fileTemplate
       entry(enforces CASCADE)
       DELETE FROM [ConversionImport].[dbo].[ImportFile]
       FROM [ConversionImport].[dbo].[importFile] AS imf
       INNER JOIN deleted d ON imf.FileTemplateId = d.FileTemplateId

       -- Delete tableTemplate entries linked to the
       fileTemplate (enforces CASCADE 1)
       DELETE FROM [ConversionImport].[dbo].[TableTemplate]
       FROM [ConversionImport].[dbo].[TableTemplate] AS tt
       INNER JOIN deleted d ON tt.FileTemplateId =
       d.FileTemplateId

       -- Delete the fileTemplate entry itself
       DELETE FROM [ConversionImport].[dbo].[FileTemplate]
       WHERE FileTemplateId IN (SELECT d.FileTemplateId FROM deleted d)
END
```

In deze trigger worden eerst de, aan de FileTemplate gerelateerde, ImportFiles (klantbestanden) verwijderd alvorens de bijbehorende TableTemplates (tabellen) worden verwijderd waardoor referentiële integriteit gehandhaafd is. De overige triggers zijn bijgevoegd als bijlage aan het ontwerprapport van de eerste iteratie (zie Bijlage I).

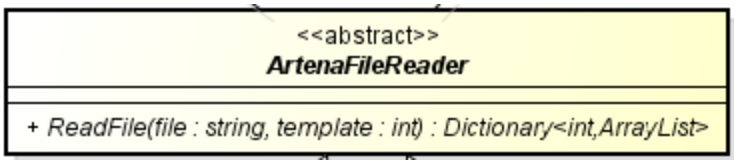
[10] http://www.christian-etter.de/?tag=multiple-cascade-paths

7.2.3 Plug-in systeem voor bestandslezers

In §7.2.2 is een FileReader tabel opgenomen in het databaseontwerp. In deze paragraaf wordt uitgelegd hoe deze tabel bijdraagt aan vervulling van andere functionele eisen.

Niet-functionele eis NF3 beschrijft een eis aan de uitbreidbaarheid van het conversiesysteem. Deze eis heeft betrekking op de uitbreidbaarheid van de ondersteunde bestandsformaten voor het conversiesysteem. De eisen F2 en F3 beschrijven de ondersteuning voor twee bestandsformaten. Binnen de scope van het project moest ik alleen ondersteuning voor Excel en CSV bestanden implementeren in het conversiesysteem omdat dit de formaten zijn waarin klanten momenteel hun gegevens aanleveren. Het zou zomaar kunnen dat in de toekomst een nieuwe klant zijn bestanden aanlevert in Microsoft Access of XML formaat.

Om deze mogelijke wijzigingen op te vangen heb ik besloten een plug-in systeem te ontwerpen voor het conversiesysteem. Bij dat systeem is elke plug-in verantwoordelijk voor het uitlezen van een bestandsformaat. Voor dit project heb ik dus twee plug-ins geïmplementeerd (één voor Excel bestanden en één voor CSV bestanden). Het conversiesysteem laadt de juiste plug-in op basis van de gegevens die vastgelegd zijn in de *FileReader* tabel in de database.



Figuur 13: abstracte klasse bestandslezer plug-in

Om ervoor te zorgen dat elke plug-in door het conversiesysteem op dezelfde manier wordt behandeld (waardoor dus de uitbreidbaarheid gegarandeerd is) wordt gebruik gemaakt van een abstracte klasse. Deze klasse representeert en definieert het plug-in object, zie figuur 13.

Het enige wat het conversiesysteem hoeft te doen om bestanden uit te lezen is de methode *ReadFile()* aan te roepen op elke plug-in die ingeladen is en de juiste verwijzingen (bestand en template) mee te geven. Het *object* wat het conversiesysteem ontvangt van de plug-in bevat de ingelezen bestandsregels, die door de plug-in zijn vastgelegd in het CSV formaat (niet-functionele eis NF1).

Het plug-in systeem is in C# geïmplementeerd, door gebruik te maken van van Reflection[11-12]. Reflection '*is het proces waarbij een applicatie zichzelf kan observeren en zijn eigen structuur en gedrag runtime (dynamisch) kan veranderen*'.

Dit houdt dit dat het mogelijk is om op basis van bepaalde informatie een bepaald type object (de plug-in) in te laden om de gewenste functionaliteiten (die dan nodig zijn) uit te voeren. Deze plug-in zal geïmplementeerd moeten zijn als een DLL. De gegevens die Reflection nodig heeft om een plug-in in te laden zijn:

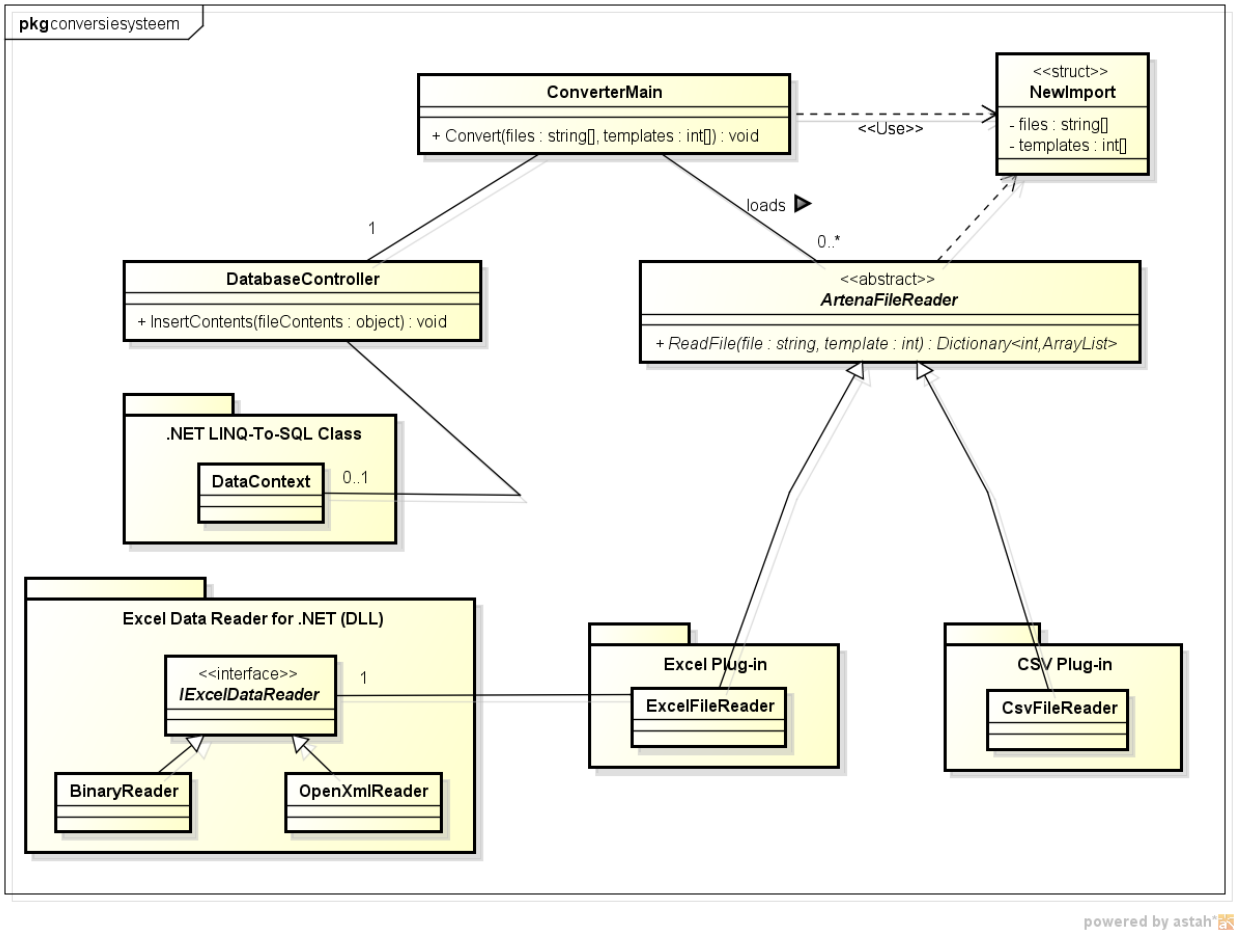
- Locatie van het DLL bestand (bijv. C:\Software\ExcelFileReader.dll)
- *Volledige type* van het in te laden object (bijv. Artena.FileReaders.ExcelFileReader)

[11] http://en.wikipedia.org/wiki/Reflection_(computer_programming)
[12] http://www.codeguru.com/csharp/csharp/cs_misc/reflection/article.php/c4257

7.2.4 Domeinmodel

Voor het conversiesysteem was dus een soort conversieroutine nodig waarin op basis van de in de database vastgelegde gegevens (templates) de juiste bestandslezers worden ingeladen. Elke bestandslezer leest de aangeleverde bestanden daarbij op zijn eigen manier uit en retourneert de bestandsregels in het juiste formaat.

In § 7.2.5 is beschreven op welke manier de entiteiten uit het datamodel van §7.2.2 worden gerepresenteerd in de programmatuur van het conversiesysteem (en de Artena Data Validator). Deze klassen zijn niet meegenomen in het domeinmodel. Figuur 14 toont het domeinmodel (uit het ontwerprapport) van het conversiesysteem, waarin het conversieproces verwerkt zit.



Figuur 14: domeinmodel conversiesysteem

In dit model is de conversieroutine in de ConverterMain klasse verwerkt (in de methode **Convert()**), deze klasse controleert dan ook het gehele proces. De relatie met een bestandslezer plug-in is ook aangegeven. Ook is de aanwezigheid(Excel Data Reader for .NET) van de gekozen oplossing voor het uitlezen van Excel bestanden aangegeven. Zoals te zien is, wordt elke plug-in als een los package beschreven, omdat elke plug-in als een aparte *Dynamic Linked Library* wordt geïmplementeerd(zie §7.2.3). De volledige beschrijving van de klassen is aanwezig in het ontwerprapport (bijlage I).

7.2.5 LINQ-to-SQL

Om de template en import entiteiten uit de database te representeren kan worden gekozen voor een implementatie van deze objecten als aparte klassen in het conversiesysteem (en de Artena Data Validator). Dit kost echter veel tijd en wanneer er iets wijzigt aan de database dan moet dat hoogstwaarschijnlijk ook in de betreffende klasse worden gedaan.

Microsoft .NET biedt hiervoor een mooie oplossing, in de vorm van een LINQ-to-SQL implementatie. Dit is een *object-relational mapping* waarbij de entiteiten/tabellen van een relationele database omgezet worden in objecten en als objecten benaderbaar zijn vanuit de programmatuur. Om *transacties* uit te voeren op de betreffende database waarvan de LINQ-to-SQL implementatie is gemaakt, wordt gebruik gemaakt van een *DataContext* klasse.

In figuur 15 staat een deel van de definitie van klasse TableTemplate, zoals deze gegenereerd is door LINQ-to-SQL. Duidelijk te zien zijn de EntitySet en EntityRef variabelen die de relatie tussen TableTemplate-ImportRow en TableTemplate-FileTemplate waarborgen.

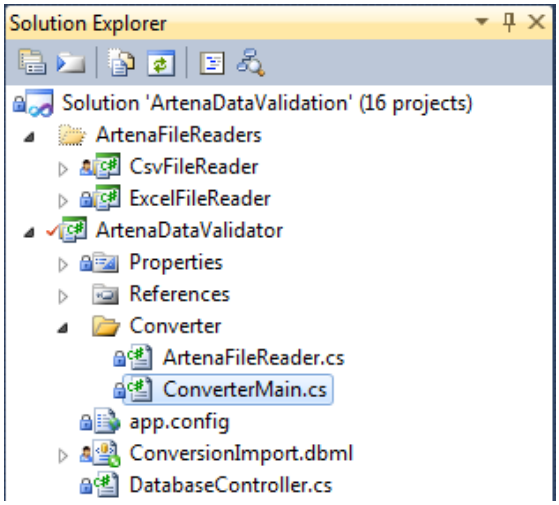
```
public partial class TableTemplate
{
        private int _TableTemplateId;
        private int _TableNumber;
        private System.Nullable<int> _HeaderRows;
        private int _FileTemplateId;

        private EntitySet<ImportRow> _ImportRows
        private EntityRef<FileTemplate> _FileTemplate;
}
```

Figuur 15: TableTemplate klasse gegenereerd door LINQ-to-SQL

7.3 Constructiefase

In de implementatie(fase) ben ik bezig geweest met het implementeren van het ontwerpdocument dat op dat moment was opgesteld. De implementatie van het ontwerp is uitgevoerd in C# met behulp van Visual Studio 2010. Figuur 16 toont de *solution* waarin het conversiesysteem zich bevindt.

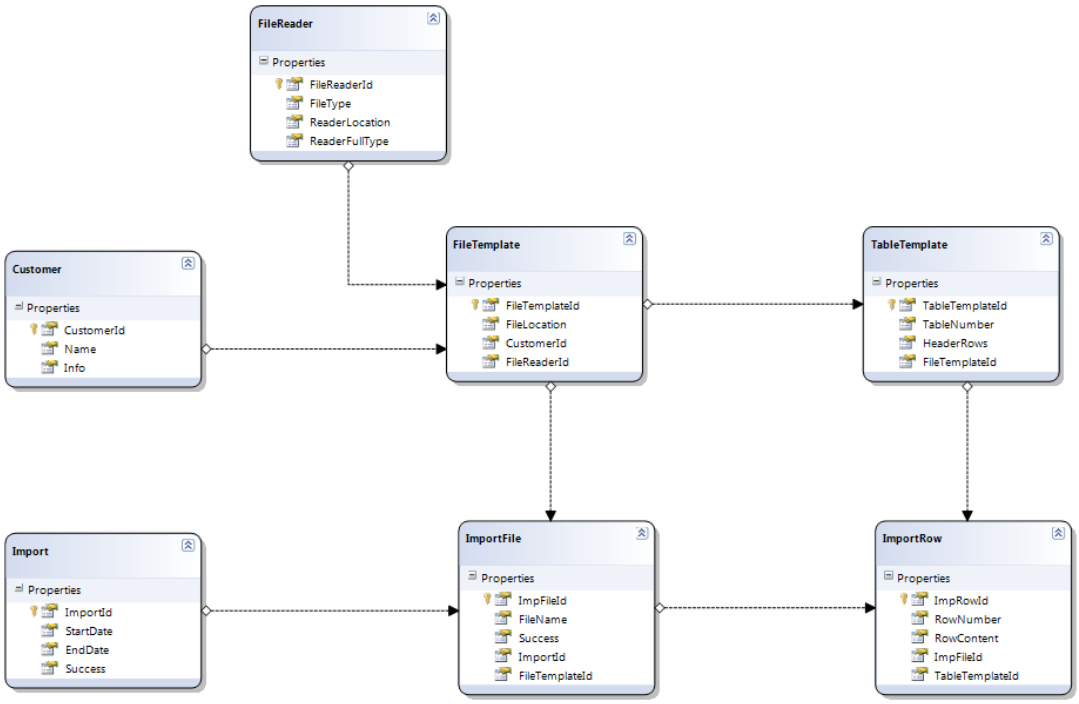
De icoontjes voor de verschillende bestanden tonen de status van het bestand (gewijzigd, up-to-date) zoals deze wordt gebruikt door de Visual Studio Team Edition versiebeheer. Het is duidelijk te zien dat de bestandslezers zich bevinden in aparte projecten om ze te implementeren als losse DLL's (zie ook §7.2.3).



Figuur 16: Solution

7.3.1 LINQ-to-SQL in Visual Studio

Zoals eerder vermeld is gebruik gemaakt van LINQ-to-SQL voor om de template –en importentiteiten als objecten in de programmatuur te kunnen gebruiken. De representatie van de gehele database voor het conversiesysteem is in figuur 17 weergeven en komt overeen met het ERD van het ontwerp.



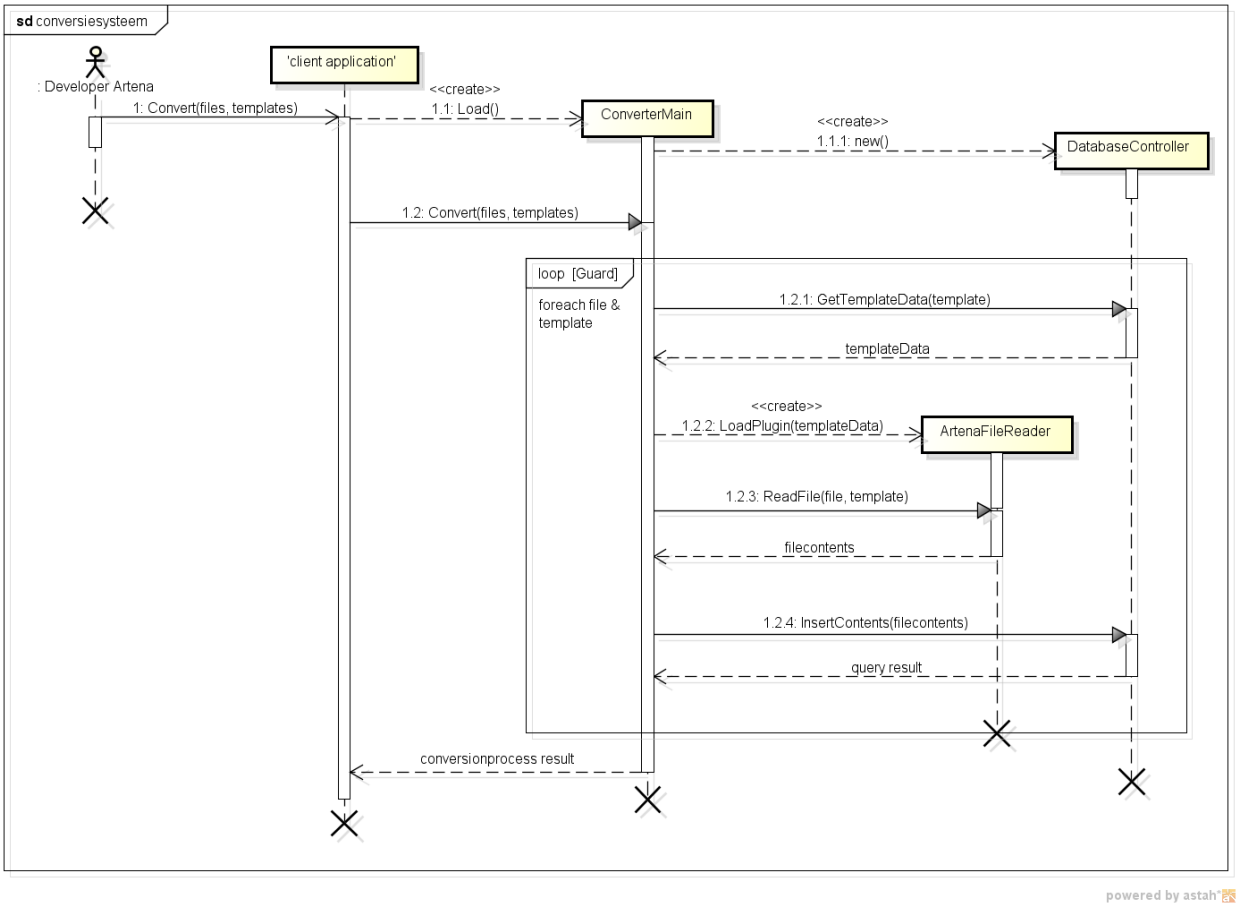
Figuur 17: LINQ-to-SQL classes

7.3.2 Conversieproces

Het volledige conversieproces heb ik vastgelegd in een UML Sequence diagram. Het diagram laat zien welke handelingen worden uitgevoerd wanneer de **Convert()** methode(zie ook het klassendiagram) wordt aangeroepen.

Het starten van een conversie wordt gedaan vanuit een client application door de ontwikkelaar van Artena. Deze client kan een willekeurig systeem betreffen, zoals een systeem dat reageert op een nieuwe upload (*file check system*) of een web-interface.

Ik heb zelf een client ontwikkeld met behulp van windows forms waarbij vanuit een eenvoudig venster de te converteren bestanden en daarvoor bestemde templates ingevoerd kunnen worden. Een eerste versie van die (client)applicatie is getoond in figuur 9.



Figuur 18: Sequence diagram conversiesysteem (uit bijlage I)

Zoals te zien is in het diagram wordt voor elk bestand/template een serie handelingen herhaald. Deze handelingen zorgen ervoor dat de inhoud van het aangeleverde bestand op de juiste wijze in database terecht komt. Beschrijving van de handelingen:

1. Developer Artena roept methode **Convert()** aan via de client
   1.1. De client applicatie maakt een nieuwe instantie aan van de ConverterMain
   1.2. Aan de nieuwe instantie wordt gevraagd om een conversie uit te voeren
      1.2.1. Converter haalt benodigde data op uit de database voor elk bestand/template
      1.2.2. Converter laadt de benodigde plug-in voor elk bestand/template
      1.2.3. Converter roept de methode **ReadFile()** van elke plug-in aan en geeft het bestand + template mee.
      1.2.4. Converter voegt de uitgelezen bestandsregels verkregen uit de bestandslezer toe aan de juiste databasetabellen

De meeste klassen beschreven in het klassendiagram zijn terug te vinden in het sequence diagram: ConverterMain, ArtenaFileReader en DatabaseController.

8. Iteratie 2: Validatie-engine

In dit hoofdstuk wordt de tweede iteratie in de ontwikkeling van de Artena Data Validator besproken, namelijk de validatie-engine. In §8.1 wordt de uitgevoerde definitiefase beschreven inclusief de nieuwe requirements. In §8.2 worden de belangrijkste databaseontwerpen en programmatuurontwerpen voor de validatie-engine beschreven. Tot slot beschrijf ik in §8.3 het door mij geïmplementeerde validatieproces door een UML sequence diagram te tonen.

Het conversiesysteem van hoofdstuk 7 heeft veel invloed op de ontwikkeling van de validatie-engine. In dit hoofdstuk zijn verwijzingen naar dat systeem ook niet uitgesloten. Zo is de database zoals deze wordt beschreven in §8.2 een uitbreiding op de database gebruikt in het conversiesysteem en is (zoals beschreven in §8.1.1) de uitvoer van het conversiesysteem een deel van de invoer van de validatie-engine.

8.1 Definitiefase

In deze fase zijn de requirements voor de validatie-engine vastgelegd. Buiten de, in het plan van aanpak en de user story, gedefinieerde problemen waren nog geen specifieke eisen vastgelegd met betrekking tot bijvoorbeeld de verschillende soorten validaties die uitgevoerd zouden kunnen worden. Een ander belangrijk onderdeel is het vaststellen van mogelijke volgorde(n) van bepaalde validatieroutines. In deze fase zijn ook use cases opgesteld en is een (klein) achtergrondonderzoek uitgevoerd naar datakwaliteit. Het volledige rapport is bijgevoegd in bijlage J.

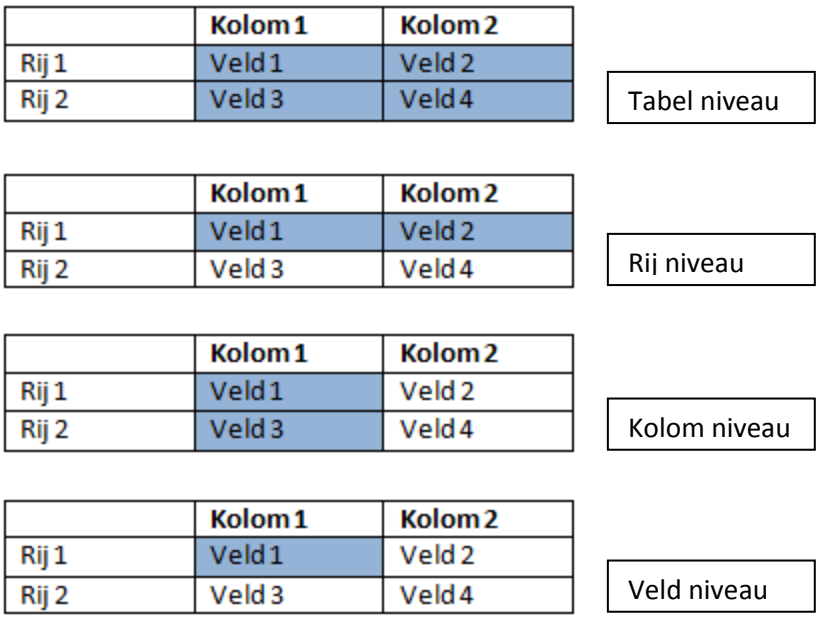
8.1.1 Requirements

De eerste uitgevoerde stap in de definitiefase is het vaststellen van de requirements van de validatie-engine. Door de definitie van deze requirements (in overleg met de stakeholders) zijn de wensen van verschillende stakeholders duidelijk geworden, maar is ook de structuur en volgorde van validaties duidelijk geworden.

Hiermee bedoel ik vooral dat naast de verschillende typen validatieregels ook een validatieflow is bedacht waarin de verschillende soorten regels uitgevoerd worden. Voordat de analyse was uitgevoerd was er bijvoorbeeld nog geen sprake van de term validatieniveau. Door de definitie van verschillende validatieniveaus is het systeem van validatieregels beter beheersbaar geworden door verschillende validaties te groeperen zijn onder een niveau. Per niveau kan vervolgens de afhandeling op de juiste wijze geïmplementeerd worden. Uiteindelijk is door het toekennen van de gewenste soorten validaties aan deze niveaus is een duidelijk beeld ontstaan van de functionaliteiten van de validatie-engine.

De vier verschillende validatieniveaus die toegepast worden in de validatie-engine zijn: tabel niveau, kolom niveau, rij niveau en veld niveau.

In figuur 19 staan vier tabellen waarin beschreven is op welke data-eenheden in een bestand verschillende niveaus van toepassing zijn. Dit figuur is afkomstig uit het ontwerprapport van de tweede iteratie (bijlage K).



Figuur 19: Beschrijving validatieniveaus

De requirements van de validatie-engine zijn dus weer tot stand gekomen door gesprekken met de stakeholders over de gewenste functionaliteiten voor een validatiesysteem. De requirements zijn wederom geprioriteerd door de mate van belangrijkheid van de eisen bij de stakeholders te toetsen.

De functionele eisenlijst hieronder is onderdeel van de volledige lijst aan eisen uit het analyserapport en bevat alleen requirements met betrekking tot de validatie-engine. In de requirements is onder andere de toekenning van verschillende validatieregels aan validatieniveaus terug te zien. In het Bijlage J staan ook een aantal niet-functionele eisen beschreven.

LEGENDA

Nr. = nummer van de eis
Systeem = subsysteem of deliverable waar de eis op van toepassing is
St. = stakeholder van wie de eis afkomstig is
Pr. = prioriteit van de eis (MoSCoW)
Eis = de eis zelf

Nr.	Systeem	St.	Pr.	Eis
F13	Validatie	S2, S3	1	De invoer voor een uit te voeren validatie door de validatie engine bestaat uit onderstaande onderdelen: • Een deel van de uitvoer van een conversie (de import): ○ Nul of meerdere rijen met de structuur van **{F7}** • Een set met validatieregels op verschillende niveaus en van verschillende typen **{vervolg in F14, F15, F16 en F17}**
F14	Validatie	S2	1	Benodigde gegevens (naast condities) voor **table-level** (tabel niveau) validatie: • Tabelnummer Typen validaties op dit niveau: • Rows Range (aantal regels) ○ Condities: aantal regels in tabel ## is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van

				####’
F15	Validatie	S2	1	Benodigde gegevens (naast condities) voor **row-level** validatie: • Tabelnummer • Rijnummer Typen validaties op dit niveau: • Field Count (aantal velden in rij) o Conditie: de rij heeft evenveel velden als er kolommen zijn vastgelegd in de template
F16	Validatie	S2, S4	1	Benodigde gegevens (naast condities) voor **column-level** validatie: • Tabelnummer • Kolomnummer Typen validaties op dit niveau: • Fields Range (aantal ingevulde velden): o Conditie: het aantal velden in kolom ## dat ingevuld is, is ‘groter dan ####’ of ‘kleiner dan ####’ of ‘van #### tot ####’ of ‘percentage van ####’ • Uniqueness o Conditie: de waarde in kolom ## mag maar één keer voorkomen
F17	Validatie	S2, S3, S4	1	Benodigde gegevens (naast condities) voor **field-level** validatie: • Tabelnummer • Rijnummer • Kolomnummer Typen validaties op dit niveau: • Datatype o Conditie: waarde in veld is van het datatype ‘datum’ of ‘double’ of ‘integer’ of ‘string’ • Presence o Conditie: waarde in veld mag niet NULL zijn • Range o Conditie: waarde in veld is ‘groter dan ####’ of ‘kleiner dan ####’ of ‘van #### tot ####’ of ‘percentage van ####’ • Format o Conditie: waarde in moet van een bepaald formaat o Mogelijke formaten: ▪ Datum – ‘dd-mm-yyyy hh:mm:ss’ ▪ Double – ‘xx.xx’ (gescheiden door punt i.p.v. komma) ▪ Integer – ‘xxxxxxx’ (geen duizendtal separatoren) • Lookup o Conditie: waarde in veld moet voldoen aan een waarde in een ‘database—tabel—veld’ • Relationship o Conditie: als ‘waarde in veld 1 gelijk is aan #### dan waarde is waarde in veld 2 ingevuld’
F18	Validatie	S1,S2	1	Het systeem moet de mogelijkheid bevatten om *quality control validation* uit te voeren. Dit type validatie slaat op de wijze waarop een ‘fout’ resultaat

				wordt geïnterpreteerd. In dit geval geeft de fout een indicatie van de waarschijnlijkheid van de gevonden waarde. **{zie ook F23}**
F19	Validatie	S2	1	Indien een table-level validatie niet succesvol is, dat wil zeggen fout, dan zullen de row-level, column-level –en field-level validaties voor die tabel niet meer worden uitgevoerd.
F20	Validatie	S2	1	Indien een row-level –of column-level validatie niet succesvol is, dat wil zeggen fout, dan zullen field-level validaties voor die rij of kolom niet meer worden uitgevoerd.
F21	Validatie	S2	1	Het moet mogelijk zijn om naast de in **{F14, F15, F16 en F17}** beschreven validatieregels nieuwe soorten/typen validatieregels op elk gewenst niveau toe te voegen wanneer dat gewenst is **{zie ook NF5}**.
F22	Validatie	S2	1	De validatie-engine moet in staat zijn validaties van verschillende klantbestanden parallel aan elkaar uit te voeren.
F23	Validatie	S2	1	Een validatieresultaat wordt alleen vastgelegd als de validatie een fout genereert. (en het gevalideerde dus niet voldoet aan de condities)
F24	Validatie	S2	2	Een validatieresultaat kan van twee niveaus zijn: • Error: er is niet voldaan aan de validatie condities • Warning: er is niet voldaan aan de validatie condities maar het gecontroleerde gegeven is niet perse fout (waarschijnlijkheid)

F25	Validatie	S1,S2	1	Gegevens van een fout resultaat van een validatie moeten voldoen aan onderstaande criteria:
				• De validatieregel (type en niveau) en condities waar niet aan is voldaan
				• De reden waarom er niet is voldaan aan de condities
				• Het tabelnummer waarop de validatie betrekking had
				  ○ Table-level
				  ○ Row-level
				  ○ Column-level
				  ○ Field-level
				• Het rijnummer waarop de validatie betrekking had
				  ○ Row-level
				  ○ Field-level
				• Het kolomnummer waarop de validatie betrekking had
				  ○ Column-level
				  ○ Field-level
				• Het resultaat moet in zijn geheel leesbaar zijn voor de gebruikers (de klanten)

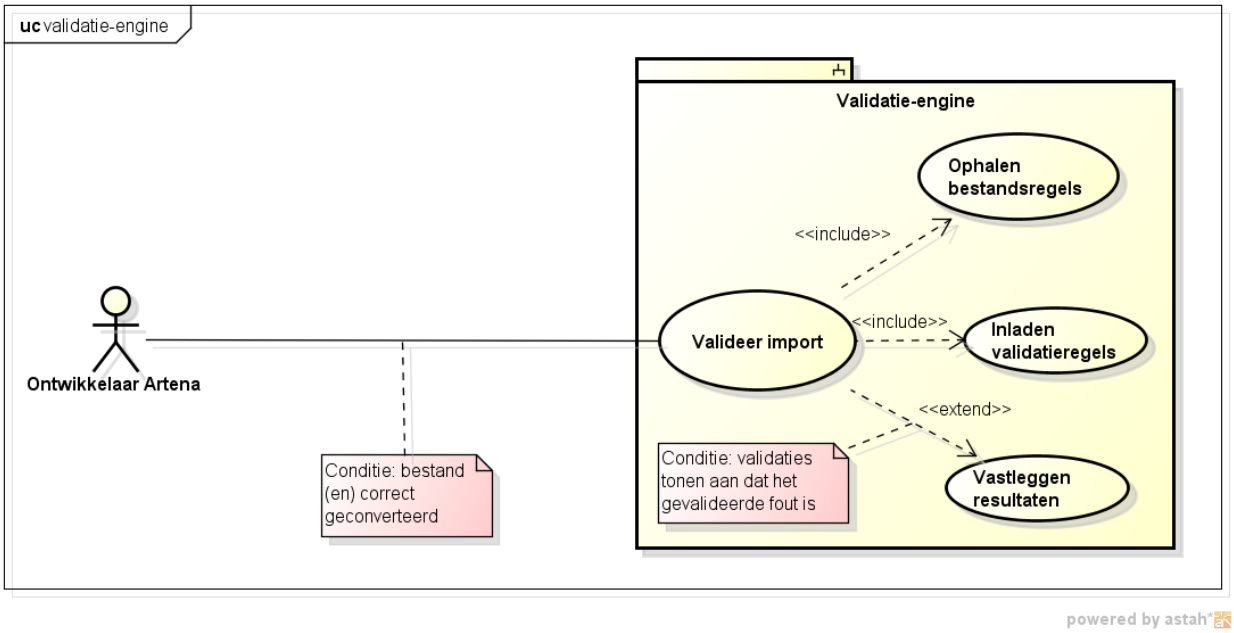
Naast de gewenste functionaliteiten is ook bij de validatie-engine een wens met betrekking tot de flexibiliteit uitgesproken, die weer valt onder de categorie *Changeability* van ISO-9126:

NF5	Validatie	S2	1	3	De flexibiliteit van de validatie-engine moet zodanig zijn dat het toevoegen/wijzigen van de validatieregels en validatietypen geen grote impact heeft op het proces dat doorlopen wordt in de validatie-engine bij de uitvoer van een validatie

8.1.2 Use cases

Om de functionele eisen te verduidelijken voor zowel mijzelf als de stakeholders heb ik wederom een aantal use cases in een use case diagram(figuur 19) beschreven. Het diagram verduidelijkt de interacties

die de ontwikkelaar van Artena met het systeem moet hebben en de interacties die het systeem intern heeft.



Figuur 20: Use case diagram - validatie-engine

Wat vooral duidelijk wordt uit het diagram is dat ik tijdens de definitiefase al nagedacht had over de verschillende processen die zich tijdens een validatieroutine zouden afspelen. Deze processen zijn vermeld in de drie use cases aan de rechterkant van het diagram. Ik wist bijvoorbeeld dat de uitvoer van het conversiesysteem een deel van de invoer van de validatie-engine zou zijn, waardoor de toevoeging van de use case 'ophalen bestandsregels' verklaard kan worden.

Ook had ik bij de definitie van de requirements al vastgesteld dat er een bepaalde volgorde zat in de uitvoer van verschillende regels door de validatieniveaus één voor één af te handelen. Dit is ook beschreven in onderstaande use case beschrijving.

Naam	Valideer import (functionele eisen **F13 – F25)**
Aannamen	De import is correct weggeschreven door het conversiesysteem en de import(verwijzing) is bekend bij de ontwikkelaar van Artena
Beschrijving	(1) De ontwikkelaar selecteert de te valideren import en start de validatie m.b.v. een button in een interface (2) **Use case 'ophalen bestandsregels'** (3) **Use case 'inladen validatieregels'** (4) De validatie-engine start de uit te voeren validaties op table-level (5) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (6) **Use case 'inladen validatieregels'** (7) De validatie-engine start de uit te voeren validaties op row-level **[I]** (8) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (9) **Use case 'inladen validatieregels'** (10) De validatie-engine start de uit te voeren validaties op column-level **[I]** (11) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (12) **Use case 'inladen validatieregels'**

	(13) De validatie-engine start de uit te voeren validaties op field-level **[I] [II]** (14) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]**
Uitzondering	**[I]** Validatie op table-level is niet succesvol, waardoor de validatie-engine de validatieregels op de andere validatieniveaus niet zal uitvoeren en het validatieproces zal afronden. **[II]** Validaties uitgevoerd op bijbehorend column –of row-level zijn niet succesvol, waardoor de validatie-engine de validatieregels op het betreffende veld niet uit zal voeren.
Resultaat	De bestandsregels uit van de import zijn gevalideerd en eventuele resultaten van de validaties zijn vastgelegd in de database.

De andere use cases, die via de *include* en *extends* relaties met de bovenstaande use case zijn verbonden, zijn beschreven in het analyserapport iteratie 2(bijlage J).

8.1.3 Onderzoek datakwaliteit & vergadering

Vanuit de vierde stakeholder kwam na de gedane analyse een vraag over de mogelijke toepassing van de validatie-engine in een data quality framework. Sinds enige tijd was stakeholder 4 bezig met de ontwikkeling hiervan en volgens deze stakeholder waren er wel mogelijkheden tot inpassing van de validatie-engine (of de Artena Data Validator in zijn geheel) in dat framework.

Vanuit het oogpunt van stakeholder 4 is een theoretische onderbouwing van de validatie-engine gewenst. Dit hield in dat ik een onderzoek moest uitvoeren waarbij ik heb gekeken naar de toepassing van de validatie-engine op het gebied van datakwaliteit. Om dit te bewerkstelligen heb ik twee datakwaliteit modellen onderzocht en de kwaliteitsaspecten uit deze modellen toegepast op de gewenste functionaliteiten uit de requirements.

Vanuit de twee modellen (IDQ[13] en PSP/IQ[14]) heb ik vastgelegd op welke aspecten de Artena Data Validator toepassing heeft, dit zijn: Integriteit, Volledigheid, Nauwkeurigheid, Plausibiliteit, Syntax, Structuur, Consistentie en Eenduidigheid. De tabel hiernaast toont de overeenkomsten tussen deze aspecten en het de validatieregels.

Een volledige beschrijving inclusief samenvatting van de gebruikte modellen is aanwezig in het analyserapport (bijlage J).

Aspect	Validatieregel(s)	
Integriteit	Datatype	(field-level)
	Presence	(field-level)
	Range	(field-level)
	Lookup	(field-level)
Volledigheid	Rows Range	(table-level)
	Row Count	(row-level)
	Fields Range	(column-level)
Nauwkeurigheid	Datatype	(field-level)
	Range	(field-level)
Syntax	Datatype	(field-level)
	Format	(field-level)
	Lookup	(field-level)
Structuur	Uniqueness	(column-level)
	Relationship	(field-level)
Consistentie	Format	(field-level)
	Lookup	(field-level)
Eenduidigheid	Format	(field-level)
	Lookup	(field-level)

De verkregen resultaten uit het onderzoek zijn gepresenteerd in een vergadering met stakeholder 4 en stakeholder 2. In deze vergadering kwam vooral naar voren dat de vraag naar datakwaliteit momenteel heel hoog is (vooral vanuit financiële instellingen) en dat de Artena Data Validator mogelijk een deel van deze vraag zou kunnen bevredigen door geïmplementeerd te worden in het framework. Wanneer dat

[13] http://www.smartest.nl/verdieping/kwaliteitsmodellen/idqinformatiekwaliteit
[14] http://web.mit.edu/tdqm/www/tdqmpub/KahnStrongWangCACMApr02.pdf

zou gebeuren en op welke manier is niet bekend geworden. In de derde iteratie is een tweede vergadering gehouden waarin ik een demonstratie heb gegeven van de Artena Data Validator zelf.

8.2 Ontwerpfase

In deze fase heb ik het ontwerp van de validatie-engine opgesteld. Het volledige ontwerp is terug te vinden in het ontwerprapport iteratie 2(bijlage K). De belangrijkste onderdelen van dit ontwerp zijn de validatieflow(§8.2.2), uitbreiding van de database(§8.2.3) en de implementatie van de benodigde flexibiliteit in de programmatuur(§8.2.5).

Daarnaast bevat §8.2.1 de definitie van een validatieregel –en validatieresultaat en bevat §8.2.4 een beschrijving van de wijze waarop condities worden opgeslagen in een validatieregel.

8.2.1 Validatieregels & Validatieresultaten

Een validatieregel bevat alle benodigde gegevens om een validatie uit te voeren op een bepaald validatieniveau. Een validatieregel is van een bepaald validatieniveau, bevat een verwijzing naar de eerder gedefinieerde template(zie §8.2.3) en bevat een aantal condities (parameters) die de regel specificeren.

Een validatieresultaat wordt vastgelegd wanneer in de validatieroutine (zie §8.2.2) het resultaat van een validatie aangeeft dat het gevalideerde fout is. Het gevalideerde is fout als het niet voldoet aan de condities van de validatieregel. Een validatieresultaat heeft afhankelijk van het niveau waarop hij vastgelegd is verschillende gegevens nodig:

Validatieniveau	Benodigde gegevens om correct resultaat op te stellen
Tabel	Validatieregel – Tabel(nummer) – Bestand
Rij	Validatieregel – Tabel– Bestandsregel
Kolom	Validatieregel – Kolom(nummer) – Bestand of Bestandsregel
Veld	Validatieregel – Kolom – Bestandsregel

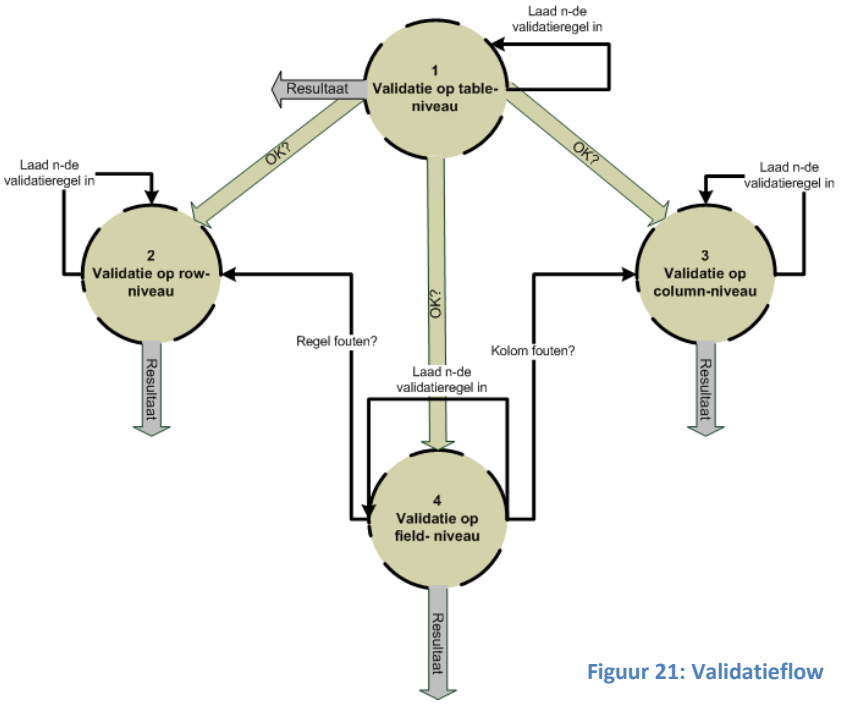
In het ontwerprapport (bijlage K) heb ik alle te ondersteunen validatieregels en (mogelijke) validatieresultaten uiteengezet. Hierin zijn de mogelijke condities per validatietype (op alle niveaus) beschreven en ook een foutbeschrijving die mogelijk weergeven zou kunnen worden aan de gebruiker van de validatie-engine.

8.2.2 Validatieflow

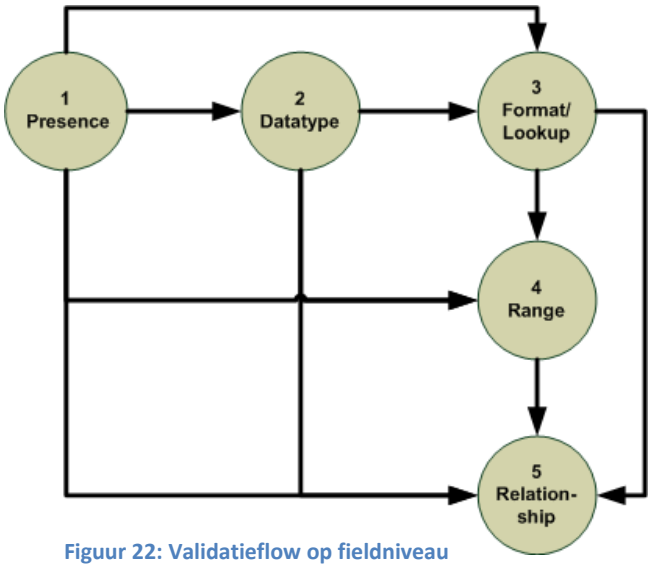
In het use case diagram dat is opgesteld tijdens de definitiefase wordt rekening gehouden met een bepaalde volgorde in de uitvoering van de verschillende validatieregels, zie ook §8.1.2. Tijdens de ontwerpfase heb ik deze volgorde verduidelijkt zodat deze begrijpelijker is geworden voor alle stakeholders. Hiertoe heb ik een ‘flowchart’ gemaakt in Microsoft Visio 2010, zie figuur 21.

In het diagram is duidelijk een bepaalde volgorde te zien in de uitvoer van validatieregels op de te ondersteunen validatieniveaus en eventuele uitzonderingen waardoor de uitvoer van validatieregels op bepaalde niveaus geblokkeerd wordt.

1. Uitvoeren van validatieregels op de inhoud een gehele tabel van een bestand.
   Wanneer een tabel validatie een fout oplevert dan worden de validatieregels op de lagere validatieniveaus (row, column en field) niet meer uitgevoerd.
2. Uitvoeren van validatieregels op rijniveau (rij voor rij doorlopen)
3. Uitvoeren van validatieregels op kolomniveau (kolom voor kolom doorlopen)
4. Uitvoeren van validatieregels op veldniveau (waarbij veld voor veld doorlopen zal worden).
   Validaties op een veld zullen niet worden uitgevoerd wanneer een rij validatie of kolom validatie overeenkomend met dat veld al een fout heeft vastgesteld.



Figuur 21: Validatieflow



Figuur 22: Validatieflow op fieldniveau

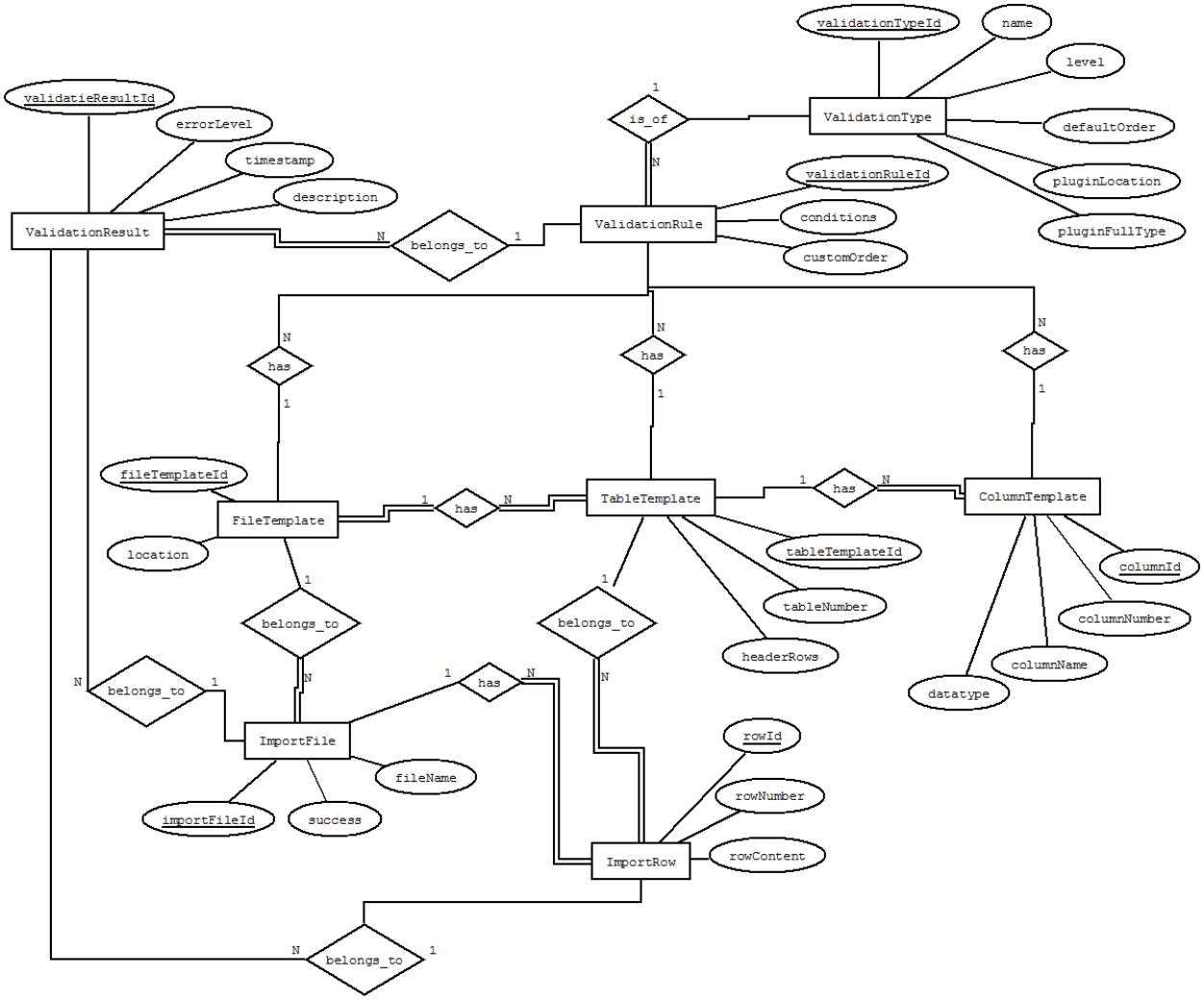
Naast de volgorde gebaseerd op de verschillende validatieniveaus hebben de validatieregels binnen één validatieniveau ook een standaard volgorde waarin ze uitgevoerd worden, omdat de resultaten van sommige validatieregels andere validatieregels uitsluiten. Voor de volledige **standaard** volgorde van de validatieregels op veldniveau, zie de getallen (1-5) in figuur 22.

8.2.3 Databaseontwerp

Nadat ik de definities van de validatieregels en validatieresultaten had opgesteld kon ik het databaseontwerp voor de validatie-engine opstellen. De eerste belangrijke stap was de **uitbreiding** van de database van het conversiesysteem met functionaliteiten voor de validatie-engine. Deze keuze is logisch omdat een groot deel van de gedefinieerde invoer van de validatie-engine (zie de functionele eis F13) bestaat uit de uitvoer van het conversiesysteem. Daarnaast was ook duidelijk dat voor de uitvoer en resultaatvastlegging van de validaties op de gedefinieerde validatieniveaus ook de bestandstemplate (*FileTemplate)* en tabel template (*TableTemplate)* structuur van het conversiesysteem gebruikt moesten

worden. Tot slot wist ik dat beide (sub)systemen in de derde iteratie samengevoegd zouden worden tot één systeem, namelijk de Artena Data Validator.

Als basis voor het databaseontwerp is dus het EER diagram van het conversiesysteem gebruikt (§7.2.2). Op dit EER diagram zijn eerst een aantal uitbreidingen gedaan voor de implementatie van validatieregels en vervolgens is een uitbreiding gedaan om eventuele resultaten van de validaties vast te leggen. Het diagram van figuur 23 toont het uiteindelijke ERD afkomstig van het ontwerprapport (bijlage K) waarin de validatieregels en validatieresultaten vastgelegd kunnen worden.



Figuur 23: ERD - validatie-engine

Gebruikte tabellen van het conversiesysteem

FileTemplate & TableTemplate (de al bestaande template structuur)
ImportFile & ImportRow (al bestaande structuur waarin de conversie-uitvoer wordt vastgelegd)

Uitbreidingen m.b.t. validatieregels

Het toevoegen van een entiteittype **ColumnTemplate**, met een relatie naar het TableTemplate

entiteittype. De ColumnTemplate is bedoeld om een kolom in een tabel van een bestand te beschrijven door middel van kolomnaam, kolomnummer en datatype.

Het toevoegen van een entiteittype **ValidationType**, waardoor verschillende typen(bijvoorbeeld datatype of presence) validatieregels bijgehouden kunnen worden. In deze entiteit wordt het validatieniveau waar een validatietype bij hoort vastgelegd en wordt vastgelegd wat de standaard volgorde is van de validatieregel (getal van 1-n). Twee andere attributen (**PluginLocation** en **PluginFullType**) van deze entiteit wordt gebruikt door een plug-in systeem (zie §8.2.5).

Het toevoegen van een entiteittype **ValidationRule**, waarin een specifieke validatieregel van een bepaald ValidationType wordt vastgelegd. Deze regel heeft een relatie met een bepaalde template (FileTemplate, TableTemplate of ColumnTemplate) afhankelijk van het validatieniveau (zie §8.2.1). De kolom **Conditions** bevat de specifieke parameters voor een validatieregel (zie §8.2.4).

Uitbreidingen m.b.t. validatieresultaten

Het toevoegen van een entiteittype **ValidationResult**, waarin een resultaat van een validatie (als deze een fout heeft geretourneerd) wordt vastgelegd. Het resultaat heeft een relatie met de ValidationRule en daardoor ook indirect met de template (zie het ERD). Daarnaast heeft een validatieresultaat een relatie met het Importfile of ImportRow waarop het resultaat van toepassing is (dit is dus het foute bestand of de foute regel). Een resultaat bevat ook een **TimeStamp** en een **ErrorLevel**. Door de implementatie van een error level is het mogelijk om resultaten vast te leggen die 'misschien fout' zijn, waardoor voldaan wordt aan de eis voor ondersteuning van quality control validaties (zie eis **F18**).

RRM, RIM & Triggers

Vanuit het ERD is een uitbreiding gedaan op het bestaande Relationele Representatie Model, Relationele Implementatie Model en Triggers van het conversiesysteem. Deze uitbreidingen zijn volledig beschreven in het ontwerprapport (Bijlage K).

8.2.4 Condities

De condities (of parameters) van een validatieregel worden vastgelegd in één kolom in de tabel ValidationRule. Om meerdere parameters te ondersteunen is er gekozen voor de vastlegging van deze parameters in CSV formaat: "parameter1","parameter2" in één kolom van de tabel ValidationRule.

Deze condities kunnen in dit (antirelationele)formaat worden vastgelegd omdat de afhandeling van de validatieregels wordt gedaan door een plug-in systeem (zie §8.2.5). Elke plug-in weet precies welke parameters hij kan verwachten wanneer het een validatieregel afhandelt en weet ook de wijze waarop het de condities kan splitsen.

Een andere mogelijkheid voor de ondersteuning van (meerdere condities) had de toevoeging van een entiteittype **Conditie** kunnen zijn (met een relatie naar de ValidationRule). Hier heb ik niet voor gekozen zodat het databaseontwerp niet te complex zou worden.

Deze condities verschillen dus van elkaar per type validatieregel, hieronder staan enkele voorbeelden van condities gebruikt voor een range validatieregel op veldniveau:

Conditie (mogelijk)	Beschrijving	Notatie “operator”,”limit1”,”limit2”
1	De waarde in het veld moet groter dan 2000 zijn	“>”,”2000”
2	De waarde in het veld moet groter dan of gelijk aan 2000 zijn	“>=”,”2000”
3	De waarde in het veld moet kleiner dan 2000 zijn	“<”,”2000”
4	De waarde in het veld moet kleiner dan of gelijk aan 2000 zijn	“<=”,”2000”
5	De waarde in het veld moet tussen 1000 en 2000 liggen	“<>”,”1000”,”2000”

Dit type validatieregel controleert de waarde in het veld op basis van door mijzelf gedefinieerde parameters. Wanneer deze mogelijke parameters uitgebreid of gewijzigd moeten worden dan kunnen deze wijzigingen worden opgevangen in de plug-ins. In de systeemdocumentatie (zie §9.5/ Bijlage P) zijn alle mogelijke condities voor alle ondersteunde validatie typen beschreven.

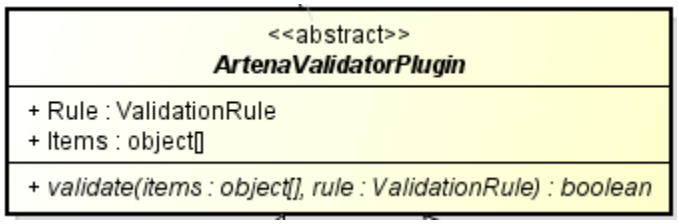
8.2.5 Flexibiliteit & Plug-in systeem

Om aan de eis met betrekking tot de flexibiliteit van de validatie-engine te voldoen (zie eisen F21 & NF5, Bijlage J) heb ik besloten om voor de afhandeling van verschillende validatieregels ook een plug-in systeem te implementeren. In dit systeem wordt elk soort/type validatieregel afgehandeld door een aparte plug-in. Dit systeem heeft veel overeenkomsten met het plug-in systeem van de bestandslezers dat gebruikt wordt in het conversiesysteem.

Het voordeel van het gebruik van een plug-in systeem voor de validatieregels is dat de afhandeling van verschillende regels van elkaar losgekoppeld is en onderhoud aan de verschillende validatieregels geen invloed zal hebben op andere validatieregels of de validatieroutine zelf. Tevens is ook de uitbreidbaarheid van het systeem vergemakkelijkt doordat het toevoegen van een nieuw type validatieregel gedaan kan worden door een nieuwe plug-in te implementeren.

Het plug-in systeem is ook hier geïmplementeerd met Reflection(zie ook §7.2.3).

Bij dit systeem was het ook belangrijk dat elke plug-in voldeed aan een aantal voorwaarden en op dezelfde manier afgehandeld zou kunnen worden door de validatie-engine. Om dit te garanderen is voor de validatie plug-ins ook een abstracte klasse geïmplementeerd: *ArtenaValidatorPlugin*, zie figuur 24.



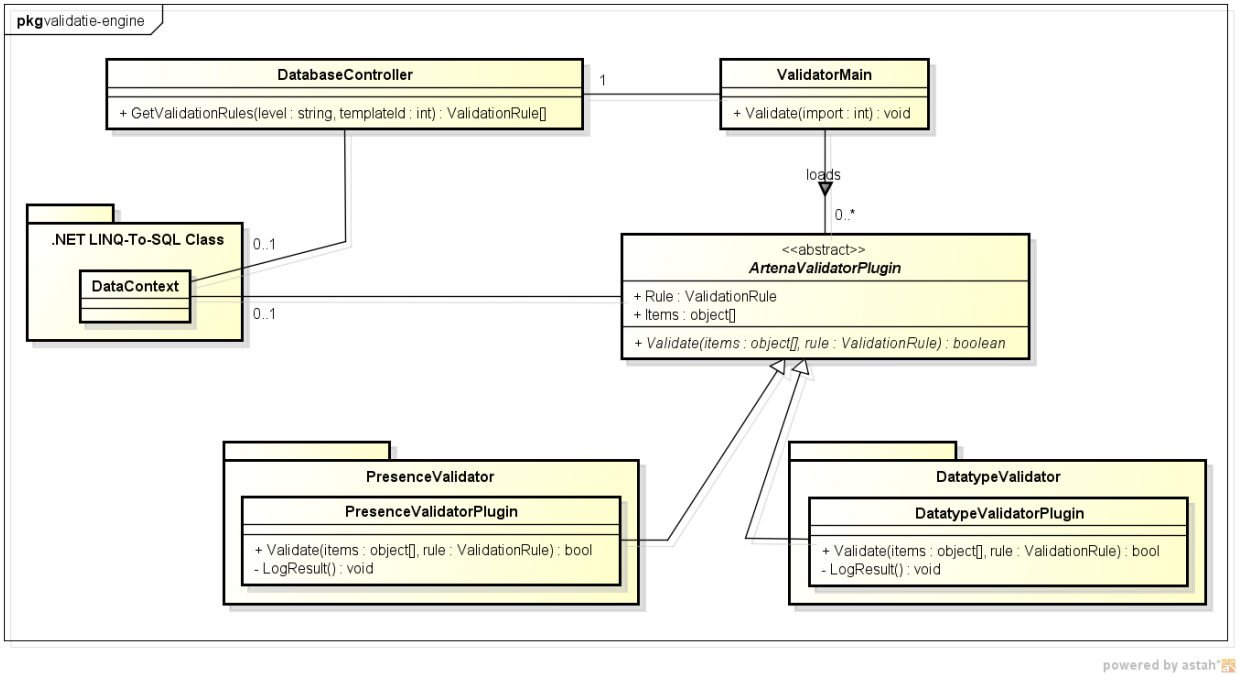
Figuur 24: Abstracte klasse validatie plug-in

De methode **validate()** moet per definitie geïmplementeerd worden door de overervende klassen (dit zijn dus de plug-in klassen) zodat de validatie-engine deze methode altijd kan aanroepen vanuit een validatieroutine. In §8.2.6 staat het domeinmodel van de validatie-engine waarin deze klasse ook verwerkt is.

Voor het inladen van een validatie plug-in worden twee gegevens uit de tabel *ValidationType* gebruikt:

- De kolom PluginLocation, met daarin de locatie (bijv. C:\Software\DatatypeValidator.dll)
- De kolom PluginFullType, met daarin het type (bijv. Artena.Validators.DatatypeValidator)

8.2.6 Domeinmodel



Figuur 25: Domeinmodel validatie-engine

Het domeinmodel maakt duidelijk dat de structuur die wordt gebruikt in de programmatuur heel veel overeenkomt met het conversiesysteem:

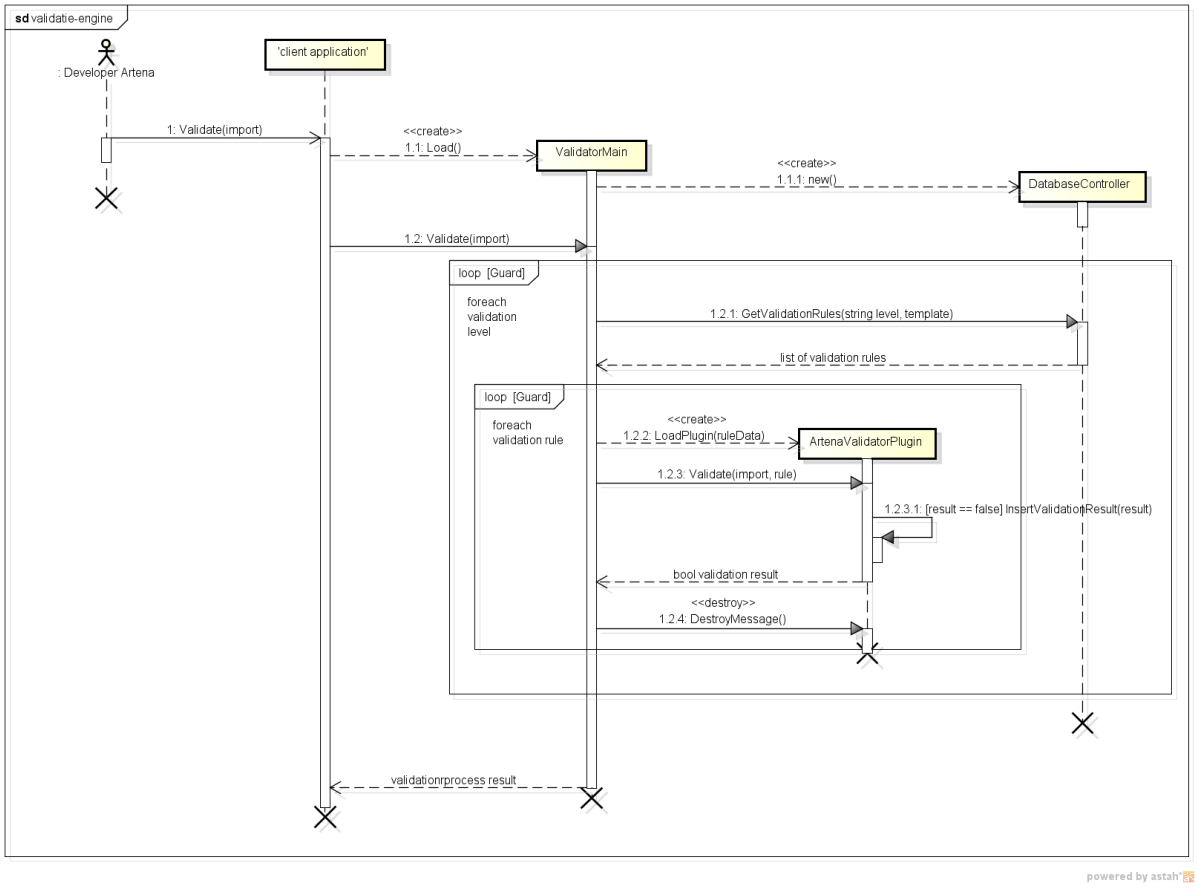
- Eén class, ValidatorMain controleert de gehele validatieroutine
- De ArtenaValidatorPlugin beschrijft het plug-in object
- De DatabaseController wordt gebruikt voor de uitvoer van (de meeste) queries.
- LINQ-to-SQL wordt gebruikt om de database-entiteiten aan te spreken als objecten

Naast deze klassen worden ook twee validatie plug-ins beschreven. Beide plug-ins erven over van de plug-in abstracte base class en voeren een ander type validatie uit. Andere plug-in klassen zijn weggelaten zodat het diagram niet te onduidelijk zou worden.

8.3 Constructiefase

Tijdens de constructie is het ontwerp van de validatie-engine uit het ontwerprapport geïmplementeerd. Van de geplande onderdelen zijn een aantal dingen geïmplementeerd en een aantal onderdelen uitgesteld. In §8.3.1 wordt een sequencediagram getoond van de validatieroutine terwijl in §8.3.2 een beschrijving wordt gegeven van de onderdelen die zijn uitgesteld naar de derde iteratie.

8.3.1 Sequencediagram



Figuur 26: Sequence diagram validatieproces (uit bijlage K)

Net zoals het conversieproces wordt het validatieproces gestart vanuit een client applicatie. De client applicatie gebruikt voor het conversiesysteem is ook gebruikt voor de validatie-engine. De klassen in het diagram komen overeen met de klassen beschreven in het klassendiagram, net zoals de methoden die worden aangeroepen (bijvoorbeeld de methode **1.2.3: Validate()**).

8.3.2 Activiteiten buiten planning

Tijdens de uitvoer van de constructie van de validatie-engine kwam ik erachter dat voor een aantal activiteiten te weinig tijd was ingepland om ze uit te voeren in de tweede iteratie. Om deze activiteiten alsnog uit te voeren zijn deze verschoven naar de derde iteratie:

1. Implementatie table-level plug-in: *RowsRange*

2. Implementatie row-level plug-in: *FieldCount*
3. Implementatie column-level plug-in: *FieldsRange*
4. Implementatie field-level plug-ins: *Format, Range, Lookup, Relationship*

De tijd benodigd voor de implementaties van verschillende soorten validatieregels was verkeerd geschat. Dit kwam door de verschillen in afhandeling per type validatie plug-in. Zo was de implementatie van een datatype validatie (waarbij de waarde van een veld wordt vergeleken met een datatype) meer werk dan de implementatie van een presence validatie (waarbij alleen gecontroleerd wordt of een veld gevuld is), terwijl ik een kleiner verschil in werk tussen alle plug-ins had verwacht.

Daarnaast was het stoppen van het validatieproces wanneer een systeemfout zich voordeed (door middel van onder andere try-catch statements, zie bijlage I) was niet overal op dezelfde wijze geïmplementeerd. Het heeft mij veel tijd gekost om dit overal (in alle belangrijke methoden van de programmatuur) gelijk te trekken.

De functionaliteit en correctheid van de validatieroutine zelf heeft dus prioriteit gehad tijdens de uitvoer van deze tweede iteratie in plaats van de implementatie van alle validatietypen door meerdere validatie plug-ins te implementeren (zoals deze gedefinieerd zijn in de requirements). Met de drie geïmplementeerde validatietypen (uniqueness, presence en datatype) was de functionaliteit van de validatieroutine prima te implementeren en te verbeteren.

9. Iteratie 3: Log systeem & Artena Data Validator

In deze derde en laatste iteratie is net zoals beschreven in de aanpak en planning de Artena Data Validator ontstaan. Deze applicatie is samengevoegd met de systemen uit de eerste twee iteraties. Dit log systeem is tijdens deze iteratie ook ontworpen en geïmplementeerd. In de paragrafen 9.1 en 9.2 wordt de specificatie en het ontwerp van het log systeem en de Artena Data Validator beschreven.

Tijdens de constructiefase heb ik het conversiesysteem en de validatie-engine volgens het ontwerp samengevoegd en vervolgens het log systeem toegevoegd aan het geheel. Het resultaat daarvan beschrijf ik in paragraaf 9.3. In paragraaf 9.4 beschrijf ik de systeemtest die ik heb uitgevoerd om de aanwezigheid van gewenste functionaliteiten te controleren. In paragraaf 9.5 beschrijf ik tot slot de systeemdocumentatie die ik heb opgesteld met behulp van de uiteindelijke implementatie van de Artena Data Validator in deze iteratie.

9.1 Definitiefase

De definitiefase van de derde iteratie heeft twee belangrijke onderdelen behandeld. Allereerst zijn de requirements met betrekking tot het log systeem toegevoegd aan de bestaande requirements. Door terug te blikken op de implementaties van het conversiesysteem en de validatie-engine zijn ook nog twee extra requirements toegevoegd(zie §9.1.1). Het andere belangrijke onderdeel is de beschrijving van de samenvoeging van het conversiesysteem en de validatie-engine door het opstellen van een nieuw use case diagram (zie §9.1.2). Het volledige analyserapport van de derde iteratie is bijgevoegd in bijlage L

9.1.1 Requirements

Nr.	Systeem	St.	Pr.	Eis
F26	Conversie	S2, S3	2	Het moet mogelijk zijn meerdere CSV formaten te ondersteunen. Om dit af te handelen moet het systeem om kunnen gaan met meerdere *delimeters(, of ; of : of \t)* en *indicators(‘ of “)*.
F27	Log	S2	1	Fouten tijdens de uitvoer van een conversieproces of validatieproces moeten worden geregistreerd volgens onderstaande criteria: • Het deel van het systeem waarin de fout ontstond • Het proces waarin de fout ontstond • De kenmerken (details) van de fout
F28	Log	S2	1	Het systeem moet meerdere soorten log niveaus kunnen ondersteunen. De twee niveaus die in ieder geval ondersteund moeten worden: • Warning • Error
F29	Conversie, Validatie, Log	S2, S3	1	De Artena Data Validator moet een op zich zelf staande softwarebibliotheek zijn die kan worden gebruikt door een willekeurig systeem via een interface.

Bovenstaande tabel toont vier requirements die ik op basis van gesprekken met de tweede en derde stakeholder heb kunnen toevoegen aan de volledige lijst met requirements uit de eerste iteratie.

Functionele eis F26 is vastgelegd omdat de bestandslezer verantwoordelijk voor het uitlezen van CSV bestanden tot dan toe maar twee CSV formaten ondersteunde (namelijk standaard CSV en CSV van een klant van Artena, zie ook §7.1.3). Het bleek achteraf echter gewenst meer dan deze twee formaten te ondersteunen door variabele *indicators* en *delimeters* te implementeren.

1. *Delimiters:* scheidingsteken tussen de waarden, zoals een komma
2. *Indicators:* teken dat de om de waarden heen staat, zoals een quote of een apostrof

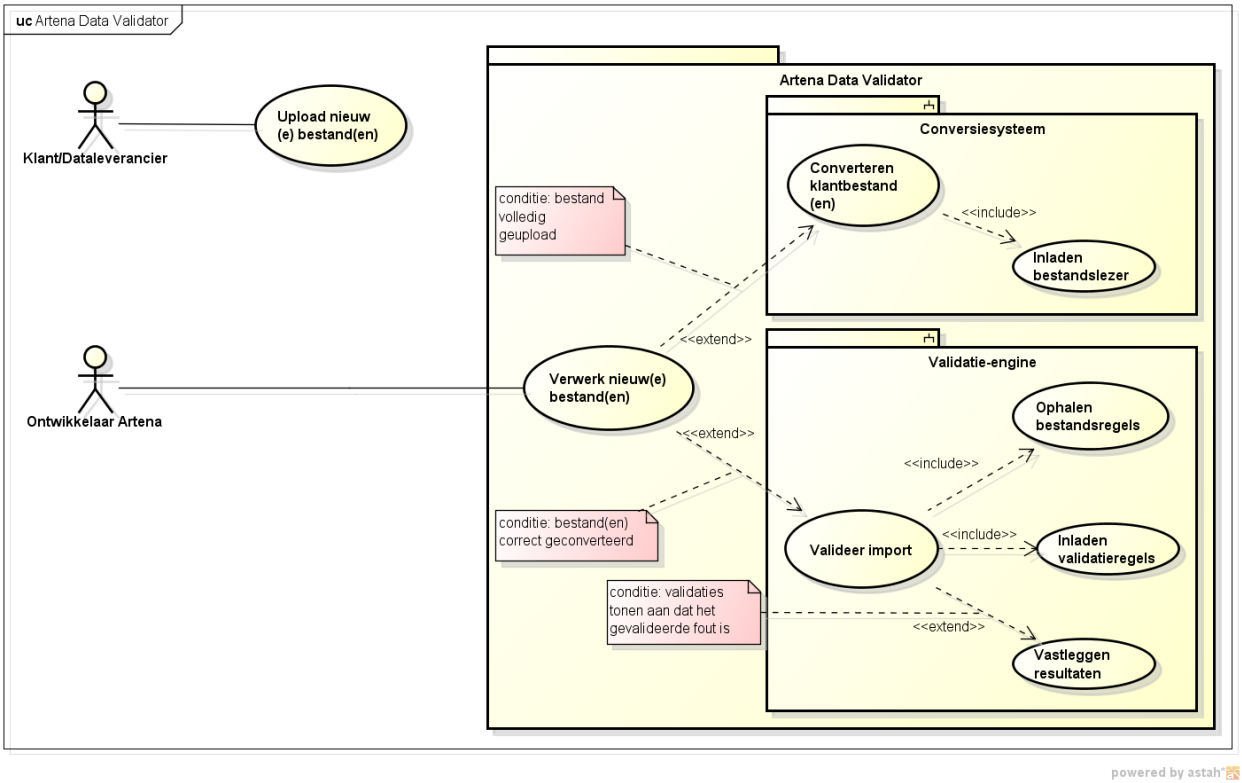
Functionele eisen F27 en F28 beschreven de gewenste functionaliteiten voor het log systeem terwijl niet-functionele eis **NF6** (zie ook bijlage L) de gewenste flexibiliteit van het log systeem beschrijft:

Nr.	Systeem	St.	Pr.	Cat.	Eis
NF6	Log	S2	2	3	Het log systeem moet ook een mate van flexibiliteit bevatten waarbij een ondersteuning voor meerdere soorten loggers gewenst is.

De laatste eis heeft betrekking op de Artena Data Validator in zijn geheel en beschrijft de wijze waarop het systeem gebruikt moet kunnen worden door een ander systeem.

9.1.2 Use cases

Op basis van de functionele eis F29 en de use case diagrammen uit de analyserapporten van de eerste en de tweede iteratie is een use case diagram ontstaan waarin de Artena Data Validator als één systeem is beschreven. Zie figuur 27.



Figuur 27: Use case diagram Artena Data Validator

Het belangrijkste onderdeel uit het diagram is de use case die de samenkomst van de processen van het conversiesysteem en de validatie-engine definieert in een nieuw bovenliggend proces, de use case 'verwerk nieuw(e)bestand(en)'.

Naam	Verwerk nieuw(e) bestand(en)
Aannamen	- De gegevens van de betreffende klant zijn bekend bij de ontwikkelaar van Artena, zoals de bestandslocaties en de benodigde templates - Het/De nieuwe bestand(en) zijn bekend bij de ontwikkelaar van Artena
Beschrijving	(1) De ontwikkelaar voert de bestandsgegevens in en de identificaties van de templates en start de gehele import en validatie m.b.v. een button in de interface. (2) Het systeem vraagt de gegevens op van de betrokken klant inclusief de templates van bestanden die ondersteund worden voor die klant **[I][II]** (3) Het systeem bepaalt welke templates aan welke bestanden gekoppeld zijn en geeft verwijzingen naar deze templates samen met het/de klantbestand(en) door aan het conversiesysteem. (4) [Extension point: upload bestand volledig uitgevoerd] **Use case 'Converteren klantbestand(en)'** (5) Het systeem geeft een verwijzing naar de zojuist geconverteerde bestandsregels mee aan de validatie-engine. **[III]** (6) [Extension point: import klantbestand(en) correct uitgevoerd] **Use case 'Valideer import'**
Uitzondering	**[I]** Databasefout(bijvoorbeeld geen verbinding): het proces wordt afgebroken en een systeemfout wordt vastgelegd via het log systeem. **[II]** Het systeem herkent de ingevoerde template niet, waardoor het proces wordt afgebroken en een systeemfout wordt vastgelegd via het log systeem. **[III]** De conversie van de aangeleverde data is mislukt, waardoor het validatieproces niet opgestart zal worden.
Resultaat	Het/de aangeleverde bestand(en) is/zijn geconverteerd en gevalideerd en de resultaten van deze processen kunnen eventueel worden verwerkt tot een rapportage voor de klant (dit valt buiten de scope van het project).

Vanuit het proces beschreven in bovenstaande use case beschrijving wordt dus eerst het conversieproces aangeroepen en daarna het validatieproces (mits het conversieproces succesvol is). Dit proces is dus wat de ontwikkelaar van Artena moest startten vanuit een client (zoals de gebruikersinterface) door de Artena Data Validator als DLL in te laden.

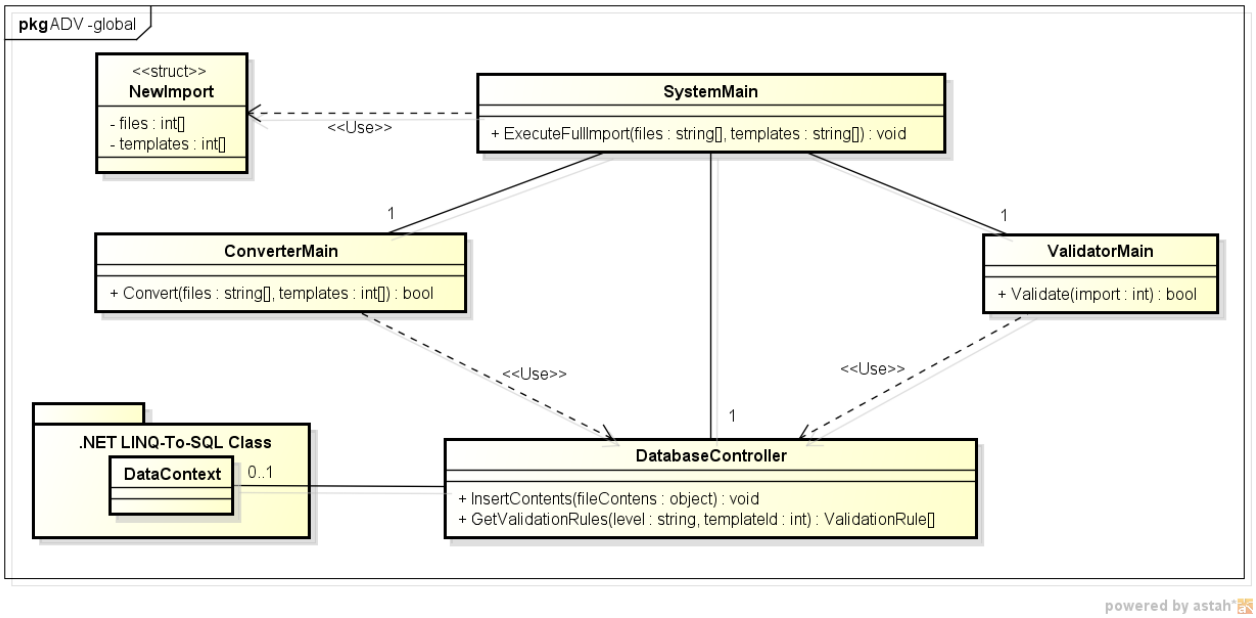
9.2 Ontwerpfase

De ontwerpfase van heeft twee onderdelen, het samenvoegen van het conversiesysteem en de validatie-engine en het ontwerp voor het log systeem opstellen. In paragraaf 9.2.1 wordt het ontwerp van de Artena Data Validator beschreven, waarin de programmatuur centraal staat. In paragraaf 9.2.2 wordt het ontwerp van het log systeem beschreven. In bijlage M is het volledige ontwerprapport bijgevoegd.

9.2.1 Ontwerp Artena Data Validator

Het ontwerprapport van de Artena Data Validator is volledig georiënteerd rondom de programmatuur omdat de database gebruikt door het conversiesysteem en de validatie-engine in één keer gebruikt kon worden alle benodigde functionaliteiten bevatte. Alleen voor het log systeem is een uitbreiding gedaan op de bestaande database (zie §9.2.2).

Het ontwerp van de programmatuur van de Artena Data Validator bestaat uit een aantal klassendiagrammen en een UML sequence diagram. Van de drie klassendiagrammen uit het ontwerprapport is één het belangrijkste, zie figuur 28.



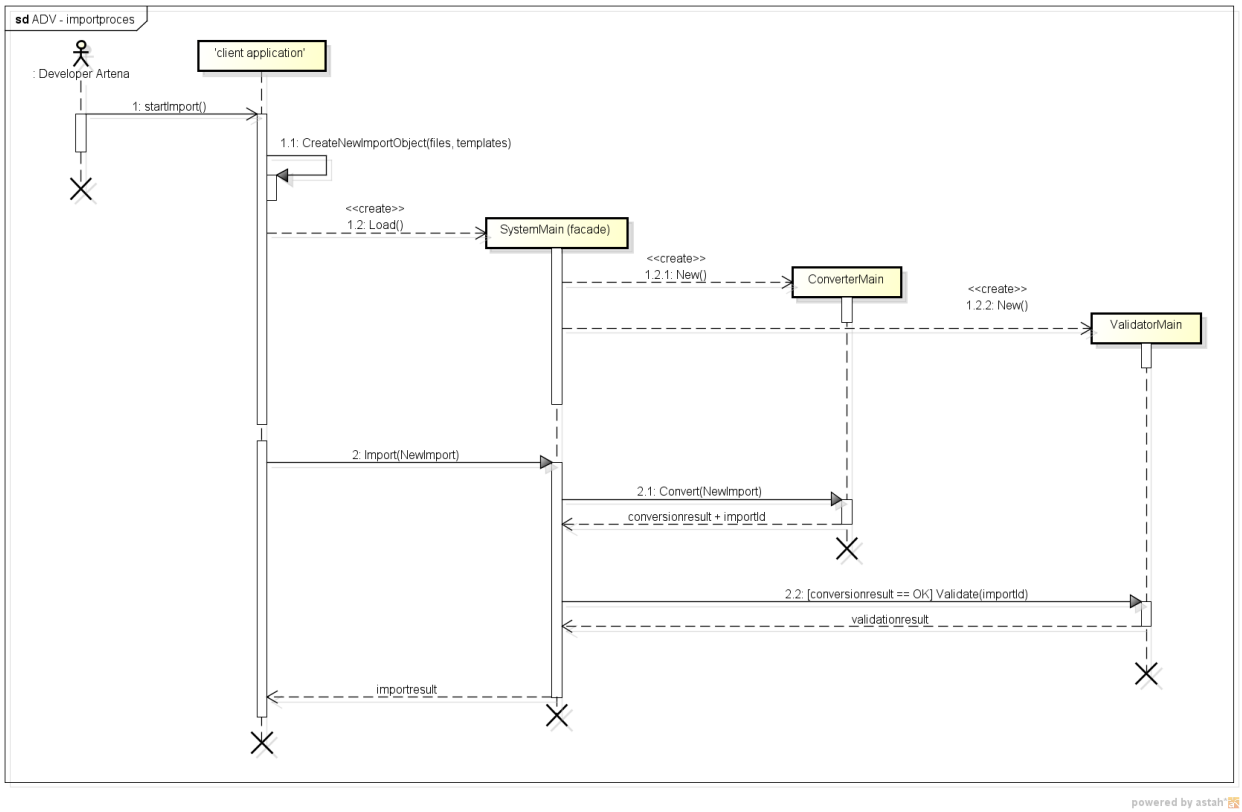
Figuur 28: Klassendiagram Artena Data Validator – Globaal

Het diagram toont de introductie van een nieuwe klasse, *SystemMain*. Deze klasse is verantwoordelijk voor het overzien van de gehele conversie –en validatieprocessen zoals beschreven in de use case beschrijving van §9.1.2. Het diagram toont duidelijk dat de SystemMain klasse de *parent* is van de ConverterMain en ValidatorMain klassen uit de eerste twee iteraties en dat de DatabaseController is vastgelegd in deze klasse.

Zoals beschreven in het ontwerprapport is deze klasse een *Facade* klasse die toegang biedt tot de functionaliteiten van de Artena Data Validator voor client applicaties van buitenaf. Deze klasse is Facade omdat de veranderingen aan de conversie –of validatieprocessen waaraan de SystemMain toegang biedt geen veranderingen teweegbrengen aan de klasse zelf.

Een andere overweging die ik bij het ontwerp heb moeten maken is dat deze klasse geen *Singleton*(zie ook §9.2.3) is. Bij de toepassing van een Facade class wordt vaak een Singleton toegepast omdat er vaak maar één instantie van een Facade class is toegestaan. In dit geval gaat dat echter niet op omdat de client applicatie die de Artena Data Validator inlaadt meerdere instanties van SystemMain aan moet kunnen maken om conversies en validaties parallel aan elkaar uit te voeren.

Met behulp van het klassendiagram, het nieuwe use case diagram uit §9.1.2 en de kennis uit de vorige iteraties kon ik vrij gemakkelijk het gehele importproces definiëren in een UML Sequence diagram, zie figuur 29.

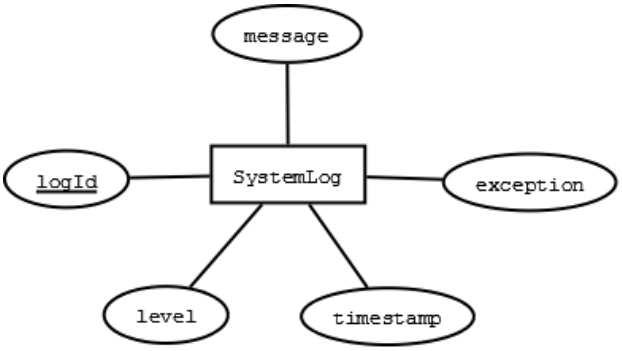


Figuur 29: Sequence diagram importproces (uit bijlage M)

In het diagram is duidelijk te zien dat de SystemMain klasse de functionaliteiten geïmplementeerd in de andere twee 'Main' klassen gebruikt. De twee processen, conversie en validatie, worden gecombineerd in deze nieuwe klasse.

9.2.2 Ontwerp log systeem – database

Het ontwerp van het log systeem bestaat uit twee delen, een databasedeel en programmatuur. Voor het databasedeel moest een uitbreiding worden gedaan op de bestaande database gebruikt door de Artena Data Validator om zo de vereiste gegevens met betrekking tot systeemfouten vast te leggen. Het databaseontwerp bleek relatief simpel te zijn, namelijk één extra tabel zonder relaties met andere tabellen.



Figuur 30: ERD log systeem

De tabel SystemLog (figuur 30) bevat alle functionaliteiten benodigd om de gewenste gegevens van een systeemfout vast te leggen. In het/de attribuut/kolom *message* kan een willekeurige foutmelding worden vastgelegd terwijl in de kolom *exception* een volledige beschrijving van

een voorgedane exceptie kan worden vastgelegd. Omdat beide typen meldingen zeer groot kunnen zijn moest hiermee rekening worden gehouden bij het RIM. De kolom message kan een *varchar* met lengte van 1000 karakters bevatten terwijl de kolom exception een *varchar* met maximale lengte kan bevatten:

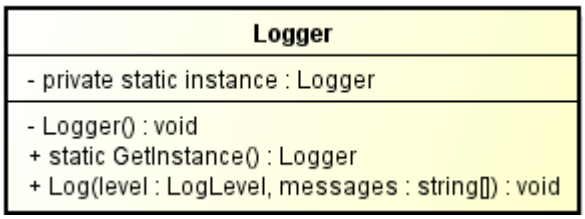
```
CREATE TABLE [SystemLog](
	[LogId] [int] IDENTITY(1,1) NOT NULL,
	[Timestamp] [datetime] NOT NULL,
	[Level] [varchar](100) NULL,
	[Message] [varchar](1000) NULL,
	[Exception] [varchar](max) NULL,
 CONSTRAINT [PK_ConversionLog] PRIMARY KEY CLUSTERED
(
	[LogId] ASC
))
```

9.2.3 Ontwerp log systeem – programmatuur

Naast de functionele eisen gesteld aan het log systeem waarin de ondersteuning voor meerdere logniveaus en de gegevens die in een logbericht vastgelegd moesten kunnen worden waren er ook twee andere wensen:

- Flexibiliteit, het ondersteunen van meerdere typen loggers waardoor de Artena Data Validator logberichten op verschillende manieren zou kunnen vastleggen (niet-functionele eis **NF6**)
- Een logsysteem moet overal in de applicatie beschikbaar zijn zodat op elke ‘plek’ een aanroep gedaan kan worden wanneer daar een systeemfout zou ontstaan

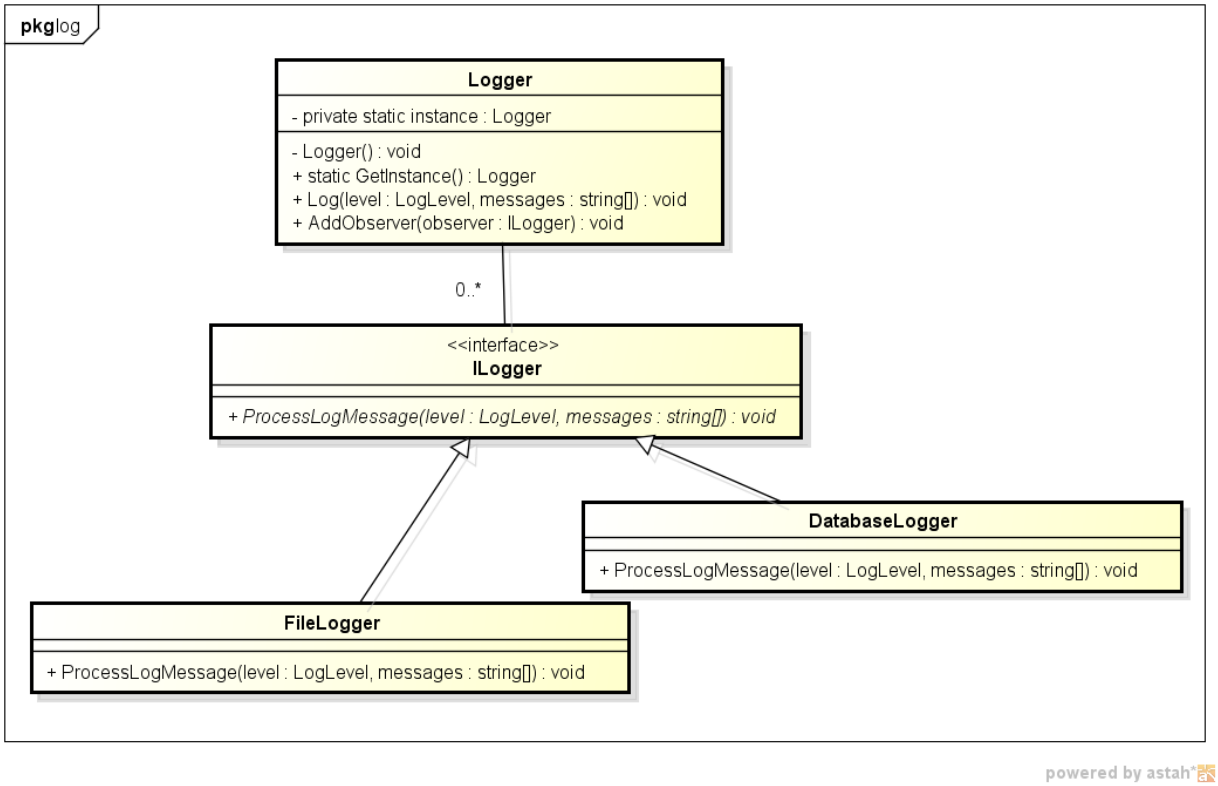
Ik heb besloten om twee design patterns toe te passen om aan deze eisen te voldoen. Het eerste pattern is het welbekende Singleton pattern waardoor de logger in het gehele systeem beschikbaar is. Tevens garandeer je hiermee dat er maar één logger instantie aanwezig is in het systeem, zie figuur 31.



Figuur 31: Logger klasse – Singleton pattern

Een andere oplossing voor dit probleem zou het gebruik zijn van een *static* methode, waarbij er ook maar één globale methode aanwezig is voor het loggen. Met behulp van een Singleton object behoudt je echter de object georiënteerde implementatie, doordat je de logger niet als een groep statische methoden implementeert maar als een echte klasse (en dus als object). Daarbij komt ook dat het implementeren van de gewenste flexibiliteit ook een dergelijke implementatie vereist.

Voor de implementatie van de gewenste flexibiliteit is besloten om een Observer pattern toe te passen in de Singleton logger. Met behulp van deze structuur houdt de Logger klasse meerdere soorten loggers bij (de observers) die allemaal aangeroepen worden om een logbericht weg te schrijven. Zie figuur 32.



Figuur 32: Log systeem - Observer pattern

Wanneer de *Logger* een logbericht ontvangt dat zorgt hij ervoor dat alle toegevoegde observers dit logbericht ontvangen. Hoe deze observers het logbericht vervolgens verwerken en eventueel wegschrijven is aan de implementaties van deze observers. In het voorbeeld van figuur 32 is een DatabaseLogger en FileLogger toegevoegd.

Door het toepassen van deze combinatie van design patterns kan een logbericht vanuit overal in de Artena Data Validator worden weggeschreven. Om dit te doen hoeft het proces waarin de systeemfout is ontstaan alleen maar de methode *Log()* aan te roepen van de instantie van de Logger (er is er maar één). Uiteindelijk zal dit bericht dan naar één of meerdere soorten media worden weggeschreven.

9.3 Constructiefase

In deze fase heb ik het ontwerp, beschreven in het ontwerprapport, ontwikkeld. Dit houdt in dat de processen van het conversiesysteem en de validatie-engine zijn samengevoegd in één systeem, zoals beschreven in het UML sequence diagram van §9.2.1. Tevens is het log systeem toegevoegd als registratiemiddel voor systeemfouten. Tot slot heb ik de validatie plug-ins die in de tweede iteratie niet ontwikkeld waren (zie §8.3.2), geïmplementeerd.

Tijdens deze fase heb ik een constructierapport opgesteld waarin de een technische beschrijving van belangrijke delen van de Artena Data Validator is vastgelegd. Dit rapport is bijgevoegd als bijlage N.

9.3.1 Code fragmenten

Voor de implementatie van de Artena Data Validator is gebruikt gemaakt van C# met behulp van Visual Studio 2010. Bij de ontwikkeling van bepaalde ontwerptechnieken zijn bepaalde stukken code cruciaal.

Plug-in systeem

Voor het plug-in systeem is gebruikt gemaakt van Reflection om een plug-in in te laden. Hierbij wordt met behulp van een type en een locatie een DLL ingeladen in het proces dat gebruik wil maken van de plug-in. Hieronder is de code om een validatie plug-in in te laden weergeven in zijn simpelste vorm:

```
Assembly assembly = Assembly.LoadFrom(location);
object o = assembly.CreateInstance(type);
ArtenaValidatorPlugin plugin = o as ArtenaValidatorPlugin
```

Singleton & Double-checked locking pattern

Voor het logsysteem wordt gebruik gemaakt van een Singleton pattern om de logger overal in de Artena Data Validator beschikbaar te maken. In de Singleton zit tevens het Double-checked locking pattern verwerkt om te voorkomen dat er problemen ontstaan wanneer meerdere threads gelijktijdig toegang willen hebben tot de logger (mocht dat in de toekomst kunnen gebeuren):

```
        public static Logger Instance
        {
            get
            {
                // If this is the first time we're referring to the
                // singleton object, the private variable will be null.
                if (_instance == null)
                {
                    // for thread safety, lock an object when
                    // instantiating the new Logger object. This prevents
                    // other threads from performing the same block at the
                    // same time.
                    lock (_lock)
                    {
                        // Double-checked locking pattern
                        if (_instance == null)
                        {
                            _instance = new Logger();
                        }
                    }
                }
                return _instance;
            }
        }
```

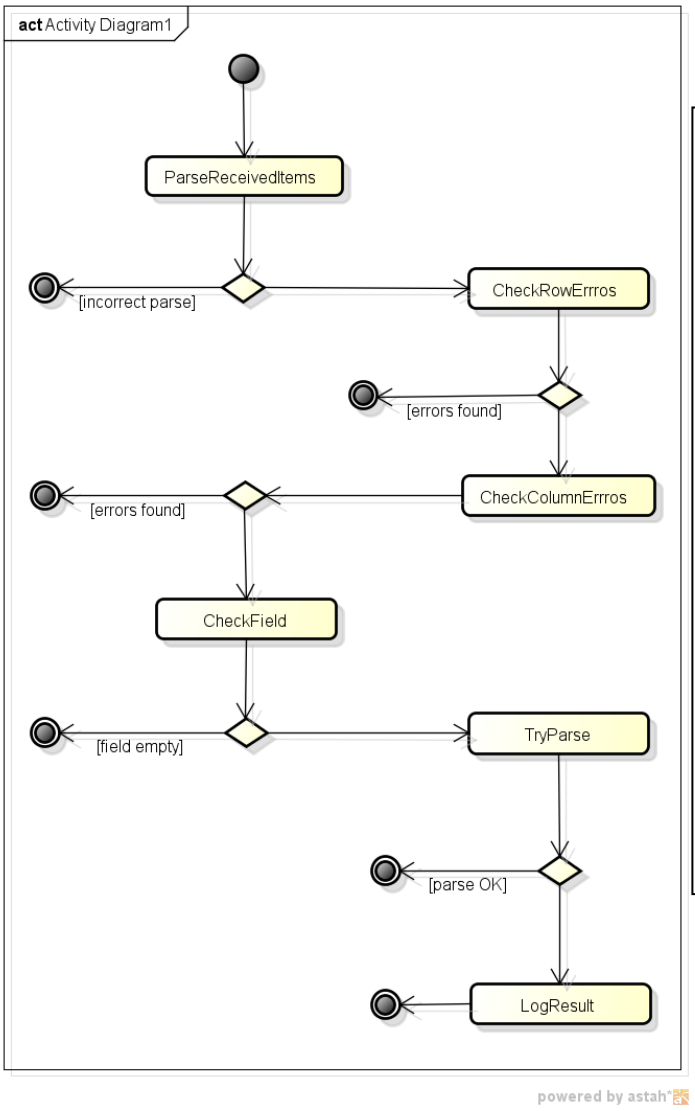
Observer pattern

Voor het log systeem is ook gebruik gemaakt van een Observer pattern om meerdere loggers te ondersteunen. Hieronder is de code weergeven wanneer het Logger object een log bericht doorgeeft aan al zijn observers:

```
        public void Log(LogLevel logLevel, params string[] messages)
        {
            foreach (ILogger observer in _observers)
            {
                observer.ProcessLogMessage(logLevel, messages);
            }
        }
```

9.3.2 Activiteitendiagram validatie

Om het proces, wat zich binnen één methode afspeelt, te beschrijven heb ik gebruik gemaakt van een UML activiteitendiagram. In het diagram van figuur 33 wordt het proces van een datatype validatie beschreven wanneer de *validate()* methode is aangeroepen vanuit de validatie-engine.



Alvorens de controle op het datatype van het ontvangen veld uitgevoerd kan worden moeten eerst meerdere controles worden uitgevoerd.

De ontvangen gegevens (velden, condities) worden gecontroleerd net zoals eerder vastgelegde fouten uit het validatieproces (*CheckRowErrors/CheckColumnErrors)*.

Wanneer er fouten ontstaan bij deze controles dan wordt de validatie geannuleerd.

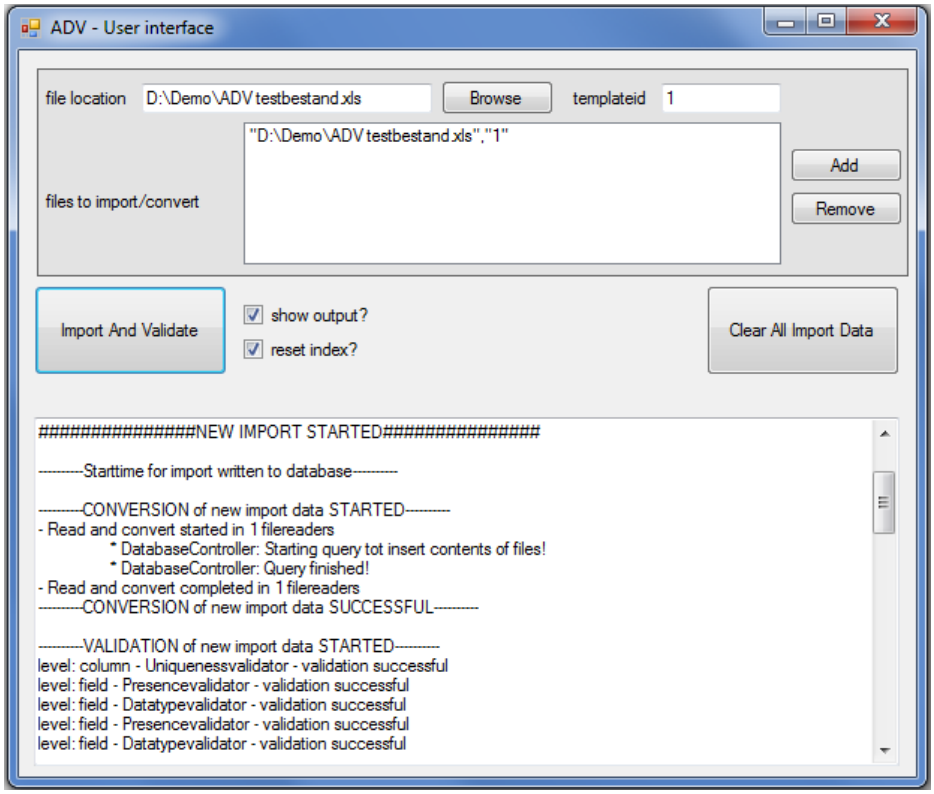
Tot slot is duidelijk te zien dat het vastleggen van een validatieresultaat (*LogResult*) alleen wordt uitgevoerd wanneer het parsen van het ontvangen veld naar het gewenste datatype mislukt.

Figuur 33: UML activiteitendiagram datatype validatie

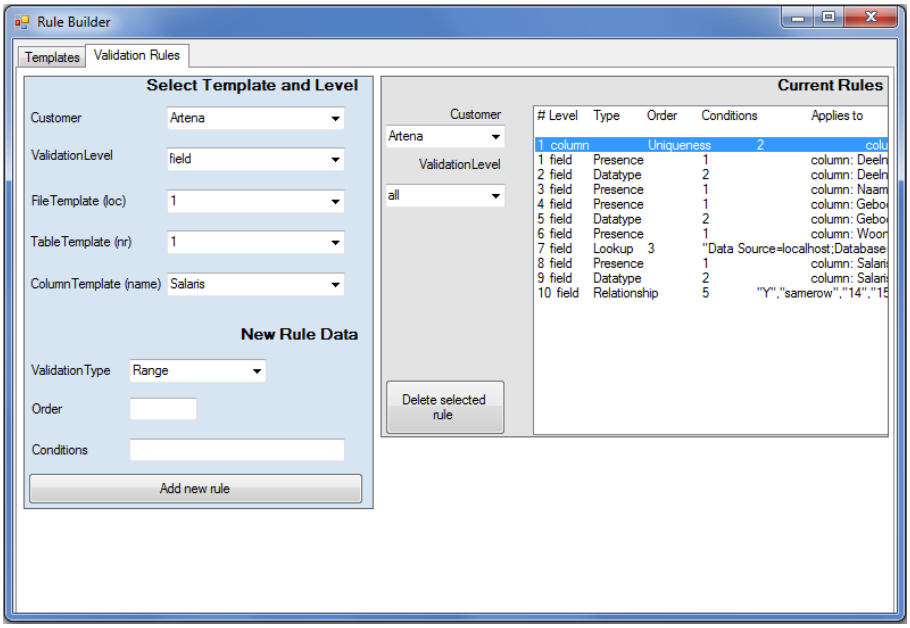
9.3.3 Gebruikersinterface(s)

Tijdens de ontwikkeling van de Artena Data Validator heb ik gebruik gemaakt van twee verschillende gebruikersinterfaces om de functionaliteiten van de Artena Data Validator te gebruiken.

Figuur 34 toont de laatste versie van de gebruikersinterface waarmee een conversie en validatie op één of meerdere bestanden gestart kan worden. Deze interface is een doorontwikkeling van de interface van §7.1.3.



Figuur 34: User interface 1 - client



Figuur 35: User interface 2 - beheer

Figuur 35 toont een andere gebruikers interface waarmee op een gemakkelijke wijze nieuwe klanten, templates en validatieregels (op verschillende niveaus) toegevoegd kunnen worden aan de database.

Door deze interface te gebruiken hoefde ik die gegevens niet meer via het DMBS te wijzigen, waardoor het beheer van deze gegevens makkelijker was.

9.4 Systeemtest

Na de afronding van de derde en laatste constructiefase is een systeemtest uitgevoerd op de Artena Data Validator. Zoals beschreven in het plan van aanpak is deze test ervoor bedoeld om vast te stellen dat alle functionaliteiten, zoals deze beschreven waren in de requirements, correct geïmplementeerd zijn in de Artena Data Validator. Voor de test is een testplan, testscript en een testrapport opgesteld, welke zijn bijgevoegd in bijlage O.

9.4.1 Selecteren testspecificatietechniek

De eerste stap in het testproces is het opstellen van een testplan waarin onder andere de testspecificatietechniek wordt geselecteerd. Volgens TMap[15] 'is een testspecificatietechniek een gestandaardiseerde manier om vanuit uitgangsinformatie testgevallen af te leiden'.

Het belangrijkste onderdeel van een dergelijke techniek is dus de beschrijving van 'wat' (de testbasis) getest wordt en hoe de testgevallen hieruit worden vastgelegd om een test uit voeren.

Bij het selecteren van deze techniek heb ik dus gebruik gemaakt van de methode TMap. In deze methode zijn een aantal technieken vermeldt waarmee een systeemtest uitgevoerd kan worden. Een systeemtest is een *Blackbox* test waarmee alleen gekeken wordt naar de buitenkant van de software (dus niet naar interne processen, maar naar de waarneembare resultaten van die processen).

Na de mogelijke technieken te hebben onderzocht is gekozen voor de *DataFlowTest (DFT)* om de testgevallen te beschrijven. Ik heb hiervoor gekozen omdat volgens de TMap methode deze testtechniek de meeste vrijheid geeft in de wijze waarop de testgevallen opgesteld kunnen worden. Met deze techniek was het dus mogelijk om op basis van mijn eigen inzicht testgevallen op te stellen om de verschillende functionele eisen te bewijzen. Een andere (meer formele) techniek die gekozen had kunnen worden is de *ProcesCyclusTest*. Hier is niet voor gekozen omdat deze test zich meer focust op de inpasbaarheid van het systeem in bijvoorbeeld de productieomgeving. De Artena Data Validator bevond zich op dat moment nog niet in die situatie.

9.4.2 Opstellen testgevallen

Nadat het testplan was opgesteld heb ik testgevallen kunnen opstellen waarmee de test uitgevoerd zou kunnen worden. De testgevallen zijn opgesteld vanuit de in het testplan beschreven testbasis. Globaal gezien bestaat de testbasis uit:

- De (functionele) requirements
- Zelf aangemaakte bestanden
- Oude importbestanden van klanten van Artena
- De database van de Artena Data Validator met daarin de juiste templates en overige gegevens om een import uit te voeren.

In het testscript zijn alle testgevallen onderverdeeld in meerdere testcases, welke weer zijn onderverdeeld in meerdere testeenheden. Deze testeenheden beschrijvingen de 'subsystemen' van de

[15] [Pol, M. & Teunissen, R. & Van Veen, E.] (2002).

Artena Data Validator. De tabel hieronder toont een aantal testgevallen waarmee de werking van een aantal validatieregels getest is:

Test geval	Validatietype	Kolomnaam	Parameters bestandsinhoud
1	Presence	Naam	Alle velden de kolom zijn ingevuld
2	Presence	Naam	Eén van de velden in de kolom is leeg
3	Datatype (int)	Klantnummer	Aantal velden in de kolom zijn getallen
4	Datatype (int)	Klantnummer	Eén van de velden in de kolom is tekst
5	Datatype (date)	Geboortedatum	Aantal velden in de kolom zijn datums
6	Datatype (date)	Geboortedatum	Eén van de velden in de kolom is tekst

Elk van deze testgevallen is vervolgens omgezet in een aantal testacties. Alle testacties samen vormen het gehele testscript. Sommige testacties bevatten specifieke acties die uitgevoerd moeten worden op de testbasis om de gewenste testgevallen te handhaven:

1. Valideer het zelfgemaakte Excel2007 bestand (testgeval 1-8)
    a. *File template = 2*

2. Valideer het zelfgemaakte Excel2007 bestand (testgeval 1-8)
    a. *File template = 2*
    b. *Zorg voor de afwijkingen zoals beschreven in de testgevallen 2, 4, 6, 8*

Tot slot heb ik de verwachte resultaten in het testscript vastgelegd, waar de daadwerkelijke resultaten mee vergeleken moesten worden:

Onderstaande controles moet worden uitgevoerd op testactie 1
C01: *Geen validatieresultaten voor de vier validatieregels uit testgeval 1-8*

Onderstaande controles moet worden uitgevoerd op testactie 2
C02-A: *Validatieresultaat m.b.t. presence validatie (kolom naam)*
C02-B: *Validatieresultaat m.b.t. datatype validatie (kolom klantnummer)*
C02-C: *Validatieresultaat m.b.t. datatype validatie (kolom geboortedatum)*
C02-D: *Validatieresultaat m.b.t. datatype validatie (kolom salaris)*

9.4.3 Belangrijkste resultaten

Na de uitvoer van het gehele testscript zijn alle testresultaten vastgelegd in het testrapport (bijlage O). Uiteindelijk kon ik concluderen dat alle functionele eisen correct geïmplementeerd waren in de Artena Data Validator. Bij bepaalde testgevallen kwamen echter ook enkele onverwachte resultaten (fouten) naar voren:

1. Bij het uitlezen van CSV bestanden worden ‘lege’ regels in een bestand niet verwerkt, maar de aanwezige regels in het bestand die na een lege regel komen worden ook niet meegenomen. Het uitlezen van een CSV bestand stopt dus bij een lege regel.

2. Bij de performance test, waarbij twee grote deelnemerbestanden geconverteerd en gevalideerd werden, kwam naar voren dat de Artena Data Validator 'tamelijk' traag was. Dit houdt in dat het systeem over het controleren van bepaalde kolommen met +/- 25000 regels meer dan enkele minuten bezig was.

De Artena Data Validator is niet afgekeurd door deze resultaten van de systeemtest omdat beide problemen niet overeenkwamen met de requirements. Zoals ook beschreven in de conclusie van het testrapport moet aan deze problemen wel aandacht worden besteed zeker omdat het vanuit 'common sense' gezien niet goed is dat het systeem dergelijke problemen bevat.

9.5 Systeemdocumentatie

De laatste stap in het project is het opstellen van systeemdocumentatie voor de Artena Data Validator. Aangezien het systeem eerst vooral door ontwikkelaars van Artena gebruikt gaat worden vond stakeholder 1 (de opdrachtgever) het vooral belangrijk dat de *features* van het systeem (in het Engels) beschreven zouden worden.

De opgestelde documentatie is bijgevoegd in bijlage P en bevat onderstaande onderdelen:

1. Een beschrijving van het gehele importproces, inclusief conversie en validatie
2. Een beschrijving van het plug-in systeem en de benodigde gegevens
3. Een beschrijving van de ondersteunde bestandsformaten (Excel en CSV)
4. Een beschrijving van de verschillende validatieniveaus, ondersteunde validatietypen per niveau en mogelijke condities per validatietype.

Het document bevat bijvoorbeeld een beschrijving van het validatietype *lookup* op veldniveau:

This validation checks if the value of the checked field matches formats/values defined in a specific lookup table. These lookup tables are used by other systems of Artena and define the required format or value for certain fields. In order to execute these validations the condition needs to hold the info needed to retrieve these lookup values.

Possible Condition	**Description**	**Notation "connectionstring","scheme", "table","column"**
1	The value of the checked field needs to be equal to the values defined in the lookup table. The lookup table is located in the database defined by the connection string	"*ConnectionString*"," *database scheme*","*lookup table*","*lookup column"*

Door dit document op te stellen is het voor andere ontwikkelaars van Artena duidelijk geworden van de belangrijkste functionaliteiten zijn van de Artena Data Validator en hoe deze toegepast worden. Met behulp van dit document moet de gebruiker/ontwikkelaar dus bijvoorbeeld nieuwe validatieregels van een bepaald type kunnen toevoegen aan het systeem met behulp van de beheer interface uit §9.3.3.

Deel III – Evaluatie

10. Procesevaluatie

In dit hoofdstuk evalueer ik het, door mij, doorgelopen proces tijdens uitvoering van het project bij Artena. Ik zal verschillende belangrijke onderdelen van dit proces aanhalen waarover ik tevreden of minder tevreden ben en aangeven waarom dat zo is.

10.1 De productbeschrijving

In de eerste versie van het Plan van Aanpak had ik meerdere producten die opgeleverd zouden worden aan Artena afzonderlijk van elkaar beschreven. Zo was er het conversiesysteem, de validatie-engine, het log systeem en de monitoring tool. In de eerste maand ben ik bezig gegaan met het eerste product (de conversie-engine) zonder veel op de andere producten te letten. Na een maand werd het (mede na feedback van de begeleidende examinator) steeds duidelijker dat de verschillende producten een dermate grote samenhang hadden dat het invloed zou hebben op de uitvoer van het project.

Vanaf dat moment heb ik meerdere wijzigingen uitgevoerd aan de indeling van producten (deliverables) en heb ik de samenhang tussen deze producten in onder andere het productstroomschema van het Plan van Aanpak beschreven. Nadat deze wijzigingen uitgevoerd waren (tot en met Plan van Aanpak versie 1.6, bijlage A) werd het gehele project voor mij een stuk duidelijker en dus beter uit te voeren.

Deze wijzigingen hadden van te voren voorkomen kunnen worden door betere gesprekken te voeren met de opdrachtgever en projectbegeleider voor de aanvang van het project, waardoor de productbeschrijving vanaf het begin van het project veel bruikbaarder zou zijn.

10.2 Planning

Zoals eerder beschreven in §4.3 en §5.3 ben ik afgeweken van de planning door de vierde deliverable niet meer op te leveren. In overleg met stakeholder 1 en stakeholder 2 is dit afgesproken. De ‘core’ van Artena Data Validator was belangrijker dan de verwerking van de resultaten. Ik heb de tijd die is vrijgekomen van het vierde product goed kunnen gebruiken om de Artena Data Validator te verbeteren.

10.3 Gekozen fasering

De verschillende versies van het Plan van Aanpak hebben niet alleen toepassing op de indeling en beschrijving van de producten maar ook op de fasering. Ik had als uitgangspunt voor de fasering Agile genomen en daarbij had ik besloten meerdere kleine iteraties(één à twee weken) per product toe te passen.

Deze ‘snelle’ fasering bleek na anderhalve maand doorlooptijd echter nadelig te werken, vooral omdat ik het beheer van het gehele ontwikkeltraject zelf moest uitvoeren. Doordat ik bijna op wekelijkse basis

wijzigingen aan het bijhouden was in verschillende documenten begon ik teveel documentatie te schrijven waar ik vanuit het oogpunt van het project steeds minder aan had. Er ontstond dus overbodige documentatie.

Ik heb uiteindelijk besloten om mijzelf toch meer richting een RUP-achtige fasering te bewegen waarbij minder iteraties uitgevoerd worden en waarbij per iteratie een analyserapport, ontwerprapport en een eventueel constructierapport op te leveren. Zo zijn er uiteindelijk in totaal drie iteraties uitgevoerd waarin verschillende producten zijn behandeld.

Achteraf gezien denk ik dat ik met deze wijziging een goede keuze heb gemaakt, omdat ik door de meer gestructureerde aanpak mijzelf meer heb kunnen focussen op het project en de producten. Dit heeft uiteindelijk geleid tot goede resultaten (zie hoofdstuk 11) en een duidelijke beschrijving van de verschillende analyse, ontwerp –en constructiestappen die ik heb gemaakt.

Ik ben zeer tevreden met de incrementele aard van het ontwikkelproces waarmee ik de Artena Data Validator in drie iteraties stuk voor stuk heb kunnen opbouwen. Dit is terug te zien door eerst de constructie van het conversiesysteem (zie hoofdstuk 7) te hebben uitgevoerd, vervolgens door de validatie-engine te maken(zie hoofdstuk 8) en tot slot door die twee subsystemen samen te voegen en een log systeem toe te voegen (zie hoofdstuk 9). Deze manier van ontwikkelen is naar mijn mening zeker bij dit soort projecten een juiste keuze.

10.4 Requirements

Zoals is beschreven in de analyserapporten van de eerste en de tweede iteratie (zie bijlage E en bijlage J) is de structuur en formulering van de requirements na 2 maanden significant veranderd. Na feedback van de begeleidende examinator werd duidelijk dat de requirements, hoewel de inhoud van de eisen goed overlegd was met de stakeholders, niet SMART genoeg geformuleerd waren.

Ik had veel requirements op een te makkelijke manier geformuleerd waardoor er vooral niet-specifieke gegevens aanwezig waren. Toen ik dat te horen kreeg besefte ik dat ik inderdaad op een aantal momenten de door mij beschreven eisen mondeling moest specificeren aan verschillende stakeholders voordat deze volledig begrepen waren. De feedback was dus gepast en daar ben ik ook zeer blij mee geweest. Ik heb op basis daarvan alle requirements gecontroleerd en gewijzigd om ze vervolgens om te zetten naar een nieuwe structuur (zie de tabellen in §7.1.1, §8.1.1, §9.1.1).

Over de nieuwe structuur ben ik zeer tevreden omdat:

1. De requirements zijn veel ‘technischer’ vastgelegd
2. Vanuit bepaalde requirements wordt doorverwezen naar andere requirements
3. Stakeholders en prioriteiten zijn veel duidelijker aangegeven

De stakeholders waren zelf ook prettig verrast met de nieuwe notatie en structuur. Met deze requirements zijn de wensen en belangen dus voor zowel de stakeholders als de ontwikkelaar duidelijker geworden.

10.5 Ontwerpproces

Over het door mij toegepaste ontwerpproces tijdens de verschillende iteraties ben ik heel tevreden. Dit komt vooral door de keuze om eerst de databaseconcepten te ontwerpen en van daaruit de programmatuur te ontwerpen:

- Zoals te zien bij het databaseontwerp van het conversiesysteem(§7.2.2) en het databaseontwerp van de validatie-engine(§8.2.3) heb ik gebruik gemaakt van EER diagrammen, een RRM en een RIM om de database van de Artena Data Validator te definiëren.

- De programmatuur van de Artena Data Validator is gebaseerd op de database en de entiteiten die daarin aanwezig zijn. Door gebruik te maken van een O/R mapper (zie §7.2.5) heb ik op een zeer gemakkelijke wijze de verschillende entiteiten kunnen gebruiken in de programmatuur en heb ik goed gebruik kunnen maken van de mogelijkheden die het .NET framework aanbiedt.

- De klassen die ik zelf had ontworpen en beschreven in de verschillende klassendiagrammen waren dan ook volledig gefocust op de verschillende processen die de Artena Data Validator uit zou moeten voeren.

Ik heb het ontwerpproces dus als goed ervaren en zou het bij een volgend project (uiteraard afhankelijk van het doel van dat project) weer toepassen.

11.Productevaluatie

In dit hoofdstuk evalueer ik de verschillende producten en tussenproducten. Per product zal ik goede en slechte punten aangeven en zal het nut van het product duidelijk worden.

11.1 Artena Data Validator

Het belangrijkste product van het project. Dit systeem is opgeleverd aan Artena om de gewenste functionaliteiten uit te voeren. Samengevat ben ik zeer tevreden met het product, omdat alle deliverables (1-3) beschreven in §4.3 zijn geïmplementeerd. Ook draagt het systeem bij aan de drie onderdelen van de doelstelling (§4.1) door overeen te komen met het verwachte resultaat(§4.2). Als losstaand systeem maakt het een geautomatiseerde validatie van de data van klantbestanden mogelijk waardoor de uiteindelijke kwaliteit van deze data verbeterd kan worden (door de klanten).

De mooiste feature van het systeem vind ik de geïmplementeerde flexibiliteit en uitbreidbaarheid. Door een plug-in systeem(§7.2.3 en §8.2.5) te implementeren in zowel het conversiesysteem als de validatie-engine is het mogelijk meerdere bestandsformaten en validatietypen te ondersteunen zonder dat het conversie –en validatieproces zelf gewijzigd hoeft te worden.

Daarnaast vind ik dat de toegepaste templatestructuur (in de database, zie §7.2.1, §7.2.2 en §7.2.3) veel voordelen heeft meegebracht. Door vast te stellen welke soorten bestanden het systeem (en dus ook andere systemen van Artena) kan verwachten en de structuur daarvan vast te leggen in templates is een zeer flexibel controlesysteem ontstaan waarin voor meerdere klanten ondersteuning aanwezig kan zijn.

Conversiesysteem

Het uitlezen van Excel bestanden is naar mijn mening goed geïmplementeerd door gebruik te maken van de Excel Reader for .NET bibliotheek. Het uitlezen van Excel bestanden gaat snel en foutloos.

Aan het uitlezen van CSV bestanden kunnen echter nog wat onderdelen verbeterd worden zoals ook de systeemtest aangaf (zie §9.4.3) en gaat dus nog niet helemaal naar mijn wens.

Validatie-engine

Het systeem van meerdere validatieniveaus dat wordt gehanteerd door de Artena Data Validator zorgt voor een goed beheersbaar systeem doordat alle soorten validatieregels hiërarchisch zijn ingedeeld. Dit systeem heeft veel duidelijkheid geschept in de mogelijke soorten validaties.

De mooiste functionaliteit van de validatie-engine vind ik de wijze waarop de validatieresultaten worden vastgelegd in de database. Doordat vanuit een resultaat altijd alle benodigde gegevens gevonden kunnen worden (via de relaties tussen tabellen in de database) kunnen deze resultaten door een rapportage-tool achteraf op elke willekeurige manier gestructureerd en beschreven worden.

11.2 Rule builder

Dit product is niet aangegeven in het Plan van Aanpak en behoort ook niet tot de scope van het project. Door deze interface(zie §9.3.3) te ontwikkelen heb ik het mijzelf echter veel makkelijker gemaakt om de templates en validatieregels te beheren. Deze gebruikersinterface kan tevens dienen als basis voor een veel uitgebreidere beheerinterface, mocht deze in toekomst bij Artena ontwikkeld worden.

11.3 Documenten

11.3.1 Systeemdocumentatie

De systeemdocumentatie (zie §9.5/bijlage P) is in het Engels geschreven en bevat informatie over de features (mogelijkheden) van het systeem en de wijze waarop de conversie –en validatieprocessen uitgevoerd kunnen worden. Dit document kan echter nog veel verder worden uitgebreid, bijvoorbeeld wanneer een rule builder gemaakt zou worden of nieuwe validatietypen toegevoegd zouden worden.

11.3.2 Analyserapporten

De analyserapporten verschaffen veel inzicht in de eisen en wensen van verschillende stakeholders met betrekking tot de functionaliteiten en de flexibiliteit van het systeem. Met behulp van de requirements en de opgestelde use cases had ik een goede basis om verschillende ontwerpdelen te ontwikkelen.

Daarnaast bleken de vastgelegde requirements goed te gebruiken om aan de stakeholders alle functionaliteiten van de Artena Data Validator te beschrijven bij de uitgevoerde demonstraties.

Achteraf gezien vind ik wel dat de drie analyserapporten veel overlapping hebben op het gebied van de requirements. Elk rapport bevat ook de requirements uit het vorige rapport. Ik had liever één analyserapport met meerdere versies gezien waarin de requirements voor alle subsystemen bijgehouden zouden zijn zodat ik minder tijd kwijt zou zijn aan het beheren van de analysedocumentaties. Dit heeft natuurlijk ook veel te maken met de samenhang tussen de verschillende subsystemen waar ik pas op een later moment rekening mee heb gehouden (zie §10.1).

11.3.3 Ontwerprapporten

Op de ontwerprapporten heb ik weinig aan te merken. Ik heb gebruik gemaakt van de door mij geleerde technieken tijdens mijn studie om ontwerpen op te stellen voor zowel databases als programmatuur. Daarbij bleken de EER diagrammen zeer functioneel om het gehele gegevensmodel duidelijk te krijgen en heb ik de UML Sequence diagrammen goed kunnen gebruiken om de conversie –en validatieprocessen te modelleren.

Andere diagrammen, zoals het validatieflow diagram(§8.2.2), waren voor zowel mijzelf als mijn projectbegeleider zeer behulpzaam om bepaalde requirements beter te begrijpen.

11.3.4 Testrapport

Het testrapport was vooral voor mijzelf belangrijk omdat ik met het rapport de aanwezigheid van alle vastgelegde functionele eisen uit het laatste analyserapport kon aantonen. Ik kon dus de conclusie trekken dat het systeem voldeed aan deze requirements wat ook mijn projectbegeleider goed beviel.

12.Beroepstaken

12.1 Uitvoeren analyse door definitie requirements (beroepstaak 1.4)

Volgens het afstudeerplan zou deze beroepstaak op niveau 4 gehaald worden tijdens de uitvoer van het project. Ik heb de beroepstaak naar mijn mening ook op niveau 4 gehaald. Ik heb in meerdere iteraties meerdere functionele –en niet functionele requirements (zie §7.1.1, §8.1.1, §9.1.1) opgesteld door gesprekken te voeren met de stakeholders. +/-90% van de requirements zijn afkomstig van stakeholder 2 omdat hij de probleemeigenaar was en de aanleiding heeft gegeven tot ontwikkeling van het systeem.

Deze requirements heb ik vervolgens geprioriteerd volgens de MoSCoW methode door de lijst met eisen gezamenlijk met de stakeholder(s) langs te lopen en te vragen aan welke eisen de stakeholder de meeste waarde hechtte. De requirements zijn ook allemaal genummerd en dus traceerbaar, dit is onder andere terug te zien in het ontwerprapport waarbij ik terugverwijs naar de beschreven requirements. Ik heb ook uitgebreid versiebeheer uitgevoerd op de requirements zoals te zien is in de revisietabellen van de analyserapporten (bijlagen E, J, L).

Er zijn geen echte tegenstrijdige eisen ontstaan bij de gesprekken met de stakeholders, dit komt doordat Artena een klein bedrijf is en de stakeholders onderling bepaalde verschillen met elkaar kunnen rechttrekken en de eisen. Echter heeft de uitgevoerde analyse (door de requirements en UML use cases

te definiëren) er wel toe geleid dat het beeld met betrekking tot het valideren van verschillende soorten bestanden op verschillende niveaus met verschillende soorten validatieregels voor alle stakeholders en mijzelf zeer duidelijk en begrijpelijk is geworden. Dit is vooral beschreven in de definities van de verschillende invoeren en uitvoeren van de subsystemen van de Artena Data Validator.

12.2 Opstellen gegevensmodel voor database (beroepstaak 2.1)

Naar mijn mening heb ik deze beroepstaak gehaald op het vooraf gedefinieerde niveau 3. Voor de Artena Data Validator is een gegevensmodel ontwikkeld waarin een aantal belangrijke gegevens vastgelegd worden:

- Klanten(§7.2.2)
- Templatestructuur(§7.2.2): Bestandstemplates, Tabel templates, Kolom templates
- Importstructuur(§7.2.2): Import, Importbestanden, Importregels
- Validatiestructuren(§8.2.1): Validatietypen, Validatieregels, Validatieresultaten
- Systeemfouten(§9.2.2)

Het gegevensmodel is ontwikkeld door (Enhanced) Entity Relationship diagrammen te maken met behulp van de tool DIA. In het gegevensmodel is er sprake van meerdere 1-1 of 1-n relaties waarin bepaalde entiteiten aan elkaar gekoppeld zijn. Het gegevensmodel behoorlijk complex wat betreft de vele relaties die entiteiten met elkaar (kunnen) hebben. Een validatieresultaat heeft bijvoorbeeld 1 verplichte relatie en twee mogelijke relaties met andere entiteiten.

Op sommige punten in het model is zorgen de relaties voor de noodzakelijke implementatie van triggers(zie §12.3) om de multiple cascade paths in het gegevensmodel te ondersteunen.

12.3 Ontwerpen bouwen en bevragen van een database (beroepstaak 2.2)

Volgens het afstudeerplan zou deze beroepstaak op niveau 3 bewezen worden. Ik vind dat ik dat ook zeker heb gedaan. Het gegevens model van §12.2 heb ik omgezet naar een relationele database is het Microsoft SQL Server 2008 DBMS door gebruik te maken van de SQL Server Management Studio en het Relationeel Implementatie Model (§7.2.2, §8.2.3).

De programmatuur van de Artena Data Validator maakt uitvoerig gebruik van de geïmplementeerde database. Zoals eerder beschreven (§7.2.5) worden de tabellen uit de gemaakte database via een O/R mapper (LINQ-to-SQL) omgezet naar objecten in C# en zijn daardoor direct aan te spreken in de code.

De programmatuur voert nieuwe importbestanden en importregels toe aan de database na een conversie te hebben uitgevoerd en maakt gebruik van de vastgelegde templates en validatieregels om de conversies en validaties uit te voeren.

Tot slot wordt de referentiele integriteit van de database gewaarborgd door het toepassen van (twee) ON DELETE CASCADE relaties en de door mijzelf geïmplementeerde Triggers(§7.2.2) om het probleem met multiple cascade paths(§7.2.2) te omzeilen.

12.4 Uitvoeren gegevensconversie (beroepstaak 2.3)

In het afstudeerplan is beschreven dat deze beroepstaak op niveau 3 gehaald zou worden. Ik ben van mening dat dit door mij wel bewezen, hoewel niet op de punten zoals deze zijn beschreven in de definitie van deze beroepstaak.

De Artena Data Validator bevat een conversiesysteem dat verschillende soorten bestanden inleest en transformeert naar één structuur. Er is dus data van verschillende databronnen die op verschillende manieren wordt ingelezen door meerdere plug-ins. Om dit te bewerkstelligen heb ik eerst een analyse gemaakt van de structuur die de aangeleverde bestanden kunnen hebben en de structuur waar deze bestanden naartoe geschreven worden door het conversiesysteem (zie de requirements in §7.1.1 en de templatestructuur in §7.2.1).

De data prepareren voor verdere verwerking wordt ook wel *staging*(zie ook §4.1 - §4.2) genoemd en past bijvoorbeeld verrijking toe om de datakwaliteit te verbeteren. Dit is ook logisch aangezien juist die kwaliteit vastgesteld zal worden door de Artena Data Validator. De transformaties die uitgevoerd worden zijn volledig gefocust op het verkrijgen van één structuur waarop verschillende validatieregels toegepast kunnen worden. Enkele manipulaties (zoals verwijderen van duizendtal separatoren en veranderen van datum formaat) hebben alleen tot doel om de structuur van de Excel97 en Excel2007 bestanden gelijk te trekken.

Er is dus in mijn ogen wel degelijk sprake van een gegevensconversie, echter komt deze beschrijving niet volledig overeen met de definities zoals beschreven in de beroepstaak.

12.5 Ontwerpen systeemdeel (beroepstaak 3.2)

In het afstudeerplan is deze beroepstaak genoteerd om te halen op niveau 3. Ik ben echter van mening dat ik deze beroepstaak ook op niveau 4 heb bewezen. Zo heb ik bij het ontwerpen van de verschillende systeemdelen gebruik gemaakt van UML klassendiagrammen (op domeinniveau, onder andere figuur 14, 24 en 27) om de structuur van de programmatuur te beschrijven en van UML sequence diagrammen om het gedrag van de programmatuur te beschrijven op een aantal aspecten. Deze diagrammen heb ik opgesteld met de tool Astah Community Edition.

Vanuit alle ontwerprapporten wordt verwezen naar de functionele –en niet functionele eisen uit de bijbehorende analyserapporten zodat aangetoond kan worden dat de requirements verwerkt zijn in het ontwerp.

Naast de toegepaste object oriëntatie in de programmatuur is zeer veel rekening gehouden met flexibiliteit. Dit is vooral terug te zien in de implementatie (2x) van een plug-in systeem waarbij veel rekening is gehouden met abstractie en modularisatie. Zo is elke plug-in (verantwoordelijk voor het uitlezen van een bestand of het uitvoeren van een validatieregel) een afgeleide van een abstracte klasse(§7.2.3, §8.2.5) en een op zichzelf staande 'module' die kan worden ingeladen door de Artena Data Validator.

Tot slot heb ik in het ontwerp een aantal design patterns toegepast waarmee op een goede manier bepaalde problemen zijn opgelost. Hierbij springt vooral het log systeem eruit waarin een Singleton en Observer pattern zijn gecombineerd (§9.2.3) tot een flexibele logger.

Samengevat heeft het ontwerp van de verschillende systeemdelen geleid tot grote flexibiliteit en uitbreidbaarheid in de Artena Data Validator op meerdere aspecten van de verschillende subsystemen.

12.6 Bouwen applicatie (beroepstaak 3.3)

Het in het afstudeerplan gedefinieerde niveau 4 is voor deze beroepstaak is door mij gehaald. De gehele Artena Data Validator, inclusief conversiesysteem, validatie-engine, (twee) bestandslezer plug-ins en (tien) validatie plug-ins zijn vanuit het ontwerp geïmplementeerd in Microsoft C#.

Bij deze implementatie is gebruik gemaakt van de mogelijkheden van het .NET framework waaronder de implementatie van verschillende dataobjecten (zoals een *Dictionary*) en het gebruik van LINQ-to-SQL voor de mapping van database entiteiten naar objecten.

Tevens is gebruik gemaakt van een versiebeheer tool, namelijk Visual Studio Team Edition, die vanuit de gebruikte ontwikkelomgeving, Visual Studio 2010, wordt aangesproken om de programmatuur te beheren.

Zoals beschreven in dit document is de applicatie stuk voor stuk geprogrammeerd en in de laatste iteratie samengevoegd tot de Artena Data Validator. Het systeem is als DLL in te laden door een willekeurige client applicatie. De verschillende plug-ins zijn allemaal ontwikkeld als losse softwarebibliotheken en zijn geïntegreerd in het gehele project (zie onder andere figuur 13, §7.3).

13.Gebruikte literatuur

[Janssen, P. & Van Craen, J.P.] (2011). **Projectmanagement volgens PRINCE2, 3e editie.** Amsterdam: Pearson Education

[Pol, M. & Teunissen, R. & Van Veen, E.] (2002). **Testen volgens TMAP, 5e druk.** 's Hertogenbosch: Uitgeverij Tutein Nolthenius

[Tré, G.] (2007). **Principes van Databases.** Amsterdam: Pearson Education

[Warmer, J. & Kleppe, A.] (2007). **Praktisch UML, 4e editie.** Amsterdam: Pearson Education

[Shalloway, A. & Trott, J.R.](2005). **Design Patterns Explained.** Boston: Pearson Education

14.Woordenlijst

Begrip	Beschrijving
Abstracte klasse	Een klasse waarin minimaal één methode abstract is. Dit houdt in dat is aangeven hoe de methode eruit ziet, maar niet wat hij doet (de implementatie). Elke klasse die overerft van de abstracte klasse moet de abstracte methode altijd implementeren
DataContext	De representatie van een LINQ-to-SQL class. Van dit object kunnen meerdere instanties aangemaakt worden zodat de applicatie met de betrokken database kan communiceren.
Deadlock	Een toestand waarin twee processen op elkaar wachten om toegang te krijgen tot een object of een gegeven waardoor het systeem vastloopt.
Dictionary	Een .NET object waarin een key-object en een value-object aan elkaar gekoppeld kunnen worden. Heeft overeenkomsten met het HashMap object van Java.
File Check Systeem	Een systeem dat één of meerdere locaties continu scant of pollt en bij upload van een nieuw bestand op één van deze locaties een actie uitvoert.
Instantie	Een nieuwe kopie van een object dat gebruikt kan worden door een systeem, object of methode.
Lookup values	Lookup values zijn waarden die in een database zijn opgeslagen waaraan gerefereerd kan worden vanuit een willekeurig systeem. Daarmee kan een controle worden uitgevoerd of bepaalde gegevens voldoen aan deze waarden.
Multiple cascade paths	Meerdere paden waar vanuit een gegeven in tabel in een database verwijderd kan worden door het handhaven van referentiele integriteit.
.NET Framework	Een software framework ontwikkeld door Microsoft. Het framework bevat een zeer uitgebreide bibliotheek met veel verschillende functionaliteiten en ondersteund meerdere programmeertalen zoals C# en Visual Basic.
Object-relational mapping	Techniek waarmee objecten worden omgezet naar/gemapped op relationele typen waardoor de objecten bruikbaar zijn in een virtuele database. Dit proces kan ook andersom gebruikt worden waardoor databasetabellen omgezet worden naar objecten zodat deze direct aangesproken kunnen worden in de programmatuur
Pensioenuitvoerder	Een instantie of bedrijf dat voor verschillende werkgevers de pensioenen van werknemers kan beheren
SSIS	Microsoft SQL Server Integration Services. Een platform voor data integratie waarmee verscheidene datamigratie taken uitgevoerd kunnen.
Staging	Het ‘klaarzetten’ van gegevens in een bepaalde omgeving (zoals een database) voor verdere verwerking van die gegevens.

15.Figuren

Afbeelding	Pagina
Figuur 1: Organogram Artena	8
Figuur 2: Iteratiemodel	15
Figuur 3: Softwareprocessen Fincircle	20
Figuur 4: SSIS TNO Importpackage	20
Figuur 5: Plaatsing Artena Data Validator	21
Figuur 6: Invoer/Uitvoer conversiesysteem	22
Figuur 7: Use case diagram conversiesysteem	25
Figuur 8: Invoer/Uitvoer conversiesysteem - herzien	26
Figuur 9: Gebruikersinterface testapplicatie	26
Figuur 10: Excel Data Reader for .NET	27
Figuur 11: Bestandsstructuur	28
Figuur 12: ERD conversiesysteem	30
Figuur 13: Abstracte klasse bestandslezer plug-in	32
Figuur 14: Domeinmodel conversiesysteem	33
Figuur 15: TableTemplate klasse LINQ-to-SQL	34
Figuur 16: Solution (MS Visual Studio 2010)	34
Figuur 17: LINQ-to-SQL classes conversiesysteem	35
Figuur 18: UML sequence diagram conversieproces	36
Figuur 19: Beschrijving validatieniveaus	38
Figuur 20: Use case diagram validatie-engine	41
Figuur 21: Validatieflow	44
Figuur 22: Validatieflow op field-niveau	44
Figuur 23: ERD validatie-engine	45
Figuur 24: Abstracte klasse validatie plug-in	47
Figuur 25: Domeinmodel validatie-engine	48
Figuur 26: UML sequence diagram validatieproces	49
Figuur 27: Use case diagram Artena Data Validator	52
Figuur 28: Klassendiagram Artena Data Validator - globaal	54
Figuur 29: UML sequence diagram importproces	55
Figuur 30: ERD log systeem	55
Figuur 31: Logger klasse – Singleton pattern	56
Figuur 32: Log systeem – Observer pattern	57
Figuur 33: UML activiteitsdiagram datatype validatie	59
Figuur 34: User interface 1 - client	60
Figuur 35: User interface 2 - beheer	60

16. Bijlagen

A. Plan van Aanpak **Versie 1.6**

B. Iteratiedocument

C. User story

D. Oriëntatie Artena & Systemen

E. Analyserapport Iteratie 1 **Versie 1.0**

F. Onderzoek bestandslezers **Versie 1.1**

G. Unit Test – Excel File Reader

H. Unit Test – CSV File Reader

I. Ontwerprapport Iteratie 1 **Versie 1.2**

J. Analyserapport Iteratie 2 **Versie 1.0**

K. Ontwerprapport Iteratie 2 **Versie 1.0**

L. Analyserapport Iteratie 3 **Versie 1.0**

M. Ontwerprapport Iteratie 3 **Versie 1.1**

N. Constructierapport Iteratie 3 **Versie 1.0**

O. Systeemtest (testplan, testscript, testrapport)

P. Systeemdocumentatie

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage A

Plan van Aanpak

Versie 1.6

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 03-04-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	12-01-2012	Concept	Initiële versie Plan van Aanpak. Toegevoegde onderdelen: - Opdrachtomschrijving - Deliverables - Proces georiënteerde producten - Projectinrichting - Planning Bijlagen: - A. Gewenste functionaliteiten deliverables - B. Planning
1.0	13-02-2012	Eerste versie	Wijzigingen: • Beschrijving iteraties • Toevoegingen tools (MiKTeX/LaTeX& MS SSIS & MS SQL Server) Planning is nu bijgevoegd in hoofdstuk 6.3
1.1	16-02-2012		Toegevoegd: • Product 1.2: ontwerp conversiesysteem • Bedrijfsorganisatie • Risico's Wijzigingen: • Inleiding aangepast • Stakeholders aangepast
1.2	20-02-2012		Toegevoegd: • **Samenvoegen deliverables in acceptatie / productie omgeving + wijziging in planning** • Beschrijving m.b.t. Rapid Prototyping bij uitvoeren onderzoek. • Beschrijving versiebeheer

1.3	21-02-2012		Op basis van feedback begeleider: • Logsysteem parallel aan bouw conversie en validatiesysteem. • Wijziging decompositie, stroomschema en planning, beschikbare tijd logsysteem omgezet naar conversie en validatie • Ingeplande tijd voor productie / acceptatie ook beschikbaar voor opvangen uitloop deliverables.
1.4	14-03-2012	Herziene versie	Op basis van feedback begeleidend examinator: • Nog meer parallellisatie van deliverables één, twee en drie. Deze uitvoer van veel activiteiten voor deze deliverables zullen samenkomen in dezelfde documenten. • Productstroomschema herzien. • Proces product 4 verwijderd (functioneel ontwerprapport 2) • Stakeholder 3 herzien. • Planning volledig herzien.
1.5	27-03-2012	Herziene versie	Op basis van feedback begeleidend examinator: • Textuele aanpassingen • Verwacht resultaat (§2.4) + flexibiliteit • Deliverable 5 → eindproduct 5 • Productdecompositie aangepast naar het gemeenschappelijke systeem waar de deliverables deel van uit maken, de Artena Data Validator. • Stroomschema aangepast aan productdecompositie • Stakeholders aangepast en stakeholder 3 gesplitst in twee groepen: FinCircle en Business Intelligence • Wijzigingen aan beschrijving fasering • Wijzigingen en versiebeheer samengevoegd, TortoiseSVN als tool • Planning gewijzigd, nu in tabel, overzichtelijker • Modelleringtechnieken voor databases en programmatuur toegevoegd
1.6	03-04-2012	Herziene versie	Planning en fasering is aangepast. Minder iteraties, maar wel grotere iteraties. Per iteratie wordt een andere deliverable behandeld. Planning is nu meer product georiënteerd. Planning is aangepast aan nieuwe iteratievorm. De vier oude 'iteraties' van het conversiesysteem worden samengevoegd tot één iteratie (van 3 weken).

Inhoudsopgave

1. Inleiding 5
2. Opdrachtomschrijving 6
2.1 Achtergrond en bedrijf 6
2.2 Probleemstelling 6
2.3 Doelstelling 7
2.4 Resultaat 7
3. Producten 8
3.1 Deliverables 8
3.2 Productdecompositie 9
3.3 Product stroomschema 10
3.4 Oplevering 11
3.5 Proces georiënteerde producten 11
5. Organisatie 13
5.1 Bedrijfsorganisatie 13
5.2 Stakeholders 14
6. Projectinrichting 16
6.1 Fasering 16
6.2 Wijzigingen & Versiebeheer 19
6.3 Tools 20
7. Risicomanagement 21
7.1 Tijd 21
7.2 Onduidelijke afspraken 21
7.3 Beschikbaarheid stakeholders 21
7.4 Oplevering producten 21
8. Planning 22
9. Begrippenlijst 24
10. Literatuur 24

1. Inleiding

Dit verslag behandelt de opzet waarin het afstudeerproject bij Artena uitgevoerd zal worden. Dit in het kader van de afstudeeropdracht die uitgevoerd moet worden voor de opleiding Informatica aan de Haagse Hogeschool.

Eerst zal de herhaling van de opdrachtomschrijving worden gedaan, op basis van de beschreven opdracht in het *Afstudeerplan v1.4*. **{Hoofdstuk 2}**

Op basis van deze omschrijving wordt een aantal producten uiteengezet, waarbij een tweetal technieken uit Prince2 worden toegepast: De productdecompositie en het product stroomschema. Door het toepassen van deze twee technieken wordt meer inzicht en logica gecreëerd in de op te leveren producten. Ook wordt onderscheid gemaakt tussen deliverables en proces georiënteerde producten. **{Hoofdstuk 3}**

Dit document beschrijft ook de organisatie van het bedrijf Artena: 'Welke afdelingen heeft het bedrijf en welke relatie hebben deze afdelingen onderling?'. De stakeholders/belanghebbenden/betrokkenen bij dit project worden ook uiteengezet. **{Hoofdstuk 4}**

Vervolgens wordt de projectinrichting besproken **{Hoofdstuk 5}**:

- Hoe gaat het project uitgevoerd worden?
- Welke fasering gaat gebruikt worden?
- Hoe wordt omgegaan met wijzigingen?
- Welke tools zullen gebruikt worden?

Na de projectinrichting komt risicobeheer van het project aan bod waarbij de aanwezige risico's beschreven zullen worden en waarbij maatregelen worden beschreven indien een risico zich voordoet. **{Hoofdstuk 6}**

Tot slot zal een planning worden beschreven waarbij aangegeven wordt op welke momenten welke activiteiten uitgevoerd zullen worden voor bepaalde producten. Ook zal deze planning duidelijk maken wat de verwachte opleverdata zijn voor de deliverables. **{Hoofdstuk 7}**

14-03-2012, Herziening Plan van Aanpak

Het document is op een aantal punten herzien op basis van het bezoek van de begeleidende examinator, de heer M. Reijnhoudt. De manier waarop de producten opgesplitst waren bleek belemmerend te zijn bij het opstellen van bijvoorbeeld analysedocumenten of ontwerpdocumenten. De scheiding in deliverables is nog steeds aanwezig, maar er is aangegeven dat deze scheiding vooral op functioneel gebied is.

27-03-2012, Extra herziening

De deliverables 1-4 maken nu onderdeel uit van de Artena Data Validator. Elk van deze deliverables is een subsysteem van de Artena Data Validator met elk een eigen verantwoordelijkheid.

2. Opdrachtomschrijving

2.1 Achtergrond en bedrijf

Artena is een groeiend consultancy/software bedrijf dat gericht is op het leveren van advies en IT oplossingen aan banken, pensioenfondsen, verzekeraars en dienstverlenende bedrijven. Complexe business kennis wordt daarbij gecombineerd met bedrijfskundige en financiële IT-toepassingen. Software producten die daarbij geleverd worden zijn:

1. **Financial Studio** (Business Intelligence): een uitgebreid datawarehouse dat specifiek is ontworpen voor banken. Dit datawarehouse is een volledige Microsoft toepassing dat gebruik maakt van de nieuwste business intelligence componenten (SQL-Server, OLAP server) (www.finarch.com)
2. **Cerrix** (Governance, Risk & Compliance): Een administratiesysteem om operationele bedrijfsrisico's en compliance risico's te beheersen. In dit systeem kunnen bijvoorbeeld processchema's ontwikkeld worden. (www.cerrix.com)
3. **FinCircle** (Pensioenoplossingen): Een applicatiesuite voor pensioenuitvoerders. Het pakket bevat softwaremodules voor pensioensimulatie, pensioenadministratie en communicatie. Diverse pensioenfondsen en bedrijven maken gebruik van deze applicatie.

De strategie van Artena omvat het succesvol uitbouwen van de markten voor deze producten, het implementeren van de producten, het doorvoeren van specifieke aanpassingen (feature requests) en het verbeteren/uitbreiden van de producten. Enkele opdrachtgevers van Artena: NIBC, Van Lanschot Bankiers, Zorg & Zekerheid en de pensioenfondsen van onder andere Ahold, TNO en de ANWB.

De opdracht zal worden uitgevoerd op de R&D afdeling van Artena. Hier werken 10 ontwikkelaars, die continu bezig zijn met bijvoorbeeld het ontwikkelen en testen van nieuwe features.

2.2 Probleemstelling

Artena biedt sinds enige tijd de applicatie FinCircle aan voor *pensioenuitvoerders*. Deze applicatie bestaat uit een administratiesysteem, een portal voor werkgevers en een portal voor werknemers. Een aantal klanten maakt alleen gebruik van het werknemersportal.

Belangrijkste functionaliteiten werknemersportal:

- Biedt werknemers inzicht in de manier waarop ze bij de pensioenuitvoerder geregistreerd staan
- Werknemers hebben de mogelijkheid hun pensioenprognose te bekijken
- Werknemers kunnen verschillende scenario's laten doorrekenen (met behulp van rekenmodellen ontwikkeld door consultants van Artena)

De werknemerportalen maken gebruik van bestanden die Artena regelmatig krijgt aangeleverd van de pensioenuitvoerders. Deze bestanden worden na aanlevering ingelezen in het werknemersportal door middel van een door Artena gebouwd verwerkingssysteem.

Het inlezen van bestanden is momenteel *foutgevoelig* en daardoor nog erg afhankelijk van handmatige controles. Voorbeelden van de belangrijkste problemen (of 'fouten') die momenteel spelen zijn:

- Het bestandsformaat (bijvoorbeeld Excel) van het aangeleverde bestand is incorrect

- Gegevensbestanden die qua datalayout niet voldoen aan wat verwacht wordt, bijvoorbeeld een verkeerd aantal kolommen, verkeerde kolomnamen en verkeerde datatypes

- Ontbrekende data: een bestand wat opeens maar de helft van het verwachte/benodigde aantal regels bevat (het verwachte aantal staat geregistreerd in het portal)

- Incorrecte *lookup values*: bijvoorbeeld Man/Vrouw in plaats van M/V

- Datumtijd/valuta notaties: verkeerde cel eigenschappen of verkeerde taalinstellingen, decimaal –en duizendtalseparators, verkeerde weergave van datum en tijd.

- Verkeerde bestandformaten: Excel 1997-2003 versus Excel 2007 of een andere/verkeerde interpretatie van het CSV formaat (wel of niet afsluiten met een puntkomma).

Naast bovenstaande (meest voorkomende) fouten speelt kunnen er problemen ontstaan doordat de gegevens onwaarschijnlijk zijn. Er moet dus worden gekeken naar de 'waarschijnlijkheid' van gegevens (bijvoorbeeld een pensioenbijdrage van 1.000.000 euro).

2.3 Doelstelling

De doelstelling van deze opdracht bestaat uit drie delen:

- Het **verhogen** van de *efficiëntie* van het proces van gegevenstransfer tussen Artena en pensioenuitvoerders voor het aanleveren, controleren en doorgeven van gegevens voor het werknemersportal (en in de toekomst voor andere data uitwisselingen)
- Het **verbeteren** van de *kwaliteit* van aangeleverde data door het toepassen van inhoudelijke controle op aangeleverde data (zie §2.2)
- Het **verbeteren** van de dienstverlening naar de klanten toe door de mogelijkheid te bieden aangeleverde data geautomatiseerd te kunnen verwerken waardoor data dus ook vaker aangeleverd kan worden door klanten.

2.4 Resultaat

Een geautomatiseerd systeem wat ondersteuning biedt voor de data uitwisseling tussen klanten van Artena en de uiteindelijke applicatie waarvoor de data bedoeld is. Dit geautomatiseerde systeem moet handmatige interventie overbodig maken en zal een verbetering opleveren in de dienstverlening naar klanten toe dankzij een betere inhoudelijke datacontrole en snellere verwerking. Verwachte functionaliteiten systeem (volledige requirements worden tijdens de opdracht vastgelegd):

- Converteren aangeleverde bestanden naar één bestandsformaat
- Valideren/Verifiëren gegevens in aangeleverde bestanden
- Loggen resultaten van validatie in een database
- Rapportage van fouten naar de klant (pensioenuitvoerder) op het moment dat problemen geconstateerd worden.
- Flexibiliteit, vooral m.b.t. uitbreiden bestandsondersteuning en validatieregels waardoor ook een eventueel toekomstig gebruik van dit systeem bij andere systemen van Artena mogelijk is.

3. Producten

In dit hoofdstuk zijn de producten beschreven die opgeleverd zullen worden aan Artena en de Haagse Hogeschool. Producten opgeleverd aan Artena worden gezien als *deliverables* of eindproducten, de planning zal ook gebaseerd zijn op dit typen producten. Naast de deliverables(§3.1) worden ook een aantal *proces georiënteerde* producten(§ 3.5) opgeleverd, bedoeld voor de Haagse Hogeschool.

3.1 Deliverables

De deliverables zijn gebaseerd op het beschreven verwachte resultaat (§2.4). Deze deliverables zijn verschillen van elkaar in functioneel opzicht maar hebben een sterke samenhang aangezien ze behoren tot hetzelfde systeem. Dit systeem zal in het vervolg de Artena Data Validator genoemd worden. Een belangrijk kenmerk van dit systeem zal flexibiliteit zijn, het toevoegen of wijzigingen van de aanwezige ondersteuning voor bepaalde bestanden van huidige/nieuwe klanten moeten geen ingrijpende, tijdrovende en dure wijzigingen veroorzaken.

Deliverable 1: Conversiesysteem

Met behulp van dit product moeten de aangeleverde klantbestanden door klanten van Artena worden geconverteerd worden naar één bestandsformaat, zodat ze doorgevoerd kunnen worden aan product 2. Manipulaties aan de inhoud van deze bestanden moeten zo min mogelijk voorkomen om zo de originele inhoud van het bestand te kunnen controleren. Gewenste flexibiliteit in dit systeem: het toevoegen van ondersteuning van nieuwe bestandsformaten of bestandsindelingen moet geen grote wijzigingen veroorzaken en gemakkelijk kunnen.

Deliverable 2: Validatie-engine

De validatie-engine is verantwoordelijk voor het valideren van het de inhoud van de uitvoer van deliverable 1 (het conversiesysteem). De validatie bestaat in ieder geval uit meerdere validatietypen, zoals beschreven in §2.2, en nieuwe validatieregels moeten op een eenvoudige manier toegevoegd kunnen worden aan het systeem. De validatieresultaten van verschillende uitgevoerde validaties zullen ook vastgelegd moeten worden vanuit dit systeem.

Deliverable 3: Log systeem

Het log systeem is een registratiemiddel wat verantwoordelijk is voor het loggen van systeemfouten die kunnen voorkomen in de uitvoeringsprocessen van de Artena Data Validator. Dit systeem zal dus geïntegreerd moeten worden in zowel deliverable 1 als deliverable 2. Logs zullen in ieder geval vastgelegd moeten worden in een nog te ontwerpen database.

Deliverable 4: Monitoring-tool

De monitoring-tool is verantwoordelijk voor de rapportage van het validatieproces aan de klanten van Artena. Deze deliverable zal de resultaten van validaties (vastgelegd door het log systeem) moeten ophalen en structuren. Hoe deze rapportage geïmplementeerd gaat worden moet nog worden vastgesteld door middel van een onderzoek. Requirements voor dit product moeten dus nog worden vastgelegd.

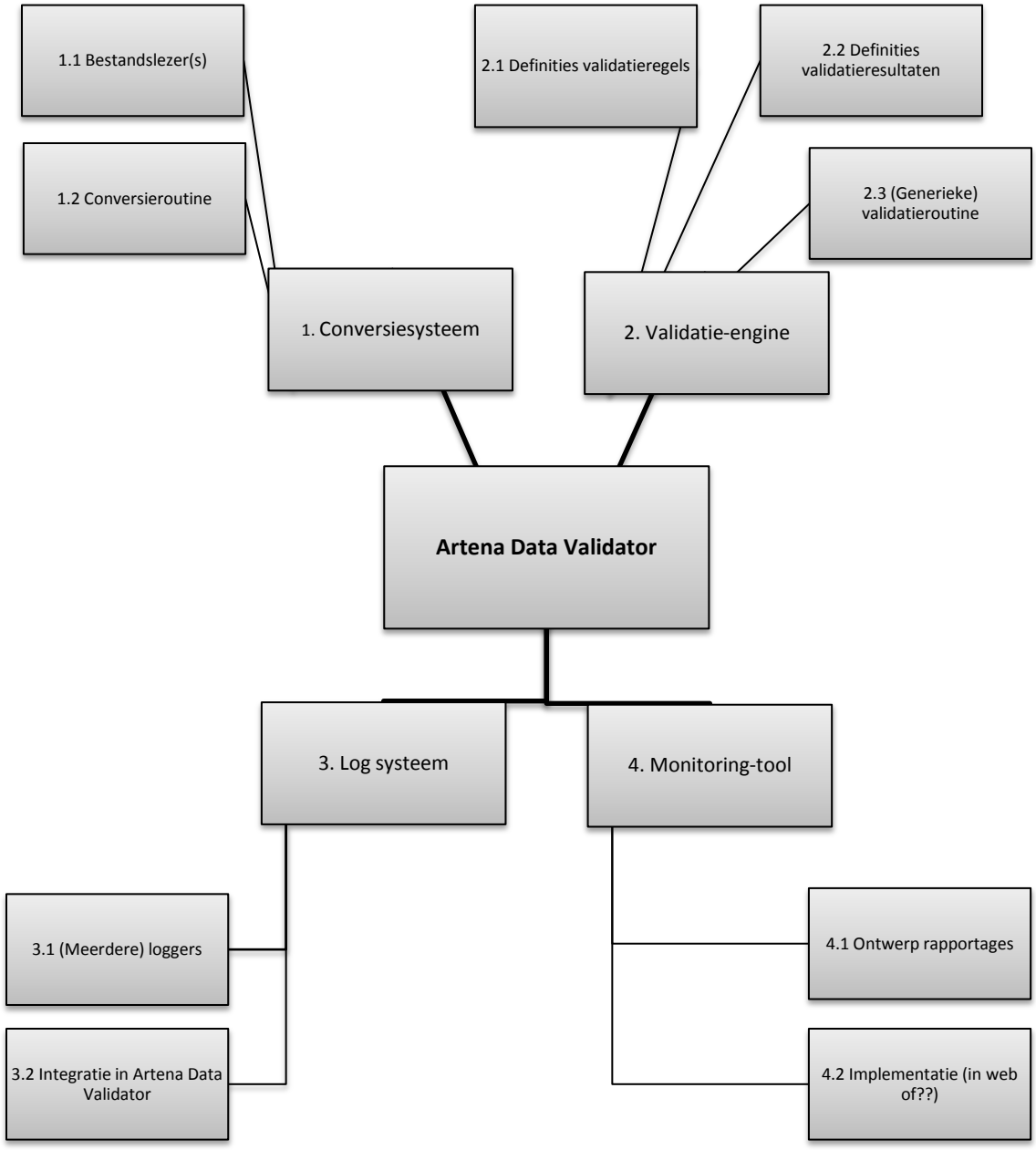
Deliverable 5: Systeemdocumentatie

De systeemdocumentatie bevat in ieder geval technische specificatie waarin alle features tot in detail

zijn beschreven. Hierbij geldt wederom dat overleg gepleegd moet worden voor de daadwerkelijke eisen voor de systeemdocumentatie. De systeemdocumentatie zal in het Engels geschreven moeten worden.

3.2 Productdecompositie

Het model op de volgende pagina toont de samenstelling van de beschreven deliverables uit §3.1. Elke deliverable hierin beschreven behoort tot de Artena Data Validator. Elke deliverable bestaat uit enkele onderdelen (subsystemen) die bepaald taken vervullen. Het model toont een globaal overzicht voor deze onderdelen. D [Janssen, P. & Van Craen, J.P.] (2011).



Figuur 1: Productdecompositie

3.3 Product stroomschema

De deliverables en hun subsystemen uit de productdecompositie in §3.2 zijn afhankelijk van elkaar. Sommige producten hebben andere producten nodig om te kunnen functioneren of om ze überhaupt de produceren. Het stroomschema hieronder toont de relatie tussen de producten uit de decompositie. De gegevens uit dit model kunnen gebruikt worden voor het opstellen van een planning voor het project. Het model voor dit schema maakt ook onderdeel uit van PRINCE2 [Janssen, P. & Van Craen, J.P.] (2011).

3.4 Oplevering

Elke deliverable (1-4) is onderdeel van het systeem (Artena Data Validator) dat beschreven is §2.4. Wanneer deze deliverables af zijn, dan zullen de subsystemen met elkaar moeten communiceren en samenwerken in de Artena Data Validator.

Het is de bedoeling dat de Artena Data Validator (inclusief de monitoring-tool) kan worden overgedragen aan Artena (een overdraagbaar resultaat). Hiermee wordt bedoeld dat de ontwikkelaars van Artena genoeg kennis hebben om verder te kunnen werken aan de Artena Data Validator. Het is dus belangrijk dat het ontwerp en de functionaliteiten goed zijn gedocumenteerd.

3.5 Proces georiënteerde producten

Naast de deliverables moeten ook een aantal andere producten worden opgeleverd, die ook bedoeld zijn voor de Haagse Hogeschool. Deze producten zijn van belang voor de Haagse Hogeschool omdat zij inzicht in het afgelegde afstudeertraject en de daarbinnen uitgevoerde activiteiten voor Artena. Daarnaast leveren deze producten een basis voor het uitvoeren van kwaliteitscontrole op de opgeleverde producten: opgestelde requirements kunnen bijvoorbeeld worden gebruikt als referentie voor een systeemtest en vanuit het ontwerpdocument kan worden vastgesteld op welke wijze bepaalde eisen zijn verwerkt.

De meeste proces georiënteerde producten zijn samengesteld uit verschillende onderdelen die gemaakt worden tijdens het doorlopen van de verschillende iteraties (zie H5).

Proces product 1: Plan van Aanpak

Het document waarin de opzet van uitvoer voor het project bij Artena beschreven is. In dit document wordt onder andere een gedetailleerde beschrijving gemaakt van de uit te voeren activiteiten en op te leveren producten.

Bedoeld voor de afstudeerder en de schoolinstelling (De Haagse Hogeschool).

Proces product 2: Analyserapport Artena Data Validator

Dit rapport is opgebouwd uit de analyseproducten van de deliverables. Het rapport bevat de laatste versies van:

1. Functionele en niet-functionele requirements
2. Toegekende prioriteiten aan requirements
3. UML use case diagram(men)
4. UML use case beschrijvingen

Bedoeld voor Artena en de Haagse Hogeschool

Proces product 3: Ontwerprapport Artena Data Validator

Dit rapport bevat het ontwerp van alle deliverables (die samen de Artena Data Validator vormen). Het document zal per deliverable meerdere onderdelen bevatten die het ontwerp van het systeem beschrijven. In dit document zal worden verwezen naar de vastgelegde requirements in proces product 2 ‘Analyserapport’. Dit document bevat onder andere:

- Databaseontwerp met behulp van onder andere EER diagrammen
- UML domein klassendiagrammen.
- UML sequence diagrammen

Bedoeld voor Artena en de Haagse Hogeschool

Proces product 4: Testrapport Artena Data Validator

Wanneer de Artena Data Validator klaar voor oplevering lijkt te zijn, zal een systeemtest worden uitgevoerd waarbij gekeken zal worden of het product aan de gestelde eisen (functioneel) voldoet.

Bedoeld voor Artena en de Haagse Hogeschool

Proces product 5: Voortgangsrapport

Beschrijft de voortgang die de afstudeerder heeft gemaakt tijdens de uitvoer van het project, waarbij keuzes, problemen en oplossingen beschreven zijn.

Bedoeld voor de Haagse Hogeschool

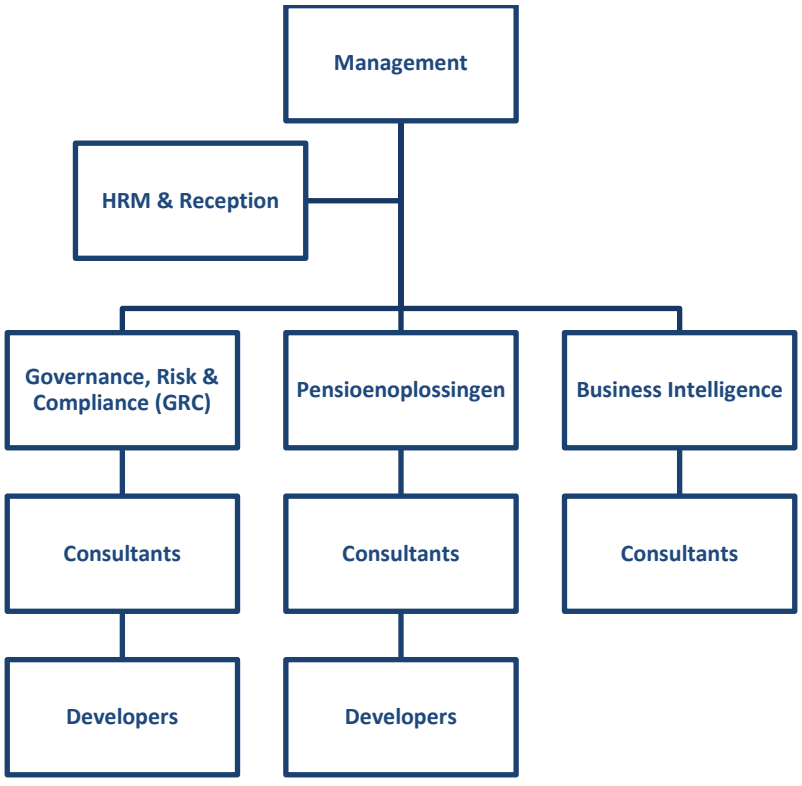
5. Organisatie

5.1 Bedrijfsorganisatie

Zoals beschreven is de core-business van Artena het leveren van advies en oplossingen aan onder andere banken, verzekeraars en pensioenoplossingen. Deze services zijn te splitsen in drie delen:

1. Adviseren financiële instellingen op het gebied van Governance, Risk en Compliance (GRC). GRC behelst ondersteuning op het gebied van Risk Management waarbij strategische scenario's opgezet worden en advies wordt gegeven over wetten en regelgeving. Daarbij levert Artena software voor ondersteuning van deze activiteiten: CERRIX[1]
2. Het leveren van een compleet softwarepakket voor onder andere pensioenfondsen en levensverzekeraars, FinCircle. Ook heeft Artena consultants in dienst voor advies op het gebied van pensioencommunicatie, pensioenadministratie en documentmanagement.
3. Business Intelligence; het adviseren van financiële instellingen op het gebied van Data Warehousing en Reporting. Hiervoor heeft Artena een aantal professionals in dienst met ervaring in het optimaliseren van data warehouses en reporting services.

Voor de eerste twee onderdelen is een ontwikkelteam actief bezig met het onderhouden, verbeteren en uitbreiden van software. De requirements voor deze activiteiten en producten worden aangeleverd door de consultants van Artena. Het organogram van Figuur 2 toont de globale bedrijfsorganisatie.



Figuur 2: Organogram Artena

[1] http://www.cerrix.com/

5.2 Stakeholders

Bij het uit te voeren project zijn meerdere stakeholders betrokken. Dit zijn de personen of bedrijven die belangen hebben bij of eisen hebben aan het project. Sommige stakeholders zijn alleen betrokken bij bepaalde delen van het project. Hieronder staat een overzicht van de betrokken stakeholders met een korte beschrijving en de producten waarbij de stakeholder betrokken bij is.

Stakeholder 1: Paul Bruggeman, Managing Director Artena
Het belang van deze stakeholder is dat de deliverables van het project aan de vastgestelde eisen voldoen en binnen de gestelde periode (17 weken) wordt opgeleverd. Deze stakeholder kan tijdens de uitvoer van het project nieuwe eisen stellen aan de deliverables op basis van nieuwe kennis of observaties.

- Betrokken deliverables: (1)Conversiesysteem, (2)Validatie-engine, (3)Log systeem, (4)Monitoring-tool

Stakeholder 2: Nico Hoekstra, Senior Developer
Deze persoon kan worden gezien als de belangrijkste stakeholder voor de Artena Data Validator. Deze stakeholders is degene die op dit moment de handmatige controles uitvoert op de aangeleverde bestanden en de resultaten van die controles rapporteert aan de klanten. Hij is tevens degene die samen met de eerste stakeholder gezien kan worden als de opdrachtgever.

De eisen die aan momenteel aan de deliverables zijn gesteld zijn van deze stakeholder afkomstig. Gedurende de opdracht kunnen er eisen wijzigen of nieuwe eisen bijkomen door nieuwe inzichten in het probleem, reviews van producten en gesprekken/vergaderingen met de afstudeerder.

Deze stakeholder zal ook de analysedocumenten, ontwerpdocumenten en programmatuur voor de Artena Data Validator evalueren.

- Betrokken deliverables: (1)Conversiesysteem, (2)Validatie-engine, (3)Log systeem, (4)Monitoring-tool, (5)Systeemdocumentatie

Stakeholder 3: Consultants & Developers FinCircle/Pensioenoplossingen
Deze stakeholder beschrijft de groep van ontwikkelaars die zich bezig houden met pensioenoplossingen (FinCircle) bij Artena. Stakeholder 2 is ook onderdeel van deze groep developers. Het is mogelijk dat een aantal developers in deze groep al ideeën/gedachten heeft over een validatie systeem (of monitoring-tool) en de implementatie daarvan. Het is mogelijk dat uit die ideeën/gedachten nieuwe eisen ontstaan voor de Artena Data Validator.

- Betrokken deliverables: (2)Validatie-engine, (3)Log systeem, (4)Monitoring-tool

Stakeholder 4: Consultants & Developers Bussiness Intelligence
Deze stakeholder beschrijft groep van consultants en developers die zich bezig houden met Business Intelligence voor klanten van Artena. In het verleden heeft deze groep al enkele projecten uitgevoerd voor klanten die te maken hebben gehad met kwaliteit van data (en het meten ervan).

Deze stakeholder heeft om deze reden misschien andere ideeën/gedachten bij een validatiesysteem of over het eventuele gebruik van een dergelijk systeem voor andere systemen ontwikkelt/gebruikt door Artena (flexibiliteiteis, §2.4).

- Betrokken deliverables: (1)Conversiesysteem, (2)Validatie-engine

Stakeholder 5: Pensioen uitvoerders, klanten van Artena
Deze stakeholder bestaat uit de klanten van Artena, in dit geval de pensioenfondsen. Zij leveren de bestanden aan die gevalideerd zullen moeten worden. Aan deze personen moet ook de rapportage worden uitgevoerd. Hoe de rapportage (monitoring-tool) geïmplementeerd gaat worden is nog onduidelijk. Verwacht wordt dat deze stakeholders eisen/wensen hebben aan de manier van rapporteren van fouten.

- Betrokken deliverables: (4)Monitoring-tool

6. Projectinrichting

Dit hoofdstuk beschrijft de wijze waarop het project uitgevoerd gaat worden. Dit wil zeggen dat er een keuze wordt gemaakt voor de fasering en/of ontwikkelmethode waarmee de producten beschreven in hoofdstuk 3 geproduceerd en opgeleverd moeten worden.

Bij het selecteren van een fasering en/of ontwikkelmethode zijn onderstaande aspecten belangrijk:

- De vastgestelde tijd voor het project is 17 weken, van 6 februari 2012 tot 1 juni 2012.
- De kennis van de afstudeerder over infrastructuur en andere belangrijke onderdelen binnen Artena is nog beperkt. Deze kennis zal tijdens de uitvoer van het project moeten groeien.
- De producten die opgeleverd moeten worden zijn beschreven in de productdecompositie in §3.2.
- Het stroomschema uit §3.3 geeft de verwachte volgorde aan waarin producten opgeleverd zullen moeten worden
- Bij Artena wordt niet specifiek volgens één ontwikkelmethode gewerkt. Wel worden de twaalf principes[2] van Agile Software Development *in meer of mindere mate* toegepast:
    - Customer satisfaction by rapid delivery of useful software
    - Welcome changing requirements, even late in development
    - Working software is delivered frequently (weeks rather than months)
    - Working software is the principal measure of progress
    - Sustainable development, able to maintain a constant pace
    - Close, daily co-operation between business people and developers
    - Face-to-face conversation is the best form of communication (co-location)
    - Projects are built around motivated individuals, who should be trusted
    - Continuous attention to technical excellence and good design
    - Simplicity
    - Self-organizing teams
    - Regular adaptation to changing circumstances

6.1 Fasering

De beschikbare 17 weken zullen verdeeld worden (zie hoofdstuk 7, planning) in meerdere *timeboxes (=iteraties)* van een aantal weken waarin verschillende deliverables aan bod komen.

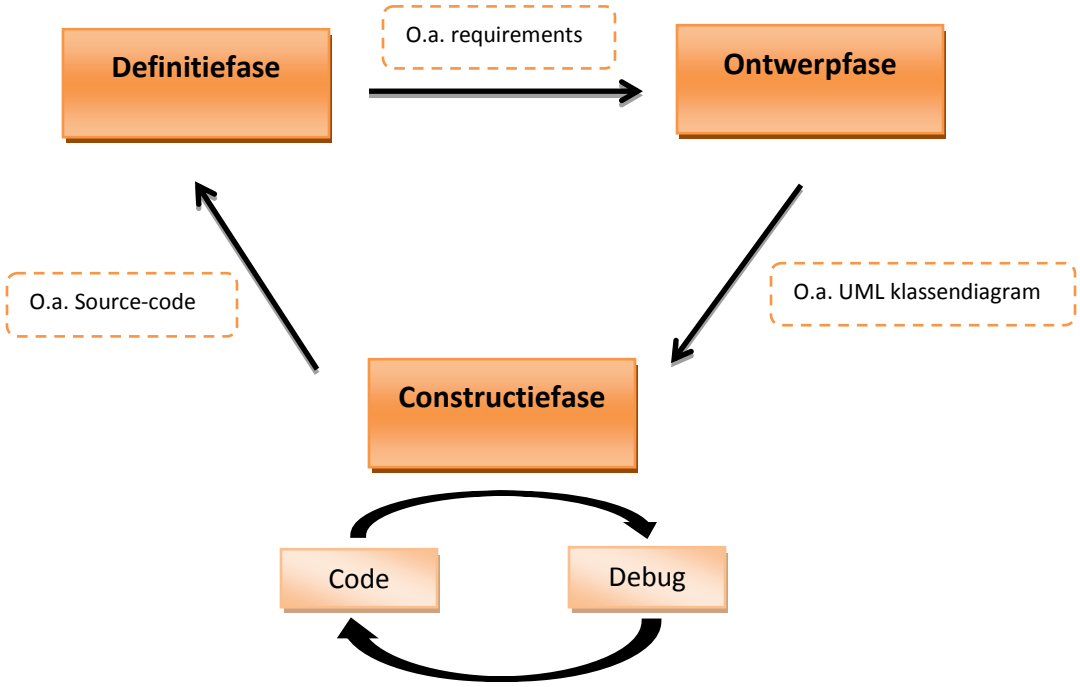
Per timebox zal de Artena Data Validator worden opgebouwd op een iteratieve en incrementele manier. Dit wil zeggen dat de activiteiten om deze producten te maken herhaald zullen worden waardoor de producten stap voor stap 'groeien'. In praktijk zal het erop neer komen dat er per iteratie één of meerdere deliverables aan bod komen die dan ontwikkeld of herzien worden:

- Iteratie 1: Ontwikkeling van het conversiesysteem
- Iteratie 2: Ontwikkeling van de validatie-engine
- Iteratie 3: Ontwikkeling van het log systeem + samenvoeging producten iteratie 1 & 2
- Iteratie 4: (Ontwikkeling) van monitoring-tool

[2] http://en.wikipedia.org/wiki/Agile_software_development

Door het gebruik van iteraties kunnen veranderende requirements voor producten opgevangen worden en kan de grootte van doorwerkende veranderingen (d.w.z. veranderingen in aangrenzende producten) geminimaliseerd worden. Wanneer een wijziging uitgevoerd moet worden aan het conversiesysteem dan kan die bijvoorbeeld ook worden meegenomen in de iteratie van de validatie-engine.

Figuur 3 geeft de activiteiten in een iteratie weer. Elke activiteit kan één of meerdere documenten opleveren of wijzigingen aan bestaande producten teweegbrengen (bij een tweede of n-de iteratie op die producten).



Figuur 3: Iteratie indeling

6.1.1 Beschrijving fases & activiteiten

Definitiefase (Onderzoek & Analyse)

Deze fase heeft betrekking op de het uitvoeren van onderzoek en analyse. Activiteiten die binnen deze fase worden uitgevoerd:

- Oriëntatie op bestaande systemen van Artena
- Een user story waarin de problemen die de aanleiding tot uitvoer van het project vormen uiteengezet worden.
- Definiëren van requirements door gesprekken te voeren met de beschreven stakeholders
- Specificeren van functionele eisen in UML use case diagrammen.

Bovenstaande gegevens zullen worden samengevoegd in een *Analyserapport*.

Ontwerpfase

Deze fase omvat het opstellen van ontwerpdelen die uiteindelijk in het ontwerprapport terecht komen.

In het ontwerp wordt aangetoond dat de, in het analyserapport vastgestelde, eisen zijn meegenomen in het ontwerp door vanuit de gebruikte ontwerpmodellen terug te verwijzen naar de requirements. Bij het databaseontwerp wordt gebruik gemaakt van 4 modellen:

- (Enhanced) Entity Relationship Diagram(*ERD*) voor het ontwerp van een conceptueel gegevensmodel waarin bijvoorbeeld validatieregels worden opgeslagen.
- Relationeel Representatie Model (*RRM*) voor de relationele definitie van het gegevensmodel.
- Relationeel Implementatie Model (*RIM*) voor een specifiek DBMS.
- Database diagram (Visio 2010 model) voor de grafische weergave van het ontwerp.

Bij het ontwerpen van de programmatuur wordt gebruik gemaakt van onderstaande modellen:

- UML klassendiagram om de structuur van de programmatuur te beschrijven.
- UML sequence diagram om het gedrag van de programmatuur te beschrijven.

Het resultaat van deze fase is een *Ontwerprapport* dat bovenstaande gegevens bevat.

Constructiefase
Deze fase bestaat uit de implementatie van de ontwikkelde ontwerpen uit de ontwerpfase. In het geval van de ontwikkeling van een datastructuur zal een (deel van een) database gebouwd worden in het gebruikte Database Management System: SQL Server 2008.

In het geval van de implementatie van de programmatuur zal gecodeerd en gedebugd worden. Dit houdt in dat de klassen beschreven in de klassendiagrammen van het ontwerp geïmplementeerd worden met behulp van Microsoft Visual Studio 2010.

Tijdens deze fase kunnen ook een aantal modellen opgesteld worden die gedetailleerde (technische) beschrijvingen van de applicatie (en code) bevatten. Dit zijn bijvoorbeeld UML klassendiagrammen op applicatieniveau of UML activiteit diagrammen. Hiervan kan een *Constructierapport* worden gemaakt.

6.1.2 Documenteren iteraties
Zoals beschreven bestaat een iteratie uit één of meerdere activiteiten waarbij een aantal activiteiten elkaar opvolgen. Ook kan het voorkomen dat sommige activiteiten binnen een iteratie helemaal niet voorkomen, omdat dat simpelweg niet nodig is.

Alvorens een iteratie gestart wordt (meestal aan het begin van de week) zal een iteratieplanning worden gemaakt waarbij vooruit wordt gekeken voor een aantal weken. De resultaten (waarin behaalde en niet-behaalde activiteiten worden beschreven) worden aan het eind van een iteratie vastgelegd.

Alle planning en resultaatdocumenten worden samengebundeld tot één groot iteratiedocument.

6.1.3 Testen
Wanneer alle eisen van de Artena Data Validator geïmplementeerd lijken te zijn zal een systeemtest[3] uitgevoerd worden door de ontwikkelaar/afstudeerder waarbij voornamelijk gekeken zal worden naar

[3] 'Testen volgens TMAP 2e druk', Hoofdstuk 3 – Blackbox testsoorten

de vastgelegde functionele en niet-functionele requirements van dit product. Wanneer het product door de deze systeemtest heen komt dan kan het product formeel gezien opgeleverd worden.

De systeemtesten maken onderdeel uit van **procesproduct 4: Testrapport Artena Data Validator.**

6.1.4 Systeemdocumentatie

De documenten behorende bij **deliverable 5, Systeemdocumentatie,** zullen worden gemaakt nadat deliverable 1 t/m 4 (dus de Artena Data Validator) zijn afgerond (en getest). Dit product zal waarschijnlijk in de laatste periode van het afstudeertraject opgebouwd.

6.2 Wijzigingen & Versiebeheer

Tijdens de uitvoer van het project komt het vaak genoeg voor dat bijvoorbeeld de analyse –en ontwerpdocumenten wijzigen. Deze wijzigingen kunnen onder andere ontstaan door veranderende eisen van de opdrachtgever(s) of door opgetreden complicaties tijdens ontwerp of implementatie.

Doordat er sprake is van een iteratieve ontwikkeling van de deliverables is het mogelijk deze wijzigingen op te vangen en onderdeel te maken van het gehele ontwikkelproces, elke iteratie kunnen wijzigingen aan betroffen documenten/producten verwerkt worden.

Tijdens het project kunnen zowel documenten als producten wijzigen. Wijzigingen gedaan aan documenten zullen worden bijgehouden in een wijzigingenlijst aan het begin van een document, waarbij de combinatie van werktitel en versienummer (‘Analyserapport v0.2) de unieke identificatie van een versie van een product is. Hierdoor wordt het direct duidelijk welke ontwikkeling een document heeft doorlopen tijdens de uitvoer van het project.

Op vrijwel alle producten wordt actief versiebeheer gehouden. Dit houdt in dat bij elke wijziging aan een product deze wijziging wordt vastgelegd en dat het versienummer verandert.

In de tabel hieronder staan een aantel eigenschappen voor het uitvoeren van versiebeheer beschreven:

Eigenschap	Beschrijving
0.1	Dit is de startversie van elk document. Dit nummer geeft aan dat het een eerste conceptversie van een document betreft
0.11	Indien zich een kleine wijziging voordoet aan bestaande data dan zal het versienummer met 0.01 worden verhoogd.
0.2	Indien zich een grote wijziging voordoet, waarbij significante veranderingen optreden of waarbij een aantal onderdelen wordt toegevoegd zal het versienummer met 0.1 worden verhoogd.
1.0	Wanneer een eerste versie klaar is voor oplevering, dan zal deze het versienummer 1.0 meekrijgen. Deze versie bevat dan in ieder geval alle onderdelen die het product moet bevatten. Er kunnen echter nog wel grote wijzigingen of toevoegingen worden gedaan na deze versie (wanneer dat nodig is).

N.B. *Voor de ontwikkeling van de programmatuur (code/database) is een dergelijk versiebeheer zoals hierboven beschreven moeilijker. Voor het versiebeheer van deze systemen zal gebruik gemaakt worden van Tortoise Subversion (§5.3.3).*

6.3 Tools

Tijdens de projectuitvoer zal gebruik gemaakt worden van meerdere tools. Deze tools zullen ondersteuning geven bij de activiteiten benodigd om de beschreven producten op te leveren.

6.3.1 Onderzoek & Analyse

Microsoft Office 2010

Deze tool zal gebruikt worden voor het maken van de analyse en ontwerpdocumenten.

Astah Community Edition[4]

Deze applicatie zal gebruikt worden voor de ontwikkeling van UML use case diagrammen.

6.3.2 Ontwerpen

Astah Community Edition

In de ontwerpfase wordt wederom Astah gebruikt voor een aantal ontwerpmodellen. Het belangrijkste model dat ontwikkeld zal worden met behulp van deze is het UML klassendiagram.

DIA (diagramming solution for Windows)

Deze applicatie zal gebruikt worden om de belangrijke EER modellen op te stellen voor het conceptuele databaseontwerp van de Artena Data Validator.

Microsoft Visio 2010

Zal gebruikt worden voor alle modellen die niet door Astah Community Edition ondersteund worden. Denk bijvoorbeeld aan databasemodellen en andere blokdiagrammen.

6.3.3 Implementeren

Tortoise Subversion (SVN)

Met behulp van SVN is het mogelijk om versiebeheer uit te voeren op de implementatie van de in dit document beschreven deliverables. Wanneer het systeem is gewijzigd en er sprake is van een nieuwe versie (bijvoorbeeld bij een nieuwe functionaliteit) kan dit worden vermeld in dit systeem.

Microsoft Visual Studio 2010

Geavanceerde ontwikkelomgeving waarmee bij Artena software ontwikkeld wordt.

Microsoft Visual Studio 2010 Team Edition

Source control inclusief branching. Dit versiebeheersysteem geeft de mogelijkheid tot registreren en bewaren van meerdere versies van gebouwde software door Artena.

Microsoft SQL Server 2008

SQL Server is de database (en het DBMS) waarmee alle data van klanten van Artena en data van Artena zelf beheerd wordt.

Microsoft SQL Server Intergration Service[5]

MS SSIS is een platform voor het maken van data-integratie –en transformatie oplossingen. Met behulp van dit systeem is het mogelijk om manipulaties uit te voeren op verschillende databronnen.

[4] http://astah.net/editions/community

7. Risicomanagement

Dit hoofdstuk beschrijft de risico's die bij dit project van belang zijn. Bij alle risico's zal aangegeven zijn welke maatregel(en) toegepast kunnen worden om de risico's te voorkomen of op te vangen.

7.1 Tijd

Risico: Voor het project is maximaal 17 weken uitgetrokken. Dit is ook zo verwerkt in de planning. Wanneer de oplevering van een deliverable extra tijd opeist heeft dit gevolgen voor de rest van de planning. Het mag niet zo zijn dat er producten helemaal niet opgeleverd worden.

Maatregel: Allereerst goed plannen, een zo goed mogelijke schatting maken van de benodigde tijd per product. Wanneer zich wijzigingen voordoen, dan moeten deze goed gedocumenteerd worden. Indien een product buiten de planning dreigt te vallen dan moet in goed overleg worden afgesproken wat de juiste stappen zijn.

7.2 Onduidelijke afspraken

Risico: Wanneer de afspraken gemaakt tussen de afstudeerder en de werkgever onduidelijk zijn en niet goed zijn vastgelegd bestaat de kans dat de werkgever tijdens de projectdoorloop niet akkoord gaat met de uitgevoerde activiteiten.

Maatregel: Alle afspraken over de opdracht duidelijk vastleggen in het afstudeerplan en het plan van aanpak, beide documenten beschikbaar stellen aan de werkgever zodat hij inzicht krijgt in de afspraken.

7.3 Beschikbaarheid stakeholders

Risico: Voor het vastleggen van de gewenste functionaliteiten moeten de stakeholders beschikbaar zijn. Dit zal in veel gevallen geen probleem zijn aangezien het grootste deel van de stakeholders van Artena zelf afkomstig is. Wanneer gekeken gaat worden naar de rapportage van de validatie dan zijn de klanten van Artena wel betrokken en dan moet (in welke vorm dan ook) gecommuniceerd kunnen worden met hen.

Maatregel: Indien stakeholders geen eisen hebben of willen geven dan moet overlegd worden met de werknemers van Artena die veel contact hebben met de klanten (zoals de consultants) zodat er, indien er eisen zijn, alsnog eisen vastgelegd kunnen worden.

7.4 Oplevering producten

Risico: Wanneer de stakeholders binnen Artena niet tevreden zijn met de deliverable, dan kan deze niet worden opgeleverd. Dit zal per product moet blijken uit de acceptatietest.

Maatregel: Tijdens de doorloop van het project veel overleg voeren met de stakeholders (en begeleiders) zodat de kans op dit risico geminimaliseerd kan worden.

[5] http://msdn.microsoft.com/en-us/library/ms141026.aspx

8. Planning

In dit hoofdstuk is de planning beschreven voor de uitvoer van het afstudeerproject bij Artena. De planning is gebaseerd op de in dit document beschreven deliverables en fasering.

Voor dit project zit de moeilijkheid bij het plannen hem in de samenhang van een aantal deliverables. Veel onderdelen van deze deliverables zullen soms tegelijkertijd uitgevoerd moeten worden, zoals ook aangegeven in §3.2. Deze deliverables zijn:

1. Conversiesysteem
2. Validatie-engine
3. Log systeem

In de planning zal worden aangegeven hoeveel tijd verwacht wordt te besteden aan een deliverable en in hoeveel iteraties dat zal gebeuren.

De monitoring-tool maakt gebruik van de gegevens gegenereerd door de validatie-engine in de vorm van validatieresultaten en heeft in dat opzicht minder samenhang met de andere deliverables.

Activiteit	Betrokken deliverables	Startdatum	Einddatum	Week-nummer
Voorbereiding				
Plan van aanpak		06-02-2012	10-02-2012	1
Requirements conversiesysteem (bestandslezers)	1-Conversiesysteem	06-02-2012	10-02-2012	1
Vergaren kennis infrastructuur systemen Artena		06-02-2012	10-02-2012	1
Iteratie 1: Conversiesysteem				
Requirements Onderzoek bestandslezers	1-Conversiesysteem	13-02-2012	17-02-2012	2
Requirements Ontwerp	1-Conversiesysteem	20-02-2012	24-02-2012	3
Ontwerp Constructie	1-Conversiesysteem	27-02-2012	09-03-2012	4-5
Iteratie 2: Validatie-engine				
Onderzoek & Analyse: Definities validatieregels Definities resultaten Validatieflow	1-Conversiesysteem , 2-Validatie-engine	12-03-2012	16-03-2012	6
Requirements	1-Conversiesysteem , 2-Validatie-engine	19-03-2012	23-03-2012	7
Ontwerp Implementatie	1-Conversiesysteem, 2-Validatie-engine	26-03-2012	06-04-2012	8-9

validatieroutine				
Iteratie 3: Logging & Afronding ADV				
Requirements & Ontwerp o.b.v. feedback iteratie 2	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem	09-04-2012	13-04-2012	10
Requirements & Ontwerp log systeem Constructie	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem	16-04-2012	27-04-2012	11-12
Iteratie 4: Monitoring & Systeemtest				
Onderzoek & Analyse, Requirements	4-Monitoring tool	30-04-2012	04-05-2012	13
Ontwerp Constructie Systeemtest ADV	2-Validatie-engine, 3-Log systeem, 4-Monitoring-tool	07-05-2012	18-05-2012	14-15
Uitloop/Afronding				
Uitloop	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem, 4-Monitoring-tool	21-05-2012	25-05-2012	16
Nazorg	1-Conversiesysteem, 2-Validatie-engine, 3-Log systeem, 4-Monitoring-tool	28-05-2012	01-06-2012	17

Tabel 1: Projectplanning

Bij de planning is de constructie van het afstudeerverslag niet inbegrepen. Dit zal in grotendeels in eigen tijd worden gedaan en daarnaast ook tijdens werkuren in de laatste periode (vier/vijf weken). Er zal gestreefd worden om de andere documenten zodanig op te stellen zodat gegevens uit deze documenten zoveel mogelijk één op één overgenomen kunnen worden in het afstudeerverslag (waar nodig uiteraard).

9. Begrippenlijst

Pensioenuitvoerder
Een pensioenfonds of verzekeraar waar pensioenfonds wordt opgebouwd door werknemer en werkgever samen. Zij betalen beide de premies voor het pensioen.
(http://nl.wikipedia.org/wiki/Pensioenuitvoerder)

Lookup values
Lookup values zijn waarden opgeslagen in één of meerdere lookup tabellen in een database. Deze waarden kunnen worden opgehaald uit de tabellen om te worden gebruikt in een systeem en om bijvoorbeeld controles uit te voeren op verkregen data.

EER, E(nhanced) E(ntity) R(elationship) model
Een conceptueel databaseontwerp model waarin concepten uit de reële wereld worden gemodelleerd aan de hand van entiteittypes en de relaties tussen deze entiteittypes. Een dergelijk diagram is onafhankelijk van elk soort databasemodel en kan gemakkelijk worden omgezet in een Relation Representatie Model.

RRM, Relationeel Representatie Model
Een logisch relationeel databaseontwerp. In een RRM worden de tabellen, kolommen, primaire sleutels en vreemde sleutels vastgelegd. Vanuit een RRM kan een RIM worden opgesteld.

RIM, Relationeel Implementatie Model
De mapping van een RRM naar een specifiek DBMS (zoals SQL Server 2008) waarmee de kenmerken vastgelegd in een RRM geïmplementeerd kunnen worden.

10. Literatuur

[Janssen, P. & Van Craen, J.P.] (2011). **Projectmanagement volgens PRINCE2, 3e editie**. Amsterdam: Pearson Education

[Pol, M. & Teunissen, R. & Van Veen, E.] (2002). **Testen volgens TMAP, 5e druk.** 's Hertogenbosch: Uitgeverij Tutein Nolthenius

[Tré, G.] (2007). **Principes van Databases.** Amsterdam: Pearson Education

[Warmer, J. & Kleppe, A.] (2007). **Praktisch UML, 4e editie.** Amsterdam: Pearson Education

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage B

Iteratiedocument

Planningen & Resultaten van doorgelopen iteraties

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 18-05-2012

Inhoudsopgave

1. Iteratie 1: Conversiesysteem 3
1.1 Planning 3
1.2 Resultaten 4
2. Iteratie 2: Validatie-engine 5
2.1 Planning 5
2.2 Resultaten 6
3. Iteratie 3: Artena Data Validator & Logging 8
3.1 Planning 8
3.2 Resultaten 9

1. Iteratie 1: Conversiesysteem

Week 2-5: 13-02-2012 t/m 09-03-2012

1.1 Planning

Activiteit 1: Requirements + Onderzoek (definitiefase)

Op basis van input van stakeholders zullen requirements (zowel functioneel als niet-functioneel) worden opgesteld. Deze requirements zullen vervolgens worden ingedeeld naar prioriteit zodat duidelijk wordt welke eisen het belangrijkste zijn. Op basis van de eisen zullen vervolgens UML use cases worden opgesteld waarin de functionele eisen uitgewerkt zullen worden.

Met behulp van de requirements moet vervolgens een onderzoek worden uitgevoerd naar het uitlezen van bestanden:

- Voor het conversiesysteem zijn er meerdere oplossingen mogelijk qua uitlezen van verschillende soorten bestanden (bestandsformaten). De mogelijke oplossingen moeten onderzocht worden en er moet een keuze worden gemaakt voor de beste oplossing.

- Om de verschillende oplossingen te testen zal een prototype of testapplicatie gemaakt moeten worden (afhankelijk van de oplossing).

- Op basis van de resultaten van de (unit)tests kan worden geconcludeerd of een oplossing juist is en waarom dat zo is.

Het resultaat van deze iteratie is een Analyserapport

Activiteit 2: Ontwerp (ontwerpfase)

De requirements zullen omgezet moeten worden in een ontwerp. Daarbij kan ook een eventueel prototype ontwikkeld bij het onderzoek worden gebruikt.

Een ontwerp zal in ieder geval bestaan uit:

- Databaseontwerp (ERD, RRM, RIM)
- UML klassendiagram (domeinniveau) voor de structuur van de software
- UML sequencediagram voor het gedrag van de software

Het resultaat van deze activiteit is een ontwerprapport met daarin alle benodigde ontwerpdelen.

Activiteit 3: Implementatie (constructiefase)

Met behulp van het ontwerpdocument kan worden begonnen met de implementatie van het conversiesysteem. Een eventueel prototype kan hierbij als basis dienen.

Het resultaat van deze activiteit is een systeem/programmatuur waarin het ontworpen conversiesysteem aanwezig is.

Dag #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Activiteit	1	1	1	1	1	1	1	1, 2	1,2	1,2	2	2	3	3	3
Dag #	16	17	18	18	20										
Activiteit	3	3	3	3	3										

1.2 Resultaten

Activiteit 1.1: Onderzoek Excel

- De Excel Data Reader for .NET (C#) is gekozen als oplossing voor Excel conversie
- Unit-tests zijn uitgevoerd voor de Excel Data Reader
- Op basis van resultaten tests zijn enkele patches uitgevoerd op de gebruikte bibiliotheek om de problemen met het uitlezen van Excel bestanden op te lossen.

Activiteit 1.2: Onderzoek CSV

- Voor de CSV lezer is gekozen voor een eigen implementatie (in C#)
- De unit-tests gaven aan dat de lezer nog steeds een aantal bugs bevat die opgelost moeten worden.

Activiteit 1.3: Requirements

Lijst met functionele en niet-functionele eisen maar daarin onder andere:

- Stakeholders zijn toegevoegd bij eisen
- Functionele en niet-functionele eisen ingedeeld op prioriteit (MoSCoW)
- Bestandsformaten te ondersteunen: Excel + CSV
- Definities invoer –en uitvoer conversiesysteem
- ANSI tekenset ondersteuning

Activiteit 1.4: Use cases

Specificatie van de functionele eisen uit de requirements:

- Use case m.b.t. het gehele conversieproces
- Use case m.b.t. het inladen van een bestandslezer

Activiteit 2: Ontwerprapport

Ontwerpdelen die in onder andere het ontwerprapport zitten:

- Definitie van verwachtte bestandsstructuur
- Toepassing van DLL
- Database om invoer conversiesysteem aan te bieden
- Database om uitvoer van conversiesysteem vast te leggen
- Toepassing O/R mapper LINQ-to-SQL
- Klassendiagram (domeinmodel)

Activiteit 3: Implementatie

Met behulp van de testapplicatie gemaakt tijdens het onderzoek en het opgestelde ontwerp is een implementatie van het conversiesysteem gemaakt. Met deze implementatie is het mogelijk om Excel (97-2003/2007) –en CSV bestanden uit te lezen en de inhoud weg te schrijven naar een database in CSV formaat. Deze inhoud is dan tevens gekoppeld aan een template waardoor is vastgelegd welke inhoud bij welke bestanden hoort.

2. Iteratie 2: Validatie-engine

Week 6-9: 12-03-2012 t/m 06-04-2012

2.1 Planning

Activiteit 1: Onderzoek (definitiefase)
Er moet eerst een onderzoek uitgevoerd worden naar de mogelijke implementatie van het validatiesysteem:

- Welke soorten validaties moeten uitgevoerd worden en waarom
- Definities van de validatieregels vastleggen
- Definiëren hoe de resultaten van validaties eruit kunnen zien
- Vastleggen van een mogelijke volgorde in verschillende soorten validaties

Het resultaat van dit onderzoek kunnen één of meerdere documenten zijn waarin bovenstaande gegevens verwerkt worden. Het is ook mogelijk dat er ontwerpdelen kunnen voorkomen in deze documenten (zoals databasediagrammen)).

Activiteit 2: Vergadering over datakwaliteit
Klanten van Artena hebben een steeds grotere vraag naar een hogere kwaliteit van hun data en naar de controle van deze kwaliteit. Artena wil in reactie hierop een data-quality-framework kunnen aanbieden waarmee datakwaliteit voor klanten gewaarborgd kan worden. De validatie-engine zou hierbij een belangrijk onderdeel kunnen vormen.

In deze vergadering zullen ideeën en gedachten over datakwaliteit worden uitgewisseld met medewerkers van Artena (ook stakeholders 2, 3, 4). Hierbij kan het resultaat van activiteit 1 belangrijk zijn.

Activiteit 3: Requirements + Use cases (definitiefase)
Met activiteit 1 en de eventuele feedback uit de vergadering is het mogelijk om requirements op te stellen voor de Artena Data Validator. De functionele eisen zullen vervolgens verder gespecificeerd worden in use cases.

Het resultaat van deze activiteit is een analyserapport met daarin eisen en use cases voor de validatie-engine.

Activiteit 4: Ontwerprapport (ontwerpfase)
Het ontwerprapport gebaseerd op het analyserapport van activiteit 3. Bij het ontwerp zal waarschijnlijk gekeken worden naar het ontwerp en implementatie van het conversiesysteem en de eventuele uitbreiding daarvan.

Het resultaat van deze activiteit zijn één of meerdere ontwerpdelen voor de validatie-engine.

Activiteit 5: Implementatie (constructiefase)
De implementatie van het ontwerprapport waardoor een eerste versie van de validatie-engine zal ontstaan. Met behulp van dit systeem moet het mogelijk zijn om de gespecificeerde validaties uit te voeren.

Week #	6	7	8

Dag #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Activiteit	1	1	1	1	1	1	1	1,3	1,3	1,2,3	3	3	4	4	4

Week #	9				
Dag #	16	17	18	19	20
Activiteit	5	5	5	5	5

2.2 Resultaten

Het verloop van de geplande activiteiten is anders verlopen dan verwacht. De vergadering vond pas plaats op 30-03-2012 waardoor er meer tijd was om de analyse en ontwerp uit te voeren. Versie 0.2 van het ontwerprapport is getoond tijdens de vergadering. Het onderzoek naar datakwaliteit is bijgevoegd in het volledige analyserapport.

Op 18-03 is door de begeleidend examinator feedback gegeven op de requirements waaruit bleek dat de requirements onder andere SMARTer geformuleerd moesten worden. De aanpassingen op de requirements zijn ook doorgevoerd op de eisen uit de eerste iteratie.

Activiteit 1 + 3: Analyserapport (onderzoek + requirements + use cases)

In het 'onderzoek' zijn alleen de theorieën achter datakwaliteit behandelt. Op basis van de bekende problemen zijn meerdere datakwaliteitsaspecten vastgelegd waarop de validatie-engine van toepassing zou zijn.

Er zijn ook requirements opgesteld op basis van gesprekken met de stakeholders (2-4). Deze requirements hebben zoals eerder beschreven na 18-03 een grote herziening gehad waardoor ze veel meer SMART geformuleerd zijn.

Tot slot zijn de requirements uitgewerkt in UML use cases .

Activiteit 2: Vergadering

- De vergadering vond pas plaats op 30-03-2012
- In deze vergadering kwam naar voren dat het gehele idee van de validatie-engine dat beschreven is in het analyserapport en ontwerprapport een belangrijke rol gaat vervullen in de eerste fase van de meeste dataprocessen die zich afspelen in de systemen van Artena.
- De stakeholders waren tevreden over de verschillende validatie-niveaus
- Één soort validatie toevoegen: uniqueness validatie op kolom niveau
- Tweede vergadering op 13-04-2012 waarin misschien een demo gegeven kan worden

Activiteit 4: Ontwerprapport

In het ontwerprapport zijn veel verschillende soorten ontwerpdelen aanwezig:

- Beschrijving verschillende validatieniveaus
- Beschrijving soorten validaties
- Beschrijving resultaten van validaties
- Databaseontwerp (gebaseerd op database conversiesysteem) incl. EER diagram
- Beschrijving programmatuur inclusief een plug-in systeem d.m.v. UML klassendiagrammen
- UML sequencediagram met daarin beschrijving validatieproces

Activiteit 5: Implementatie

Met behulp van de conceptdocumenten is een implementatie van de validatie-engine gemaakt in C#. Hierbij is een validatieroutine, plug-in structuur en de databasestructuur voor het vastleggen van de validatieregels en (enkele) validatieresultaten meegenomen.

Er zijn maar drie typen 'validatie plug-ins' gemaakt: presence, datatype en uniqueness. De overige plug-ins zullen in de derde iteratie geïmplementeerd moeten worden.

3. Iteratie 3: Artena Data Validator & Logging

Week 10-14: 09-04-2012 t/m 11-05-2012

Het aantal weken in deze iteratie wijkt af van de planning (waarin week 10-12 beschikbaar zijn gesteld). Dit komt doordat in overleg met de belanghebbenden is vastgelegd dat de monitoring-tool van ondergeschikt belang is aan de Artena Data Validator. De vierde iteratie zal dus afvallen en de Artena Data Validator zal verder verbeterd en uitgebreid kunnen worden in de vrijgekomen tijd.

3.1 Planning

Activiteit 1: Analyserapport (definitiefase)

Het analyserapport van de laatste iteratie heet twee doelen:

- Beschrijving van de gewenste functionaliteiten log systeem
- Beschrijving van de wijze waarop het conversiesysteem en de validatie-engine samenkomen tot één systeem (de Artena Data Validator).

Hiertoe zullen de eisen uit de eerste twee activiteiten worden uitgebreid en samengevoegd

Activiteit 2: Vergadering over datakwaliteit 2

Op 13-04-2012 zal er weer een vergadering zijn over datakwaliteit waarbij ik zal trachtten een demo te tonen van de Artena Data Validator. De demo zal bestaan uit een conversie en validatieproces.

Activiteit 3: Ontwerp (ontwerpfase)

Met behulp van de ontwerprapporten van de eerste en tweede iteratie zal het ontwerp voor de volledige Artena Data Validator opgesteld kunnen worden. Dit ontwerp is vooral van toepassing op de programmatuur van het systeem omdat de database al geschikt is voor de Artena Data Validator.

- Databaseontwerp log systeem
- Programmatuur Artena Data Validator: UML klassendiagram

Het resultaat van deze activiteit is een ontwerprapport voor de gehele Artena Data Validator

Activiteit 4: Implementatie (constructiefase)

In deze activiteit zullen de systemen van de eerste en tweede iteratie samengevoegd worden op basis van het ontwerprapport van activiteit 3. Daarnaast zullen de overgebleven validatie plug-ins vanuit de tweede iteratie geïmplementeerd worden tijdens deze activiteit.

Deze activiteit heeft een constructierapport als resultaat. In dit rapport worden meer technische beschrijvingen gegeven van belangrijke onderdelen van de Artena Data Validator.

Activiteit 5: Systeemtest (constructiefase)

Nadat de implementatie is afgerond zal een systeemtest uitgevoerd moeten worden om te controleren of de functionaliteiten van de Artena Data Validator voldoen aan de eisen gesteld in analyserapport (activiteit 1).

Het resultaat van deze activiteit is een testplan, testscript en testrapport.

Week #	10					11					12				
Dag #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Activiteit	1	1	1	1	1,2	*pasen*	3	3	3	3	4	4	4	4	4

Week #	13					14				
Dag #	16	17	18	19	20	21	22	23	24	5
Activiteit	4	4	4	4	4	*5*	5	5	5	5

3.2 Resultaten

Activiteit 1: Analyserapport

Het analyserapport bevat de laatste versies van de requirements (functioneel en niet-functioneel). Er zijn enkele nieuwe requirements toegevoegd met betrekking tot het conversiesysteem. Nieuwe use case diagrammen waarin de use cases uit eerdere iteraties zijn samengekomen + use case diagram over beheer Artena Data Validator.

Activiteit 2: Vergadering datakwaliteit 2

- De vergadering vond pas plaats op 20-04-2012
- In deze vergadering heb ik een demo gegeven van de implementatie van de Artena Data Validator tot dan toe
- Het conversiesysteem en validatie-engine samengevoegd en meerdere validatieregels (niet alle soorten)

Activiteit 3: Ontwerp

Het ontwerprapport bevat:

- Databaseontwerp log systeem (EER, RRM, RIM)
- Programmatuurbeschrijving
- DLL + facade class
- UML klassendiagrammen na samenkomen conversiesysteem en validatie-engine
- UML klassendiagram log systeem
- Singleton en Observer pattern log systeem
- Beschrijving Importproces a.d.h.v. UML sequence

Activiteit 4: Implementatie

De Artena Data Validator is volledig geïmplementeerd doordat gebruik gemaakt kon worden van extra weken die zijn vrijgekomen door het laten vallen van de ontwikkeling van de monitoring-tool.. Het systeem bestaat nu uit 4 onderdelen:

- Globaal deel met daarin facade class en entry point van de Artena Data Validator DLL
- Conversiesysteem met daarin een conversieproces en bestandslezers
- Validatie-engine met daarin een validatieproces en plug-in systeem voor verschillende soorten validatieregels
- Log systeem met daarin een Singleton en Observer pattern waardoor een mogelijkheid tot verschillende soorten loggers aanwezig is.

Een viertal validatie plug-ins kon uiteindelijk geïmplementeerd worden door de vrijgekomen tijd in de planning:

- RowsRange (tablel-level)
- Fieldscount (column-level)

- Lookup (field-level)
- Relationship (field-level)

Het constructierapport is met succes opgesteld. In het rapport is een technische beschrijving te vinden van de belangrijke onderdelen van de Artena Data Validator waaronder UML klassendiagrammen op applicatieniveau en UML activiteitsdiagrammen.

Activiteit 5: Systeemtest

Voor de systeemtest is met succes een testplan en testscript opgesteld. Er is gebruik gemaakt van de testspecificatietechniek *DataFlowTest* van de methode TMap. Het testscript is uitgevoerd en van de resultaten is een testrapport opgesteld. Volgens de test voldoet het systeem aan de functionele eisen gesteld in de requirements. Het systeem heeft echter nog wel enkele kinderziektes (zoals lage performance en kleine bugs).

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage C

User story

Toelichting op de problematiek

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 01-04-2012

1. Inleiding

Dit document beschrijft de problemen die momenteel worden ervaren door de ontwikkelaars bij Artena wanneer nieuwe klantbestanden worden aangeleverd. Deze problemen waren de aanleiding tot uitvoer van de afstudeeropdracht en zijn gedeeltelijk aangegeven in het Plan van Aanpak.

Met behulp van dit document wordt getracht meer inzicht te geven in de problematiek die momenteel aanwezig is bij het verwerken van de bestanden. De meeste voorkomende en belangrijkste problemen worden in dit document uiteengezet. De problemen zoals in dit document beschreven hoeven natuurlijk niet altijd voor te komen of in dezelfde combinatie of volgorde.

2. Case

In het voorbeeld dat in dit document wordt gegeven wordt uitgegaan van een nieuwe aanlevering van een bestand van klant X. Dit bestand is in Excel formaat, bevat twee sheets en verschillende soorten gegevens.

2.1 Tabel 1

In deze paragraaf worden de meest voorkomende problemen die zich kunnen voordoen binnen één tabel beschreven. Figuur 1 toont de eerst tabel van het door klant X aangeleverde bestand.

Klantnr	Naam	Geb. Datum	Geslacht	Salaris	Percentage	Burgerlijke staat
11	A. Polet	03-11-1989	M	3000	0.01	Alleenstaand
22	J. Smit	21 februari 2012	Female	1000000	0,01	GEH
22		01-01-2012	V	vijfduizend	1.01	ALS

Figuur 1: Tabel 1 uit Excel bestand van klant X

Zoals beschreven in het Plan van Aanpak (en afstudeerplan) zijn er meerdere problemen die de ontwikkelaar van Artena tegenkomt bij het verwerken van een klantbestand. Hieronder zijn de belangrijkste problemen beschreven aan de hand van de waarden uit de tabel van figuur 1.

Uniekheid
Het derde klantnummer is hetzelfde als het tweede klantnummer (22) terwijl het klantnummer uniek moet zijn.

Verplichte invulling
Het derde veld van de kolom *Naam* is niet ingevuld terwijl de naam van de klant verplicht ingevuld moet zijn

Formaat
Het tweede veld in de kolom *Geb. Datum* bevat een datum die niet van het juiste formaat/layout is.
Het tweede veld uit de kolom geslacht voldoet niet aan het vereiste formaat (in dit geval M | V)

Bereik
Het salaris van de tweede klant is 1000000 terwijl dit getal tussen de 1000 en 20000 moet liggen.

Datatype
De waarde van het salaris van de derde klant bevat tekst terwijl het systeem verwacht dat er een getal aanwezig is.

Formaat
Het (decimale) getal van de tweede klant in de kolom *Percentage* is gescheiden door een komma i.p.v. punt

Lookup
De burgerlijke staat van de eerste klant voldoet niet aan de mogelijke waarden die voor die kolom in een lookup tabel zijn vastgelegd (bijvoorbeeld ALS | GEH)

2.2 Tabel 2

De tweede tabel toont een tweetal problemen die kunnen voorkomen wanneer een bestand meerdere tabellen bevat.

Klantnr	Naam partner	Geboortedatum partner
22	P. Post	
33	P. Boks	03-03-1990

Figuur 2: De tweede tabel

Verplichte invulling (relationeel)
De burgerlijke staat van de tweede klant uit tabel 2 (klantnr 22) is GEH. Hierdoor met zowel de naam als de geboortedatum van de partner ingevuld zijn. In de tweede tabel van figuur 2 is dat niet het geval.

Integriteitcontrole
Er bestaat geen klant met klantnr 33 in de eerste tabel van het aangeleverde bestand.

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage D

Orientate Systemen Artena

Beschrijving relevante systemen Artena

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 02-04-2012

Inhoudsopgave

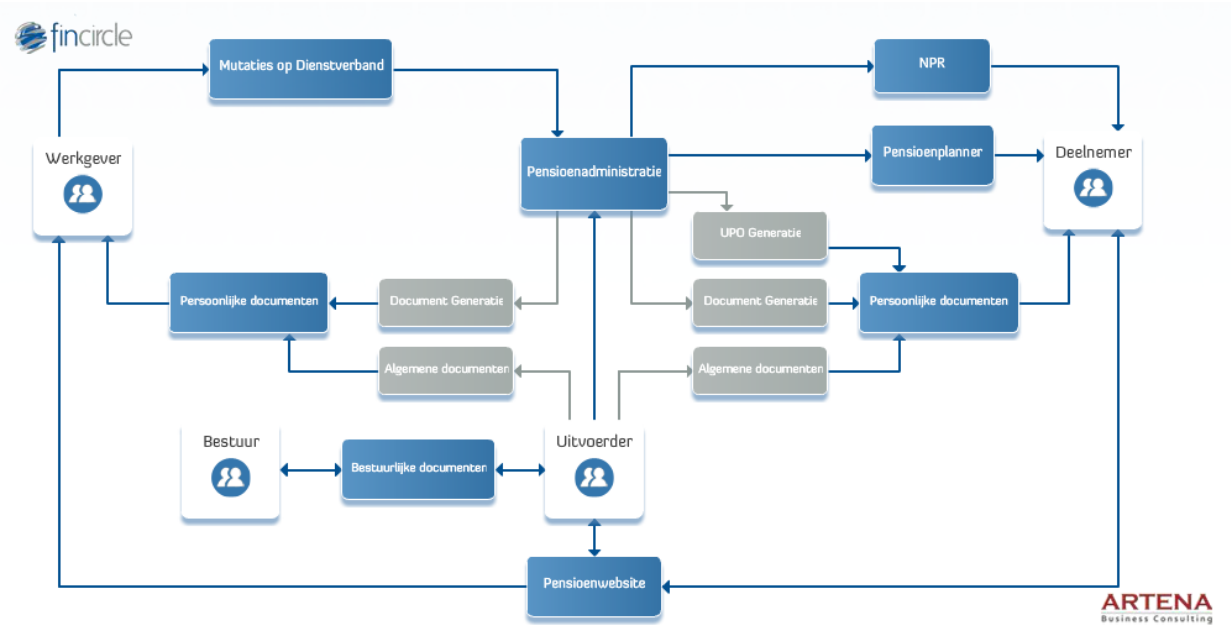
1. Inleiding 3
2. Technische specificaties FinCircle 4
3. Deelnemer portal 4
4. Architectuur FinCircle (globaal) 5
5. Architectuur Artena Data Validator (globaal) 7

1. Inleiding

Dit document beschrijft (globaal) de huidige infrastructuur van een systeem waar de Artena Data Validator bij betrokken kan worden: FinCircle. Fin(ancial)Circle is een volledige pensioenoplossing waarmee pensioenuitvoerders (pensioenfondsen), werkgevers en werknemers inzicht kunnen krijgen in en wijzigingen kunnen doen aan pensioenen.

Het systeem heeft globaal gezien de volgende functionaliteiten:

- Pensioenadministratie, bedoeld voor beheer van administratieve pensioengegevens en het berekenen van pensioenrechten.
- Pensioenplanner, waarmee werknemers (of deelnemers) via een portal zijn pensioensituatie kan simuleren en wijzigingen kan aanbrengen aan de opbouw van zijn pensioen
- Documentopslag voor werkgever, werknemer en bestuur
- Genereren van Uniforme Pensioen Overzichten (UPO's) vanuit het administratiesysteem waarmee
- Communicatie met het Nationaal Pensioen Register, waarmee deelnemer en beheerder gegevens kunnen inzien.



Figuur 1: Softwareproces FinCircle

Zoals beschreven in het Plan van Aanpak is voor dit project alleen het onderdeel voor de werknemers belangrijk, het deelnemers portal. Van dit onderdeel zal in dit document de huidige structuur worden beschreven. Meer informatie over het gehele systeem van FinCircle is in te zien op http://www.fincircle.nl .

2. Technische specificaties FinCircle

Fincircle is ontwikkeld met behulp van het Microsoft.NET 4.0 framework. Onderstaande technologieën zijn gebruikt bij de implementatie van het FinCircle systeem:

- Microsoft IIS Server 6.0 & 7.0
- Microsoft Visual C#
- Microsoft ASP.NET
- DotNetNuke[1]
- AJAX (JavaScript in combinatie met XML)
- Webservices zoals WCF (Windows Communication Foundation)

Alle gegevens en data gebruikt door beheerders, werkgevers en deelnemers in het FinCircle systeem worden bijgehouden in Microsoft SQL Server 2008 databases op individuele database servers.

De gegevens gebruikt voor het FinCircle systeem worden voor het overgrote deel door derden aangeleverd aan Artena. Alvorens deze gegevens gebruikt kunnen worden voor het FinCircle systeem moeten er meestal één of meerdere mutaties uitgevoerd worden op deze gegevens. Voor het uitvoeren van deze mutaties wordt Microsoft SQL Server Integration Services gebruikt.

3. Deelnemer portal

Het deelnemer portal kan worden gezien als de website waarop een werknemer/deelnemer inlogt met zijn/haar eigen persoonlijke gebruikersnaam en wachtwoord. Na het inloggen kan deze persoon de door zijn/haar pensioenfonds aangeleverde gegevens inzien of wijzigen.

Naast deze basisfunctionaliteiten verschillen de deelnemer portals per pensioenfonds op het gebied van functionaliteiten, sommige portals bieden de deelnemer de mogelijkheid om verschillende pensioenscenario's te simuleren waardoor de deelnemer de beste pensioenvorm voor zijn/haar situatie kan aangegeven bij zijn/haar pensioenfonds.

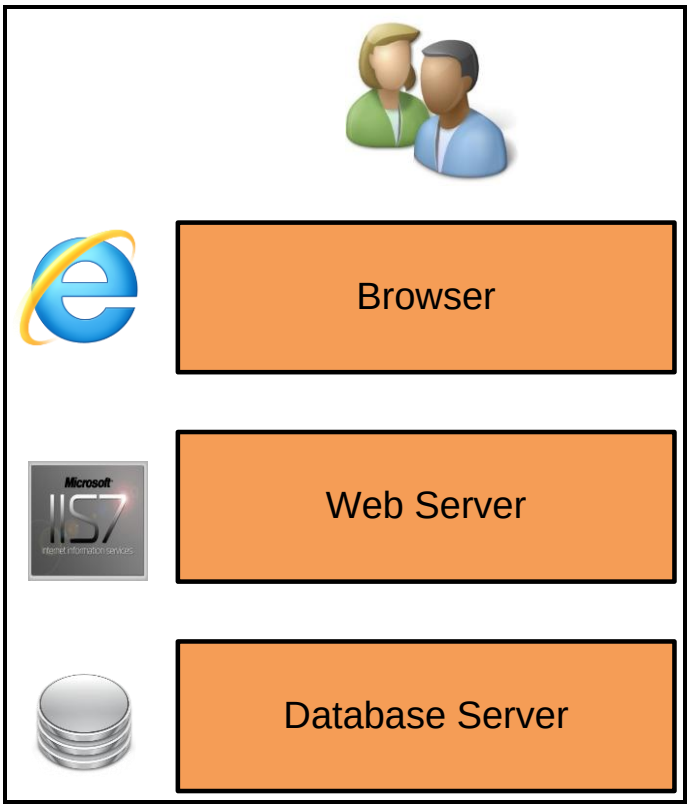
Figuur 2 toont een deel van de user interface van een deelnemer portal voor ANWB deelnemers.

Figuur 2: ANWB portal

[1] http://www.dotnetnuke.com

4. Architectuur FinCircle (globaal)

FinCircle is een web-based platform voor pensioenoplossingen en bestaat uit meerdere onderdelen zoals aangegeven in figuur 1. Vanuit de deelnemer gezien zijn de delen waaruit het portal bestaat vrij logisch: De deelnemer communiceert via een browser met een web server die zijn gegeven vervolgens weer uit de database op de database server haalt. Zie figuur 3.



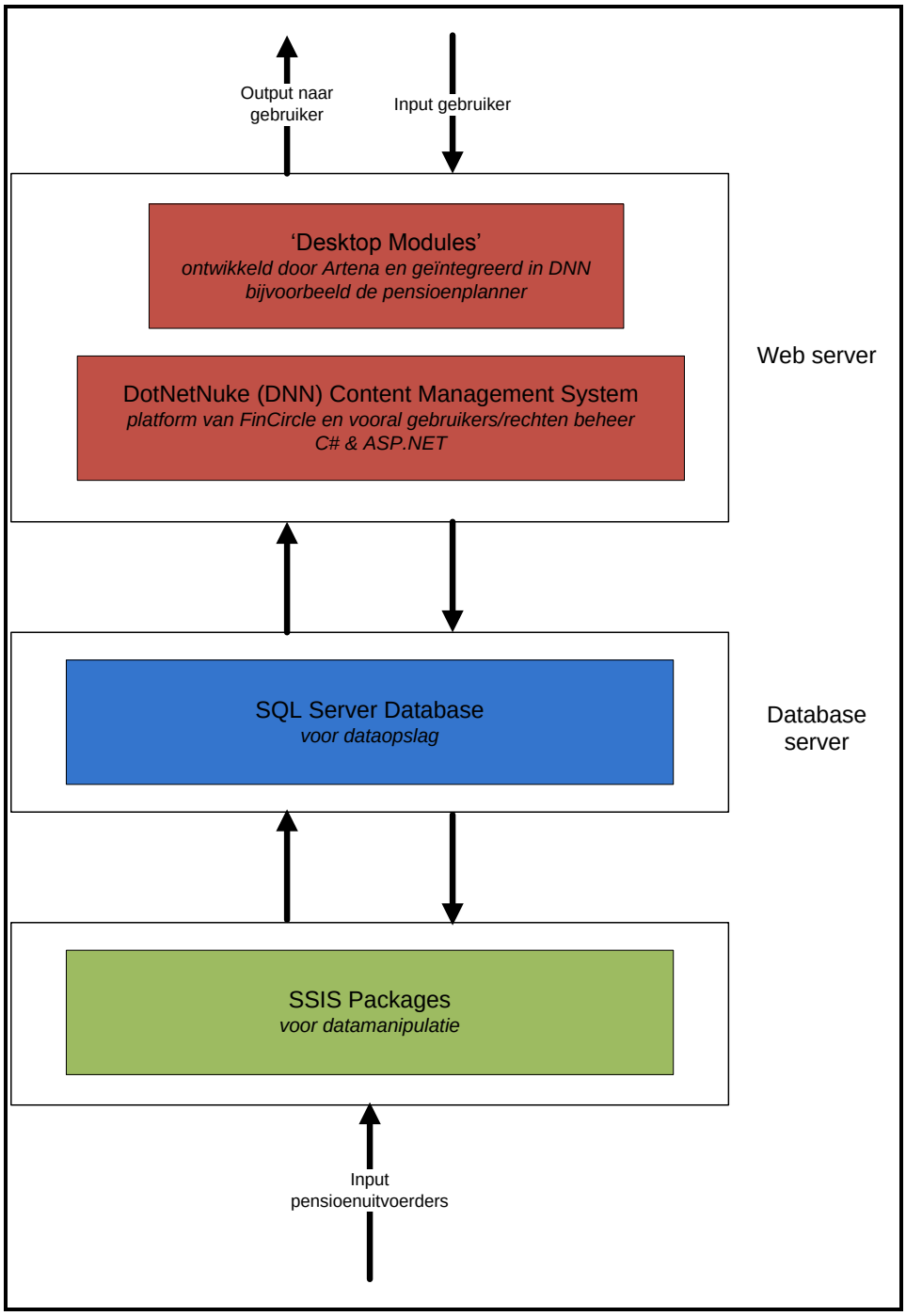
Figuur 3: Lagen van deelnemer portal

Voor het gemak is het logo van Internet Explorer 10 bij de browser gezet, andere browsers zoals Google Chrome zijn ook compatibel. De web server is een Microsoft Internet Information Services Server.

Het is mogelijk om verder en dieper te kijken dan de indeling van figuur 3. **Figuur 4** toont een verdere detaillering van de lagen uit figuur 3 en beschrijft ook de globale gegevensstroom tussen deze lagen.

In de gegevensstroom is duidelijk te zien dat er in het FinCircle systeem een soort hiërarchie aanwezig is:

- De data wordt aangeleverd door klanten van Artena
- De data wordt verwerkt met behulp van één of meerdere SSIS Packages
- De data wordt weggeschreven naar de database op de web server
- Wanneer een deelnemer een request uitvoert via zijn browser dan wordt dat request afgevangen door de web server om vervolgens verwerkt te worden met de data aanwezig in de database.



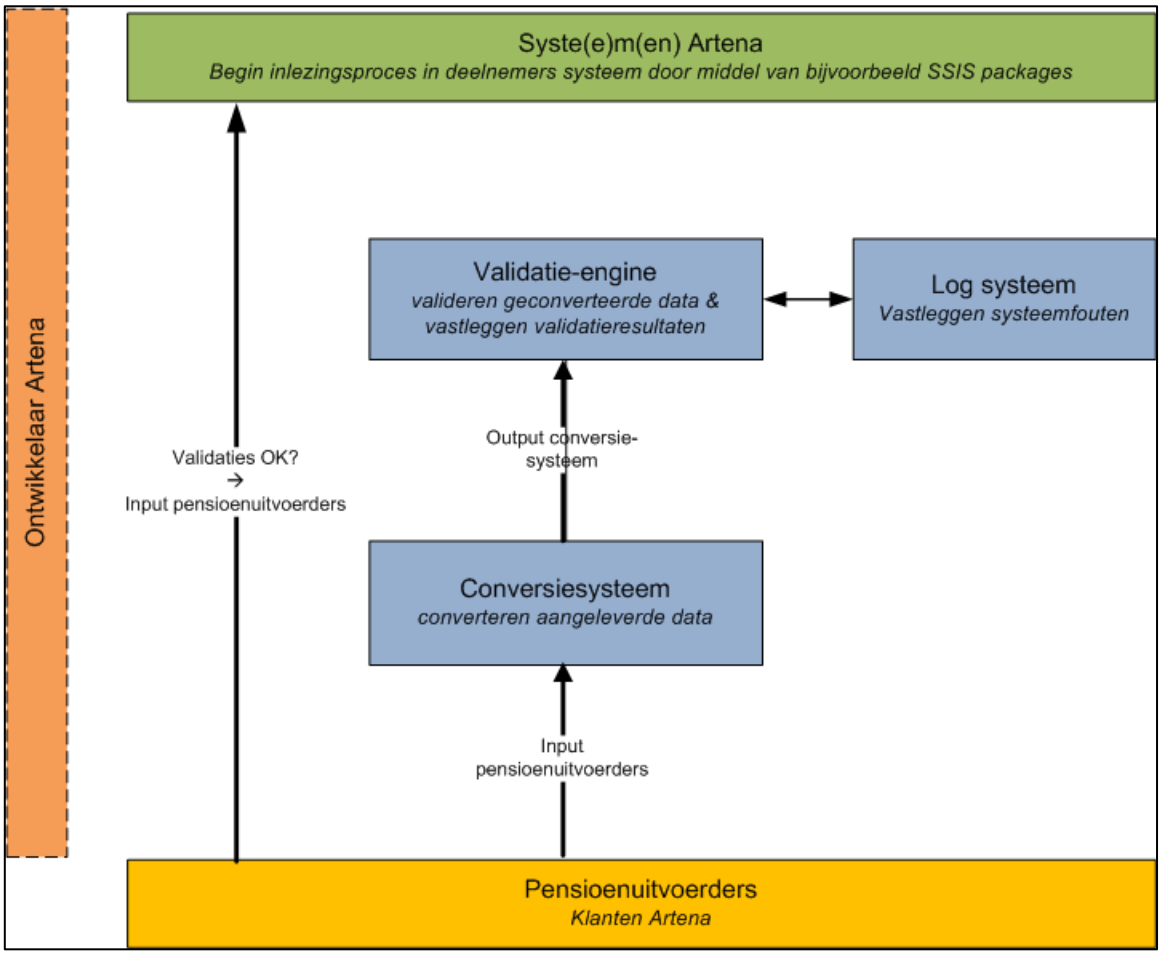
Figuur 4: Detail lagen & gegevensstroom

N.B. Het is duidelijk dat er vanuit het huidige systeem geen directe feedback is naar de pensioenuitvoerders toe wat betreft de aangeleverde gegevens. Indien er toch feedback gegeven moet worden dan gebeurt dat meestal per mail of telefoon door de ontwikkelaars van Artena.

5. Architectuur Artena Data Validator (globaal)

Op basis van globale beschrijving van de architectuur van het deelnemer portal, de opdrachtomschrijving en het plan van aanpak is het mogelijk om aan te geven waar de te implementeren systeemdelen van de Artena Data Validator in de hiërarchie thuishoren en hoe de architectuur eruit ziet.

De systeemdelen zullen in het lagenmodel op het laagste niveau aanwezig zijn, namelijk direct na de input van de pensioenuitvoerders. Het is de bedoeling dat de gewenste functionaliteiten van de systeemdelen uitgevoerd zullen worden voordat de data het huidige systeem bereiken. Ter illustratie figuur 5.



Figuur 5: Plaatsing validatie-systeem in beschreven hiërarchie

Zoals beschreven in bovenstaand diagram is het de bedoeling dat de Artena Data Validator niet alleen gebruikt zal worden voor het in dit document beschreven FinCircle systeem, maar ook voor andere systemen van Artena.

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage E

Analyserapport Iteratie 1: Conversiesysteem

Versie 1.0

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 23-02-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	10-02-2012	Concept	Toegevoegd: • Inleiding functionele eisen • Eerste functionele eisen
0.2	14-02-2012	Concept	Toegevoegd: • Functionele eis 3: conversie van XML • Functionele eis 4: specificatie van invoer conversiesysteem
0.3	15-02-2012	Concept	Toegevoegd: • Functionele eisen 6-8 over de uitvoer van het conversiesysteem • Niet-functionele eis 1: RFC-4180m.b.t. CSV notatie • Prioritering eisen • Betrokken stakeholders per eis Gewijzigd: • Functionele eis 1: invoer betreft niet meer een verwijzing naar een klant, maar een verwijzing naar een template
0.4	21-02-2012	Concept	Toegevoegd: • Functionele eis 9: meerdere conversies tegelijkertijd • Functionele eis 10: automatisch bepalen bestandsformaat • Functionele eis 11: automatisch bepalen bestandsformaat • UML use case diagram + beschrijvingen
1.0	23-02-2012	Eerste versie	Toegevoegd: • Functionele eis 12: opstartmechanisme van het conversiesysteem • Niet-functionele eis 2: ondersteuning karakterset • Niet functionele eis 3: flexibiliteit conversiesysteem • Beschrijving uitgevoerd onderzoek naar bestandslezers

Gerelateerde documenten	Versie	Datum
Onderzoek bestandslezers	1.1	17-02-2012
Unit Test – Excel File Reader	1.0	21-02-2012
Unit Test – CSV File Reader	1.0	22-02-2012

Inhoudsopgave

1. Inleiding 5
2. Requirements 6
2.1 Functionele eisen 6
2.2 Niet-functionele eisen 8
3. Use cases (UML) 8
3.1 Use case diagram 9
3.2 Use case beschrijvingen 9
4. Onderzoek bestandslezers 11

1. Inleiding

In dit document wordt de uitgevoerde analyse voor de eerste iteratie (het conversiesysteem) beschreven. Het grootste deel van deze analyse omvat de definitie van requirements. Door de requirements vast te leggen voor het conversiesysteem is vooral veel duidelijk geworden over de verwachte invoer en uitvoer van het systeem.

De requirements voor het conversiesysteem zijn vastgesteld door overleg te voeren met de stakeholders, vooral stakeholder 2, zodat veel duidelijk is geworden over de visie op het conversiesysteem.

In dit document zullen eerst de requirements zoals deze zijn opgesteld in samenwerking met de stakeholders uiteengezet worden (**hoofdstuk 2**). Vervolgens worden de UML use case diagrammen die zijn opgesteld vanuit deze requirements beschreven (**hoofdstuk 3**). Tot slot wordt het onderzoek dat is uitgevoerd naar bestandslezers globaal beschreven (**hoofdstuk 4**).

NB. *Dit rapport bevat de nieuwe requirements structuur die is ingevoerd na feedback van de begeleidende examinator. De nieuwe structuur zorgde voor een veel duidelijkere en meer SMART beschrijving van de requirements. Deze feedback heeft plaatsgevonden op 28-03-2012. In het analyserapport van de laatste iteratie (3) waren de requirements dan ook op deze manier ingedeeld. Om het verhaal consistent te houden zijn alle analyserapporten op deze manier bijgewerkt.*

2. Requirements

Het conversiesysteem is het eerste subsysteem dat verantwoordelijk is voor de verbetering van de gegevenstransfer bij Artena. Het conversiesysteem is verantwoordelijk voor het uitlezen en eventueel converteren van de aangeleverde bestanden (van klanten van Artena) naar een database zodat de data uit de (verschillende) klantbestanden op eenzelfde manier gestructureerd en opgeslagen is. Dit wordt ook wel *staging* genoemd. Het conversiesysteem is dus eigenlijk een stagingsysteem met enkele conversiemogelijkheden. In dit document zal gesproken worden van een conversiesysteem. Wanneer gegevens zijn 'geconverteerd' worden deze (uitvoer)gegevens aangeduid als een import.

De aangeleverde bestanden van de klanten (hierna te noemen 'klantbestanden') kunnen bedoeld zijn voor verschillende systemen die Artena in gebruik heeft. Deze klantbestanden kunnen dus verschillende doelen dienen en hebben dus ook een verschillende opbouw/opmaak/indeling.

In de huidige situatie worden de klantbestanden na aanlevering geïmporteerd in de systemen van Artena door middel van SSIS (SQL Server Integration Services) packages. Deze packages voeren een aantal handelingen uit op de klantbestanden waardoor de data in die bestanden gereed wordt gemaakt voor de systemen/applicaties van Artena.

Een klantbestand bestaat uit 0 of meer rijen met daarin verschillende soorten gegevens. Bijvoorbeeld deelnemerbestanden (een deelnemerbestand is een type klantbestand) bevat één rij persoonsgegevens van een werknemer met daarbij bijvoorbeeld financiële gegevens. Het komt erop neer dat één rij altijd 0 of meerdere velden bevat met per veld een verschillende invulling.

2.1 Functionele eisen

Per functionele eis in dit hoofdstuk is aangegeven van welke stakeholder deze eis afkomstig is. Daarnaast worden de functionele eisen beschreven in dit hoofdstuk worden ingedeeld naar prioriteit. Om de eisen naar prioriteit in te delen zal gebruik gemaakt worden van de MoSCoW methode. Met behulp van deze methode wordt het mogelijk om de eisen in te delen in de volgende categorieën:

1. **Must have:** deze eisen worden in ieder geval binnen de gemaakte timebox/planning verwerkt in het systeem
2. **Should have**: deze eisen zijn net zo belangrijk als de must have eisen. Echter is het voor deze eisen niet verplicht om dat binnen vastgelegde timebox te doen.
3. **Could have:** een dergelijke eis is gewenst maar niet noodzakelijk, deze eis kan meegenomen als tijd en kosten dat toelaten.
4. **Won't have**: deze eisen zullen niet worden verwerkt in het te implementeren systeem.

Bovenstaande categorieën zijn voorgelegd aan de stakeholders terwijl door de eisen is heen gelopen. Per eis is vervolgens een prioriteit toegekend.

LEGENDA

Nr.	= nummer van de eis
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Eis	= de eis zelf

De eisen hieronder zijn genummerd zodat er vanuit andere documenten (zoals het ontwerprapport) kan worden terugverwezen naar de eisen.

Zoals te zien is bevatten onderstaande eisen veel gegevens over de in –en uitvoer van het conversiesysteem. Deze (technische) eisen zijn van nu al belang om de geëiste flexibiliteit af te bakenen.

Nr.	St.	Pr.	Eis
F1	S1, S2	1	Invoer van conversiesysteem: • Één of meerdere klantbestanden **{vervolg in F2, F3, F4}** • Één of meerdere verwijzingen naar de templates waarin de benodigde gegevens voor een conversie van een klantbestand staan **{vervolg in F5}**
F2	S2	1	Soorten klantbestanden te ondersteunen door conversiesysteem: • Excel (xls of xlsx) • CSV Het aantal ondersteunde bestandsformaten is flexibel **{zie NF3}**
F3	S3	2	Het conversiesysteem moet de conversie van XML klantbestanden uit kunnen voeren wanneer deze genoeg gegevens bevatten en complex genoeg zijn zodat kan worden voldaan aan de uitvoer van het conversiesysteem **{vervolg in F6}.**

F4	S2	1	Structuur klantbestanden voor invoer conversiesysteem:
			• Een bestand bestaat uit één of meerdere tabellen
			  ○ Een tabel bestaat uit één of meerdere kopregels plus nul of meerdere rijen
			• Structuur rij in een tabel:
			  ○ (xls/xlsx) nul of meerdere velden
			  ○ (csv) nul of meerdere velden gescheiden door een scheidingsteken
			• Een veld heeft een datatype:
			  ○ Datum of getal of double of string

F5	S2	1	Gegevens benodigd in een template om conversie uit te voeren: • Aantal tabellen aanwezig in bestand (minimaal één) • Bestandstype (xls of xlsx of csv)

F6	S2	1	Uitvoer van een conversieproces (de import):
			• Database met daarin:
			  ○ Naam klantbestand
			  ○ Nul of meerdere rijen uit klantbestand **{vervolg in F7}**

F7	S2	1	Kenmerken van een rij in uitvoer van een conversie: • Nummer van de tabel • Nummer van de rij • Nul of meerdere velden gescheiden door een komma (volgens de RFC-4180 standaard[1])
F8	S2	1	Formaten van uitvoervelden van xls/xlsx conversies moeten voldoen aan onderstaande definities, zodat dit gelijk zal zijn: 1. Datum – ‘dd-mm-yyyy hh:mm:ss’ 2. Integer – ‘xxxxxxx’(geen duizendtal separatoren) 3. Double – ‘xx.xx’ (geen duizendtal separatoren & gescheiden door punt i.p.v. komma)

[1] http://tools.ietf.org/html/rfc4180

F9	S2, S3	2	Het conversiesysteem moet in staat zijn conversies voor bestanden van verschillende klanten parallel aan elkaar uit te voeren.
F10	S2	1	Indien zich fouten voordoen tijdens een conversieproces (=één run van een conversie) moet het conversiesysteem de gegevens/kenmerken van deze fout kunnen vastleggen/loggen.
F11	S2	2	Het conversiesysteem moet in staat zijn het bestandsformaat van het klantbestand automatisch te bepalen aan de hand van de inhoud van de meegeleverde template (of extensie) wanneer een klantbestand is geselecteerd.
F12	S2	3	Het opstartmechanisme van het conversiesysteem moet gebaseerd zijn op een systeem dat een nieuwe aangeleverd klantbestand detecteert.

2.2 Niet-functionele eisen

In deze paragraaf worden de niet-functionele eisen (ook wel kwaliteitseisen genoemd) beschreven. De niet-functionele eisen beschrijven vaak extra eisen aan de wijze waarop bepaalde functionaliteiten uitgevoerd worden door het systeem. Een complete lijst van soorten niet-functionele eisen is te vinden in de ISO-9126 standaard[2]

De categorieën gebruikt in de niet-functionele eisen:

1. Functionality compliance
2. Changeability

LEGENDA

Nr.	= nummer van de eis
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Cat.	= categorie van de niet-functionele eis (ISO-9126)
Eis	= de eis zelf

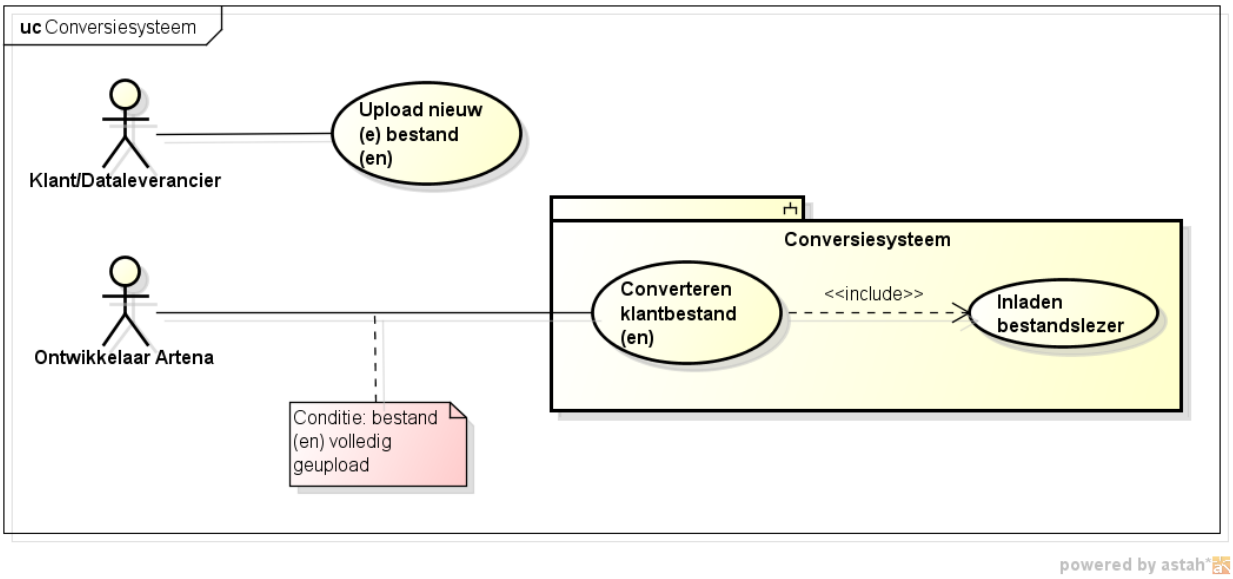
Nr.	St.	Pr.	Cat.	Eis
NF1	S2	1	1	De uitvoer van een conversie voldoet zoals in **{F7}** beschreven aan de RFC-4180[3] standaard voor Comma Separated Values. Dit houdt in dat: • Individuele velden in een rij zijn omgeven door dubbele quotes (''item'') • Individuele velden zijn van elkaar gescheiden door een komma (,) • Het laatste veld wordt niet afgesloten met een komma
NF2	S2	1	1	Het conversiesysteem moet de ASCII standaard[4] volledig ondersteunen. Dit houdt in dat het systeem de tekens uit deze karakterset kan herkennen en verwerken.
NF3	S2	1	2	De flexibiliteit van het conversiesysteem moet zodanig zijn dat een ondersteuning van een nieuw bestandsformaat geen wijzigingen aan de code van het conversieproces te weeg moet brengen, de wijze waarop het conversieproces omgaat met verschillende soorten bestanden zal dus generiek moeten zijn.

3. Use cases (UML)

[2] http://www.testconsultancy-groep.nl/index.php/downloads - zie ISO-9126 toegelicht
[3] http://tools.ietf.org/html/rfc4180
[4] http://ascii-table.com/ansi-codes.php

UML use cases worden gebruikt om aan te geven welke interacties bepaalde actoren zullen hebben met het conversiesysteem. Met behulp van de use case beschrijvingen is het mogelijk om bepaalde functionele eisen te specificeren. §3.1 toont het use case diagram van toepassing op het conversiesysteem. In §3.2 worden per use case de use case beschrijvingen gegeven.

3.1 Use case diagram



Figuur 1: Use case diagram

3.2 Use case beschrijvingen

De eerste interactie, waarbij de klant een (deelnemer)bestand upload naar een ftp-server is momenteel al geïmplementeerd. Deze manier wordt dus nu al gebruikt om data uit te wisselen tussen klanten van Artena en Artena zelf.

Naam	Upload nieuw(e) bestand(en)
Aannamen	De klant heeft de juiste autorisatie om het bestand te kunnen uploaden en is in bezit van eventueel benodigde gebruikersnaam en wachtwoord
Beschrijving	(1) De klant logt op de (FTP) server of andere service **[I]** (2) De klant selecteert het te uploaden bestand (3) De klant start het uploaden van het bestand
Uitzondering	**[I]** FTP server niet bereikbaar of offline waardoor het bestand niet geüpload kan worden, de klant neemt dan waarschijnlijk contact op met Artena.
Resultaat	Het (deelnemer)bestand van de klant dat door Artena verwerkt moet worden is aanwezig op de juiste FTP locatie.

De twee andere interacties hebben betrekking op het uitvoeren van acties waardoor het aangeleverde bestand van de klant geselecteerd en geconverteerd kan worden door de ontwikkelaar van Artena.

*Hierbij is belangrijk te vermelden dat onderstaande interacties voor deze versie van het conversiesysteem handmatig gestart gaan worden vanuit een (user) interface. Het uiteindelijke doel is om ervoor te zorgen dat het systeem zelf een conversie activeert wanneer een nieuw bestand naar locatie is gekopieerd (functionele eis **F12**).*

Naam	Converteren klantbestand(en)
Aannamen	De ontwikkelaar van Artena weet de locatie van het/de klantbestand(en) en de template(s) die daarbij hoort(horen)
Beschrijving	(1) De ontwikkelaar selecteert de juiste bestand(en) en de juiste template(s) (2) De ontwikkelaar start de conversie m.b.v. een button in een interface **[I]** (3) Het conversiesysteem selecteert de juiste bestandslezer voor het bestand - **Use case 'Inladen bestandslezer'** (4) De bestandslezer leest de regels van het bestand in en voert afhankelijk van het bestandsformaat de noodzakelijke manipulaties uit. (5) Het conversiesysteem schrijft de geconverteerde bestandsregels weg naar de database is het gewenste formaat **[II]**
Uitzondering	**[I]** Het systeem herkent de ingevoerde template niet, waardoor het conversieproces wordt afgebroken en een systeemfout wordt getoond aan de ontwikkelaar **[II]** Databasefout (*kan geen verbinding maken met database*), waardoor het conversieproces wordt afgebroken eindigt en een systeemfout wordt getoond aan de ontwikkelaar
Resultaat	De data aanwezig in het geselecteerde bestand is geconverteerd naar de betreffende database.

Naam	Inladen bestandslezer
Aannamen	Alle gegevens die benodigd zijn om een correcte bestandslezer te vinden en in te laden zijn bekend
Beschrijving	(1) Het systeem zoekt de locatie van de bestandslezer op **[I]** (2) Het systeem laadt de bestandslezer in het conversiesysteem en zorgt ervoor dat er mee gecommuniceerd kan worden
Uitzondering	**[I]** De locatie van de bestandslezer is niet beschikbaar/bereikbaar, waardoor een fout wordt gegenereerd en getoond aan de ontwikkelaar en het conversieproces wordt afgebroken.
Resultaat	Een nieuw ingeladen bestandslezer die gebruikt kan worden voor het conversiesysteem

4. Onderzoek bestandslezers

Zoals beschreven in de functionele eisen (§2.1) zal het conversiesysteem meerdere bestandsformaten moeten kunnen ondersteunen. Momenteel zijn dat in ieder geval Excel97-2003 bestanden, Excel2007 bestanden en CSV bestanden.

Alvorens het ontwerp van de het conversiesysteem wordt opgesteld is een onderzoek uitgevoerd naar de verschillende mogelijkheden tot het uitlezen van Excel of CSV bestanden. Het onderzoek is beschreven in één document (**Onderzoek bestandslezers versie 1.1**).

De volgende resultaten (conclusies) zijn getrokken uit het onderzoek:

1. Voor het uitlezen van Excel bestanden (zowel 97-2003 als 2007) zal gebruik worden gemaakt van de C# softwarebibliotheek *Excel Data Reader for .NET.*
2. Voor het uitlezen van CSV bestanden zal gebruik worden gemaakt van een eigen implementatie aangezien .NET meerdere mogelijkheden heeft tot het uitlezen van dit soort bestanden.

Deze conclusies zijn getrokken in het onderzoek document en zijn vervolgens bevestigd door de uitvoering van enkele unit-tests op hierboven beschreven bestandslezers. Deze unit-tests zijn beschreven in de documenten '**Unit Test – Excel File Reader**' en '**Unit Test – CSV File Reader**'.

De conversie zal dus niet worden uitgevoerd met SSIS packages maar met een implementatie in C#, waarin in ieder geval ondersteuning wordt gegeven voor het uitlezen van Excel –en CSV bestanden. In het ontwerp zal hiermee rekening gehouden moeten worden.

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage F
Onderzoek bestandslezers

Versie 1.1

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 17-02-2012

Documentbeheer

Versie	Datum	Status	Approvedby	Omschrijving
0.1	14-02-2012	Concept		Initiële versie document Toegevoegde onderdelen: - Inleiding - Te ondersteunen bestandsformaten - Opstart Excel onderdeel Excel onderdeel 90% af: - Vijf verschillende mogelijkheden - Diagrammen To do: Conclusie m.b.t. voorkeur Excel conversie
1.0	16-02-2012			Toegevoegd: - CSV sectie
1.1	17-02-2012			Bijgewerkt: - Begrippenlijst

Inhoudsopgave

1. Inleiding 4
2. Te ondersteunen bestandsformaten 5
2.1 Excel 5
2.2 CSV 5
3. SSIS of C#? 5
4. Conversie Excel bestanden 6
4.1 C# en .NET 6
4.1.1 OLEDB Jet Engine 4.0 7
4.1.2 OLEDB ACE Engine 12.0 7
4.1.3 Office Interop Excel Object 8
4.1.4 Excel Data Reader (open-source .NET bibliotheek) 8
4.2 SSIS 9
4.3 Conclusie 9
5. Conversie CSV bestanden 10
5.1 Formats 10
5.2 SSIS 10
5.3 C# en .NET 10
5.4 Conclusie 11
6. Begrippenlijst 12

1. Inleiding

Alvorens een ontwerp opgesteld kan worden op basis van de functionele en niet-functionele requirements moet een onderzoek worden uitgevoerd naar de mogelijke oplossingen tot uitlezen van de te ondersteunen bestandsformaten.

Dit document beschrijft de verschillende mogelijkheden per te ondersteunen bestandsformaat met hun voor –en nadelen en beschrijft daarnaast de uiteindelijke keuze voor een oplossing (inclusief onderbouwing).

Deze keuze kan worden gezien als een voorkeur voor de betreffende oplossing. De gekozen oplossing moet dan eerst nog beter getest worden op verschillende foutgevoelige data (zoals datum/tijd notaties enzovoorts…). Deze (unit)tests zullen in andere documenten beschreven worden.

2. Te ondersteunen bestandsformaten

2.1 Excel

Te ondersteunen varianten:

- 97-2003 .XLS
  - Dit bestandformaat is van het binaire type, dat wil zeggen dat de data, opmaak en andere gegevens binair worden verwerkt. Het binaire file format dat gebruikt wordt bij dit bestandtype is gebaseerd op het Binary Interchange File Format (BIFF)

- 2007 .XLSX
  - Sinds de invoer van Office 2007 wordt voor Excel bestanden de Open Office XML (OOXML) specificatie gebruikt. Een .XLSX bestand bestaat uit een gecomprimeerd, op XML gebaseerd, bestand. De OOXML standaard maakt tegenwoordig ook onderdeel uit van de ISO norm.

- 2010 .XSLX
  - Het bestandsformaat van Office 2010 is van precies hetzelfde format als dat van Office 2007, echter is bij de invoer van deze versie ondersteuning voor meer varianten van het OOXML formaat toegevoegd.[1]

2.2 CSV

CSV bestanden zijn altijd van de bestandsextensie *.csv*. Intern kan het formaat van CSV bestanden wel verschillen doordat er verschillende scheidingstekens gebruikt kunnen worden (zie §5.1).

3. SSIS of C#?

Voor de conversie zijn momenteel twee oplossingen bekend om de aangeleverde data te converteren. Het is mogelijk om SSIS te gebruiken om de verschillende bestanden te converteren naar een database of één type bestandsformaat. Het is echter ook mogelijk om gebruik te maken van de .NET libraries en daarmee conversiesysteem te implementeren.

Momenteel worden bij Artena vele SSIS implementaties gebruikt om binnenkomende data in te voeren in de systemen en databases van Artena. Deze implementaties verschillen structureel per klant.

In het .NET framework zitten een aantal mogelijkheden om verschillende soorten bestanden uit te lezen en te verwerken. Deze mogelijkheden hebben verschillende voor –en nadelen en zullen dus onderzocht moeten worden.

In de volgende hoofdstukken wordt per type bestandsformaat dat aangeleverd wordt door klanten een beschrijving gegeven van de mogelijkheden tot conversie van deze bestanden.

[1] http://en.wikipedia.org/wiki/Office_Open_XML

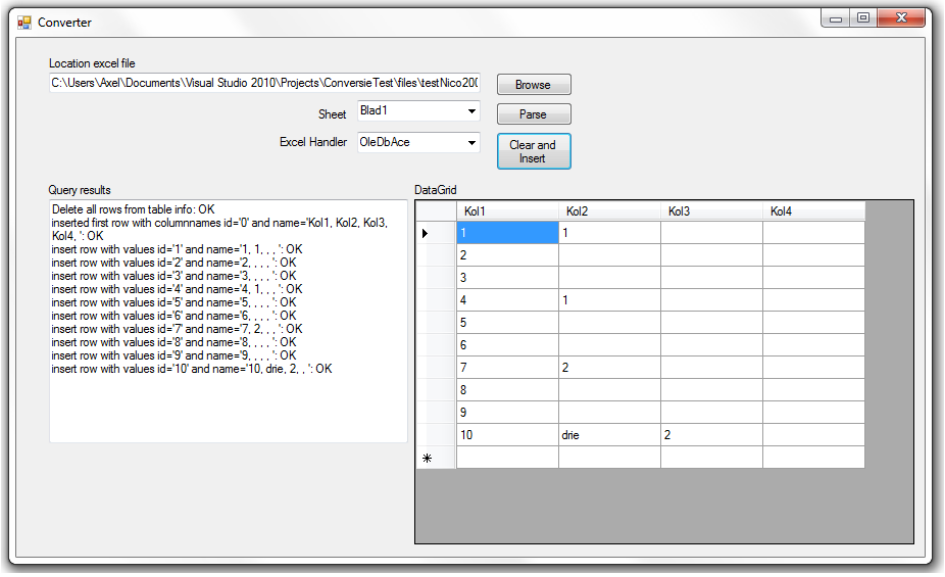
4. Conversie Excel bestanden

4.1 C# en .NET

Allereerst zijn de mogelijkheden met betrekking tot conversie van Excel bestanden uitgezocht. Om de mogelijkheden inclusief hun voor –en nadelen te onderzoeken is eerst een testapplicatie opgezet in C# met behulp van Visual Studio 2010.

De testapplicatie bevat onderstaande functionaliteiten:

- Zoeken naar gewenste Excel bestand
- Weergeven van de gehele inhoud van het Excel bestand in de applicatie
- Weergeven van eventuele kenmerken van de ingelezen cellen
- Wegschrijven van de inhoud van de Excel bestanden naar een lokale SQL Server CE database.



Figuur 1: Interface testapplicatie

In figuur 1 is te zien dat een Excel bestand is ingelezen en dat de inhoud wordt weergeven in de 'tabel' rechtsonder in het venster.

De knop 'ClearandInsert' slaat de inhoud van het Excel bestand allereerst in een *runtime object* (een *DataSet van .NET*) op waarna op dit object onderstaande bewerkingen worden uitgevoerd:

- Alle kolomnamen uit het Excel bestand worden achter elkaar neergezet en van elkaar gescheiden door een komma. Bijvoorbeeld de kolomnamen van Figuur 1:
  - "Kol1, Kol2, Kol3, Kol4"
- De inhoud van alle rijen worden individueel één voor één achter elkaar gezet en gescheiden door een komma. Voorbeeld uit Figuur 1:
  - "1, 1,, "
  - "2, , , ,"
  - Enzovoorts

Vervolgens wordt de data van de kolomnamen ingevoerd in één kolom op de lokale database. Tenslotte wordt de data van de rijen individueel ingevoerd in dezelfde kolom in de lokale database, zoals in tabel 1 is te zien.

ID	Data
1	Kol1, Kol2, Kol3, Kol4
2	1, 1, ,
3	2, , ,

Tabel 1: Selectie van database na wegschrijven data uit Figuur 1

Er zijn vier manieren geïmplementeerd om de data uit het Excel bestand te lezen. Elk van deze manieren heeft zijn eigen voordelen en nadelen, welke in de volgende paragrafen beschreven zijn.

4.1.1 OLEDB Jet Engine 4.0

De Jet (Data) Engine is een OLE DB provider van Microsoft. Een OLE DB provider is een software component waardoor een gebruiker kan communiceren met een bepaalde databron.[2] De gebruiker kan via de interfaces van een OLE DB provider een databron uitlezen, wijzigen of aanmaken.

De Jet Engine is een provider waarmee onder andere data van Access of Excel bestanden uitgelezen kan worden[3]. Met behulp van de 'connectie' die m.b.v. deze engine met het bestand gemaakt kan worden, kan de gehele inhoud van het bestand worden uitgelezen en deze inhoud worden opgeslagen in een *DataSet*.

Nadelen Jet Engine:

- Niet mogelijk om Excel 2007 bestanden te gebruiken(.xlsx)
- Standaard worden de eerste acht rijen van elke kolom uit het Excel bestand doorlopen om zo het 'datatype' (getal, tekst, booleaans waarde) van de hele kolom te bepalen.
    - Het risico hiervan is dat in een kolom waarbij de eerste acht rijen een *integer* of *double* bevatten en waarbij de tiende rij een tekstuele waarde of *string* bevat dat deze string waarde wordt weggelaten door de Jet Engine.

Workaround:

- Het is mogelijk om in te stellen dat niet de eerste acht rijen worden doorlopen, maar dat de alle rijen worden doorlopen.
    - Hiervoor moet een waarde worden gewijzigd in het register van Windows 7: *HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Jet\Engines\Excel\TypeGuessRows*

Door deze workaround worden alle waarden uit het Excel bestand weergeven, echter wordt het datatype van de laatst gelezen waarde aangehouden voor de gehele kolom.

4.1.2 OLEDB ACE Engine 12.0

De Access Connectivity Engine is de opvolger van de Jet Engine. Deze engine ondersteund de nieuwste bestandtypes van Office en is op vrijwel dezelfde manier te gebruiken als de Jet Engine.

[2] http://en.wikipedia.org/wiki/OLE_DB_provider
[3] http://msdn.microsoft.com/en-us/library/windows/desktop/ms709836(v=vs.85).aspx

Voordeel ACE Engine:

- Ondersteuning voor zowel Excel 97-2003 als Excel 2007 en Excel 2010 bestanden (.XLS en .XLSX)

Nadelen ACE Engine:

- Hetzelfde nadeel als de Jet Engine wat betreft de datatypering van kolommen
  - Workaround net zoals de Jet Engine, maar een andere locatie van het registersleutel: *HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Office\12.0\Access Connectivity Engine\Engines\Excel\TypeGuessRows*
- Op verscheidene websites worden bugs beschreven waarbij het uitvoeren van *query's* met *UNIONS* op de datasets gegenereerd door de ACE Engine fouten oplevert[4]

4.1.3 Office Interop Excel Object

Het Microsoft Interop Excel Object maakt deel uit van de Excel Object Model, waarmee de gebruiker via een aantal objecten kan communiceren met een Excel bestand. Deze objecten zijn:

- Microsoft.Office.Interop.Excel.Application, waarmee een daadwerkelijk Excel applicatie wordt opgestart.
- Microsoft.Office.Interop.Excel.Workbook, waarmee een Excel bestand vanuit de gestarte Excel applicatie kan worden geopend
- Microsoft.Office.Interop.Excel.Worksheet, waarmee binnen het workbook een sheet kan worden geselecteerd
- Microsoft.Office.Interop.Excel.Range, waarmee een 'range' aan cellen binnen de worksheet kan worden uitgelezen en bewerkt

Met behulp van deze objecten was het mogelijk om een sheet uit te lezen. Echter waren er twee grote nadelen:

- Na het afsluiten van de applicatie bleef een Excel proces op de achtergrond draaien, zelfs nadat het applicatie object specifiek was afgesloten.
- Het inladen van de sheet en de data in de sheet was aanzienlijk trager dan de andere geteste implementaties. Dit heeft waarschijnlijk te maken met het feit dat de communicatie moet lopen via het Excel applicatie object.

4.1.4 Excel Data Reader (open-source .NET bibliotheek)

De Excel Data Reader is een kleine open-source .NET bibliotheek waarmee Excel bestanden uitgelezen kunnen worden in C#. De bibliotheken zijn gratis te downloaden[5] en toe te voegen als referentie aan Visual Studio 2010.

Het grootste voordeel van deze bibliotheek is dat standaard alle waarden uit het Excel bestand omgezet worden naar een string waardoor de kans op fouten bij de conversie geminimaliseerd worden. Immers, alle soorten data (integers, doubles, *booleanse waarden*…) kunnen opgeslagen worden in een string.

[4] http://www.mrexcel.com/forum/showthread.php?t=546394
[5] http://exceldatareader.codeplex.com/

Voordelen:

- Snelheid (zoals aangegeven op website)
- Gebruikt geen OLEDB of andere data communicatie providers van Microsoft, waardoor problemen zoals de verkeerde datatypering voorkomen kunnen worden.
- Compatibel met Office 97, Office 2007 en Office 2010 bestanden
- Alle waarden in een Excel document zijn van het datatype *string* nadat zij zijn ingelezen in de applicatie.

Nadelen:

- Mogelijke fouten met booleaanse waarden en lege velden
    - Volgens het forum van de website van de Excel Data Reader

4.2 SSIS

Naast de mogelijkheid tot conversie van Excel bestanden met behulp van C# is het ook mogelijk om de conversies uit te laten voeren met behulp van SSIS. Bij sommige bestanden worden deze SSIS transformaties momenteel al gebruikt om de bestanden in te laden in de databases van FinCircle.

Een groot nadeel van de SSIS versie die gebruikt wordt (namelijk diegene die wordt meegeleverd met SQL Server 2008) is dat het systeem onderliggend ook werkt met een OLE DBprovider (zowel de Jet als de ACE provider kan gebruikt worden).

De nadelen die bij het gebruik van de OLE DB providers in de testapplicatie uit hoofdstuk 3.1 aan het licht kwamen zijn dus ook aanwezig wanneer de conversie uitgevoerd zal worden met behulp van SSIS.

Het voordeel van SSIS is natuurlijk dat het gemakkelijker is om een dergelijke conversie op te stellen, omdat gewerkt kan worden met de grafische user interface van SSIS, dan wanneer de software voor de conversie eigenhandig geïmplementeerd moet worden. Dat levert tijdwinst op.

4.3 Conclusie

Op basis van de beschreven voor –en nadelen van alle mogelijkheden kan een voorkeur voor een bepaalde oplossing om Excel bestanden uit te lezen uitgesproken worden. Om Excel bestanden uit te lezen ligt de voorkeur bij de Excel Data Reader.

- De OLE DB Jet Engine valt af vanwege de manier waarop datatypes worden bepaald en het feit dat deze engine niet compatibel is met Excel97-2003 bestanden.
- De OLE DB ACE Engine valt af om de dezelfde reden als de Jet Engine, de datatypering. Daarbij zijn er ook meldingen gemaakt van fouten bij sommige query's.
- De Excel Interop oplossing valt ook af omdat dit systeem zeer traag is in vergelijking met de andere oplossingen. Ook ontstonden er enkele problemen met Excel processen.
- De SSIS oplossing valt automatisch af, omdat SSIS gebruik maakt van de OLE DB Jet Engine of de OLE DB ACE engine.

Voordat een definitieve keuze gemaakt kan worden, zullen eerst enkele tests uitgevoerd worden met de Excel Data Reader.

5. Conversie CSV bestanden

5.1 Formats

Volgens RFC-4180 van het IETF is er een beperkt aantal mogelijkheden voor de opmaak van een Comma Separated Values bestand. Het voorbeeld dat wordt aangehaald in het RFC:

- aaa, bbb, ccc CRLF (standaard)
- "aaa","bbb","ccc" CRLF (dubbele quotes om values)
- "aaa","b""bb","ccc" CRLF (wanneer value (b"bb) dubbele quotes bevat)

CRLF staat voor een end-of-line of een line-break, waarmee wordt aangegeven dat het einde van de regel/rij is bereikt.

Het komt ook voor dat in plaats van komma's puntkomma's worden toegepast om de verschillende waarden te scheiden. Dit is niet volgens de *RFC-standaard* maar deze manier van scheiding wordt wel gebruikt door klanten van Artena. Het kan ook gebeuren dat op het einde van een regel een puntkomma of komma voor de CRLF is toegevoegd:

- aaa; bbb; ccc; CRLF (puntkomma gescheiden en puntkomma op het einde)

Het conversiesysteem moet rekening kunnen houden met alle bovenstaand beschreven mogelijkheden.

5.2 SSIS

Omdat het converteren van Excel-bestanden hoogstwaarschijnlijk gaat gebeuren met behulp van een zelf geschreven converter is het niet handig om het inlezen van CSV bestanden wel te doen met SSIS.

Het is beter als alle conversies van alle te ondersteunen formaten in hetzelfde systeem plaatsvinden, omdat dit onder andere de begrijpbaarheid en eenvoud van het conversiesysteem bevordert.

5.3 C# en .NET

In het .NET framework zit een goede mogelijkheid om .CSV bestanden uit te lezen omdat een .CSV bestand eigen een eenvoudig tekstbestand is.

Met behulp van een *StreamReader* en het opsplitsen van een string op bepaalde scheidingstekens (zoals een komma of puntkomma) kunnen individuele velden per rij uitgelezen en opgeslagen worden.

In het prototype (testapplicatie) is dit dan ook met succes uitgevoerd. De data kan vanuit een .CSV bestand worden ingelezen in een DataSet om deze vervolgens op het scherm te tonen of naar een database weg te schrijven.

In het testbestand, een deelnemer bestand van de Pensioenfond De Eendracht, zijn waarden gescheiden door een puntkomma en wordt de laatste waarde op een regel/rij afgesloten door een puntkomma:

- aaa; bbb; ccc;

De eerste regel/rij wordt bevat de 'kolomnamen'.

5.4 Conclusie

Uit de resultaten is te concluderen dat SSIS niet zal worden gebruikt bij de implementatie van het conversiesysteem.

Door de implementatie af te handelen door een eigen implementatie in C# kan de conversie worden van alle te ondersteunen bestandsformaten worden afgehandeld door één systeem.

Er zijn veel mogelijkheden aanwezig in .NET om een CSV bestand uit te lezen. Op de implementatie moeten nog wel (unit)tests worden uitgevoerd om vast te stellen dat alle mogelijke oorzaken voor problemen afgevangen zijn.

Het is wel de bedoeling dat zoveel mogelijk data op dezelfde manier eruit komt te zien als uit de Excel bestanden (dus bijvoorbeeld punten i.p.v. komma's om decimale getallen te scheiden).

6. Begrippenlijst

- Runtime Object
  - *Object dat tijdens de uitvoer van de applicatie wordt aangemaakt, dus runtime*
- DataSet
  - *Een DataSet is een .NET object waarin meerdere DataTables opgeslagen kunnen worden, waarin meerdere rijen en kolommen opgeslagen kunnen worden*
- Integer
  - *Een .NET Value Type, waarin getallen opgeslagen kunnen worden met een domein van* -2,147,483,648 to 2,147,483,647[6]
- Double
  - *.NET Value Type voor getallen met één of meer decimalen*[7]
- String
  - *.NET Reference Type waarin nul of meerdere Unicode karakters opgeslagen kunnen worden*[8]
- Query (query's)
  - *Opdracht die aan een database kan worden gegeven zodat de database deze kan uitvoeren en het resultaat kan retourneren of weergeven*
- UNION
  - *Onderdeel van een query waarin twee rijen aan elkaar gekoppeld kunnen worden op basis van nul of meerdere kolommen uit nul of meerdere tabellen. Dubbele waarden worden standaard weggelaten in het resultaat*
- Booleaanse waarde
  - *In c#: bool. Bevat waarde 0 (false) of waarde 1(true)*
- CLRF
  - *Carriage Return Line Feed, zorgt voor het eindigen van de huidige regel en het invoeren van een nieuwe regel*
- RFC-standaard
  - *Request for Comments, waarin eigenschappen of notities over het internet of IT gerelateerde onderdelen staat beschreven. Een RFC wordt een standaard waneer het IETF dat beslist en wanneer er genoeg interesse is in een standaard.*
- StreamReader
  - *.NET object waarmee bestanden uitgelezen kunnen worden door stromen van bytes (streams) met behulp van een bepaalde tekenset (encoding) uitgelezen kunnen worden.*

[6] http://msdn.microsoft.com/en-US/library/5kzh1b5w(v=vs.80).aspx
[7] http://msdn.microsoft.com/en-US/library/678hzkk9(v=vs.80).aspx
[8] http://msdn.microsoft.com/en-us/library/362314fe(v=vs.80).aspx

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage G

Unit Test – Excel File Reader

Iteratie 1: Conversiesysteem

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 21-02-2012

Inhoudsopgave

1. Inleiding 3
2. Testcases 4
2.1 Testcase 1: Verschillende veldtypen 4
2.1.1 Uit te voeren acties 4
2.1.2 Verwachte resultaten 5
2.2 Testcase 2: Converteren groot Excel bestand 5
2.1.1 Uit te voeren acties 5
2.1.2 Verwachte resultaten 6
3. Resultaten 6
3.1 Testcase 1, Excel 2007 6
3.2 Testcase 1, Excel 97-2003 6
3.3 Testcase 2 7
4. Conclusie 7

1. Inleiding

In het document **'Onderzoek bestandslezers'** is aangegeven dat voor het uitlezen van Excel bestanden gebruik gemaakt zal worden van de *Excel Data Reader for .NET* bibliotheek.

Alvorens een definitieve keuze gemaakt kan worden moet deze oplossing eerst uitvoerig getest worden, zodat het zeker is dat er geen fouten gemaakt worden bij het uitvoeren van de conversie.

In dit document worden een aantal testcases vastgelegd in de vorm van Excel spreadsheets. Elk van deze spreadsheets zal op een juiste en consistente manier ingelezen moeten worden door de Excel Data Reader om de data vervolgens te converteren naar een database. De resultaten van deze tests zullen ook worden vastgelegd in dit document.

Deze test kan worden gezien als een unit test omdat de Excel File Reader interface in de testapplicatie getest wordt op het verwerken van verschillende Excel formats.

2. Testcases

Gegevens uit een Excel bestand kunnen worden uitgelezen en weergeven in een tabel in de applicatie (inclusief datatypen per veld) of kunnen worden weggeschreven naar een SQL Server database (tabel deelnemer_info met kolommen 'ID' en 'data').

2.1 Testcase 1: Verschillende veldtypen

Deze case behandelt de verschillende soorten formaten die per veld (cel) ingesteld kunnen worden in een spreadsheet. Zo zijn er verschillende soorten getallen of valuta's. In Tabel 1 staan de te ondersteunen typen inclusief voorbeeld, de voorbeelden zullen ook gebruikt worden in de daadwerkelijke testspreadsheet.

Voor de getallentest zal het getal 1000345,0456789 gebruikt worden.
Voor de datumtest zal de datum 12-01-2012 gebruikt worden.
Voor de percentagetest zal het percentage 69,051 gebruikt worden.

Type veld	Extra instellingen	Weergave in Excel
Standaard	Geen	1000345,046
Getal	Geen	1000345,05
Getal	8 decimalen	1000345,04567890
Getal	Scheidingsteken duizendtallen	1.000.345,05
Valuta	€ -teken	€ 1.000.345,05
Valuta	Geen teken	1.000.345,05
Financieel	€ -teken	€ 1.000.345,05
Tekst	Geen	1000345,046
Datum		12-1-2012
Datum		12-01-12
Datum		donderdag 12 januari 2012
Datum		12 januari 2012
Datum		12-jan-2012
Datum	Locatie: English (V.S)	1-12-2012
Datum	Locatie: English (UK)	12 January 2012
Percentage		69,05%
Percentage	4 decimalen	69,0510%

Tabel 1: Te testen veldtypen

Van de gegevens in Tabel 1 zijn twee Excel spreadsheets gemaakt:

- Excel 97-2003 (.xls)
- Excel 2007 (.xlsx)

Deze zijn bijgevoegd in Bijlage A en Bijlage B.

2.1.1 Uit te voeren acties

1. Start testapplicatie
2. Zoek naar het testbestand van excel2007
3. Klik 'Parse'
4. Leg inhoud va *DataGridView* vast
5. Klik 'Clear and Insert'
6. Controleer of juiste inhoud naar database is weggeschreven.

7. Sluit applicatie

1. Start testapplicatie
2. Zoek naar het testbestand van excel97
3. Klik 'Parse'
4. Leg inhoud va *DataGridView* vast
5. Klik 'Clear and Insert'
6. Controleer of juiste inhoud naar database is weggeschreven
7. Sluit applicatie

2.1.2 Verwachte resultaten

Getallentest:

- De getallen worden consistent vertaald, dit wil zeggen: 10003245,04(56789)
- Duizendtal scheidingstekens worden weggelaten
- Speciale tekens (zoals €) worden weggelaten

Datum:

- Alle data krijgen dezelfde uitvoer: 12-(0)1-2012

Percentage:

- Gegevens worden weergeven inclusief percentages

2.2 Testcase 2: Converteren groot Excel bestand

De tweede testcase behelst het inladen en wegschrijven van een zeer groot Excel bestand. Voor deze case zal een Excel bestand gebruikt worden van één van de klanten van Artena (Ahold). Dit bestand mag om privacy –en veiligheidsredenen niet worden bijgevoegd.

Kenmerken gebruikte bestand:

- Aantal rijen: 25508
- Aantal kolommen: 76

2.1.1 Uit te voeren acties

1. Start testapplicatie
2. Zoek naar het testbestand van Ahold
3. Klik 'Clear and Insert'
    a. ('Parse is zinloos omdat het wegschrijven naar de GUI te lang duurt en omdat het niet zinvol is om via de GUI alle data langs te lopen)
4. Wacht tot de query executie is afgerond
5. Leg het aantal rijen uit de *ChangeSet* vast (dit doet de applicatie)
6. Leg het aantal ingevoerde kolomnamen uit de *ChangeSet* vast (dit doet de applicatie)
7. Voor de query: "SELECT COUNT(data) FROM deelnemer_info" uit in SQL Server Management Studio op de betreffende database en leg het resultaat vast
8. Sluit applicatie

2.1.2 Verwachte resultaten

- Aantal rijen in ChangeSet: 25508
- Aantal rijen in database tabel 'deelnemer_info': 25508

- Aantal kolommen in ChangeSet: 76

3. Resultaten

3.1 Testcase 1, Excel 2007

	Standaard	Getal	Valuta	Financieel	Tekst	Datum	Percentage
▶							
	1000345.0456789	1000345.0456789	1000345.0456789	1000345.0456789	1000345.0456789	12-1-2012 0:00:00	0.690509999999...
		1000345.0456789	1000345.0456789			12-1-2012 0:00:00	0.690509999999...
		1000345.0456789				12-1-2012 0:00:00	
						12-1-2012 0:00:00	
						12-1-2012 0:00:00	
						12-1-2012 0:00:00	
*							

Tabel 2: Excel 2007 inlezen

- Alle getallen worden volledig overgenomen bij het uitlezen van de cellen (dus niet de afgeronde getallen)
- Decimalen worden gescheiden met een punt (.)
- Alle data worden genoteerd in de vorm van *dd-mm-yyyy uu:mm:ss*
- De percentages zijn van 0.69051 naar 0.69050999999… omgezet
    - Veranderen van celtype (bijvoorbeeld van 'percentage' naar 'getal) verandert niets.

De getoonde data in Tabel 2 komt overeen met de data aanwezig in de database (zie Bijlage C).

3.2 Testcase 1, Excel 97-2003

	Standaard	Getal	Valuta	Financieel	Tekst	Datum	Percentage
▶							
	1000345,0456789	1000345,0456789	1000345,0456789	1000345,0456789	1000345,0456789	40920	0,69051
		1000345,0456789	1000345,0456789			40920	0,69051
		1000345,0456789				40920	
						40920	
						40920	
						40920	
*							

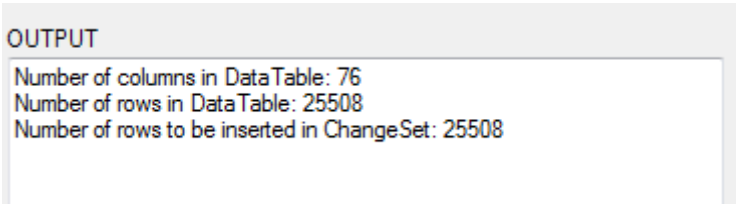
Tabel 3: Excel 97-2003 inlezen

- Alle getallen worden volledig overgenomen bij het uitlezen van de cellen (dus niet de afgeronde getallen)
- Decimalen worden gescheiden met een komma (,)
- De data worden genoteerd in de vorm van het aantal dagen vanaf 1 januari 1900
- De percentages zijn behouden t.o.v. het originele bestand

De getoonde data in Tabel 3 komt overeen met de data aanwezig in de database (zie Bijlage D)

3.3 Testcase 2

Gegevens volgens de testapplicatie, zie Figuur 1.



Figuur 1: Aantallen van ingevoerde rijen en kolommen

Query: `SELECT COUNT(data) FROM ConversionDb.dbo.deelnemer_import`

Resultaat: 25508

4. Conclusie

- De resultaten van testcase 2 tonen aan dat de Excel File Reader op een correcte wijze omgaat met grote Excel bestanden: de juiste hoeveelheid kolommen en rijen worden ingevoegd in de database.

- Excel 97-2003 bestanden worden op een correcte manier ingelezen. Het grootste verschil met Excel 2007 bestanden is dat de datum en tijd worden ingelezen als het aantal dagen van 1 januari 1900.
    - Omdat er geen verschil mag zitten in de notaties van datum en tijd tussen Excel97-2003 en Excel2007 bestanden, moet een patch worden toegepast (afkomstig van het Excel Data Reader forum).

- Ook worden getallen met decimalen bij Excel97-2003 bestanden door een komma gescheiden. Deze notatie moet omgezet worden, zodat de data uit zowel Excel97-2003 als uit Excel2007 bestanden consistent is.

- Bij Excel2007 bestanden is het grootste probleem de onjuiste omzetting van sommige decimale getallen. Het is niet de bedoeling dat het volgende systeem, de validatie-engine, deze ‘veranderde’ waarden gaat gebruiken.
    - De oorzaak van deze afwijking blijkt te liggen in het omzetten van de sommige decimale getallen naar een *string*. Echter wanneer deze strings weer worden omgezet naar een *double* (of *decimal)* krijgen ze hun oorspronkelijke waarde weer terug.

Samengevat blijkt dat de *Excel Data Reader for .NET* op juiste manier omgaat met vrijwel alle geteste mogelijkheden. Het grootste probleem, de onjuiste decimale waarden bij sommige getallen, zal geen gevolgen hebben voor de verdere validatie omdat deze *string* weergave weer omgezet kan worden naar een *double*.

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage H

Unit Test – CSV File Reader

Iteratie 1: Conversiesysteem

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 22-02-2012

Inhoudsopgave

1. Inleiding 3
2. Testomgeving 4
3. Testcases 4
3.1 Testcase 1: Data volgens RFC-4180 format 4
3.1.1 Uit te voeren acties 5
3.1.2 Verwachte resultaten 5
3.2 Testcase 2: Geen kolomnamen of decimaalconversie 5
3.2.1 Uit te voeren acties 5
3.2.2 Verwachte resultaten 5
3.3 Testcase 3: Converteren deelnemerbestand 5
3.3.1 Uit te voeren acties 6
3.3.2 Verwachte resultaten 6
4. Resultaten 6
4.1 Resultaat case 1 6
4.2 Resultaat case 2 7
4.3 Resultaat case 3 7
5. Conclusie 8

1. Inleiding

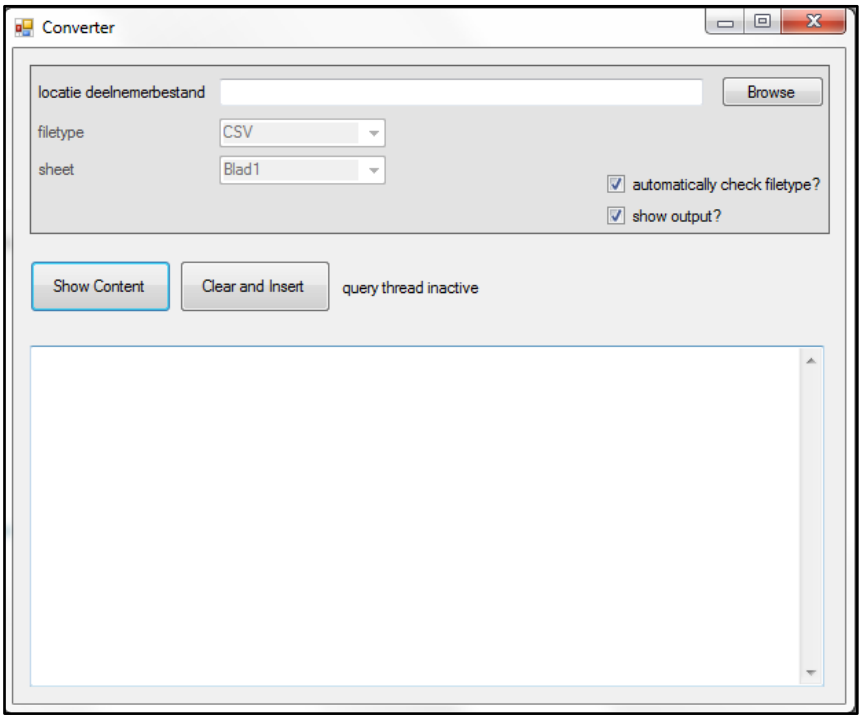
In het document **'Onderzoek bestandslezers'** is aangegeven dat voor het converteren van CSV bestanden gebruik gemaakt zal worden van een eigen implementatie in C#.

Alvorens een definitieve keuze gemaakt kan worden moet deze oplossing eerst uitvoerig getest worden, zodat het zeker is dat er geen fouten gemaakt worden bij het uitvoeren van de conversie.

In dit document worden een aantal testcases vastgelegd in de vorm van CSV documenten. De data in ieder van deze documenten moet op een juiste en consistente manier worden ingelezen. De resultaten van de uitgevoerde tests zullen ook worden vastgelegd in dit document.

2. Testomgeving

Bij de Excel Data Reader tests is gebruik gemaakt van een testapplicatie om de functionaliteit van de Excel lezer uitvoerig te testen. Voor de CSV lezer zal ook gebruik worden gemaakt van deze applicatie. Zie figuur 1 voor de grafische interface van deze testapplicatie.



Figuur 1: Grafische interface applicatie, versie 0.1

3. Testcases

Gegevens uit een CSV kunnen worden uitgelezen en weergeven in een tabel in de applicatie (inclusief datatypen per veld) of kunnen worden weggeschreven naar een SQL Server database (tabel deelnemer_info met kolommen ‘ID’ en ‘data’).

Standaard configuratie CSV reader:

- De eerste regel uit een CSV bestand wordt omgezet naar kolomnamen
- Alle decimalen gescheiden door een komma worden geconverteerd naar decimalen gescheiden door een punt

3.1 Testcase 1: Data volgens RFC-4180 format

In de requirements wordt een verwijzing gedaan naar het RFC-4180 format voor CSV bestanden:

- “aaa”,“bbb”,“ccc”

Voor deze case is een CSV testbestand gemaakt, bijgevoegd in Bijlage A. Dit bestand bevat een aantal afwijkingen waardoor de CSV lezer de fout in zou kunnen gaan:

- “aa’a”
- “aa,a”
- “aaa”,,”ccc”

3.1.1 Uit te voeren acties

1. Start conversie applicatie
2. Zoek naar het CSV testbestand
3. Klik 'Show Content'
4. In het verschenen venster is de output van de CSV lezer te controleren
5. Klik 'Clear and Insert'
6. Controleer of juiste inhoud naar database is weggeschreven.
7. Sluit applicatie

3.1.2 Verwachte resultaten

'Show Content'

- De kolomnamen en rijen bevatten allemaal de waarden zoals deze tussen dubbele quotes staan in het CSV bestand.

'Clear and Insert'

- De data in de database bevat alle waarden zoals deze tussen dubbele quotes in het CSV bestand staan, omgeven door dubbele quotes (dus hetzelfde als in het CSV bestand).

3.2 Testcase 2: Geen kolomnamen of decimaalconversie

In deze testcase zal de applicatie anders worden geconfigureerd voor het uitlezen van het testbestand. De eerste regel in het CSV bestand zal nu niet worden omgezet naar kolomnamen en de scheiding tussen getallen en hun decimalen zal nu niet worden omgezet van komma naar punt.

3.2.1 Uit te voeren acties

1. Starten applicatie
2. Configureren applicatie:
    a. Eerste rij niet als kolomnamen
    b. Decimalen gescheiden door komma niet geconverteerd
3. Zoek naar het CSV testbestand
4. Klik 'Show Content'
5. In het verschenen venster is de output van de CSV lezer te controleren
6. Sluiten applicatie

3.2.2 Verwachte resultaten

'Show Content'

- De rijen bevatten allemaal de waarden zoals deze tussen dubbele quotes staan in het CSV bestand.
- Getallen en decimalen zijn onderling niet allemaal gescheiden met een punt, maar zoals ze in het bronbestand staan.

3.3 Testcase 3: Converteren deelnemerbestand

De tweede testcase bestaat uit het inlezen van een groot deelnemer bestand. Voor deze case zal een deelnemer bestand gebruikt worden van Pensioenfonds De Eendracht. Dit bestand zal om privacy – en veiligheidsredenen niet worden bijgevoegd.

Kenmerken gebruikte bestand:

- Gebruikt niet standaard CSV formaat, waarden zijn in een rij gescheiden met een puntkomma (;) en elke rij eindigt met een puntkomma(;)
- Aantal rijen: 20569
- Aantal kolommen: 155

3.3.1 Uit te voeren acties

1. Start testapplicatie
2. Zoek naar het deelnemerbestand van De Eendracht
3. Klik 'Clear and Insert'
    a. ('Show Content' is niet zinvol omdat het zeer lang duurt de data weg te schrijven naar de GUI en omdat het veel te veel data is om m.b.v. de GUI door heen te lopen)
4. Wacht tot de query executie is afgerond
5. Leg het aantal rijen uit de *ChangeSet* vast (dit doet de applicatie)
6. Leg het aantal ingevoerde kolomnamen uit de *ChangeSet* vast (dit doet de applicatie)
7. Voor de query: "SELECT COUNT(data) FROM deelnemer_info" uit in SQL Server Management Studio uit op de betreffende database, zodat het aantal rijen in de database duidelijk wordt.
8. Sluit applicatie

3.3.2 Verwachte resultaten

- Aantal rijen in de ChangeSet en het aantal rijen in de database: 20569
- Aantal kolommen in de ChangeSet: 155

4. Resultaten

4.1 Resultaat case 1

	Column1	Column2	Column3	Column4	Column5
▶	Naam	Adres	Geboortedatum	Lege kolom	Getallen
	Axel Polet	's Gravenhage	01-12-2012		123123.123123
	Mw. Verh◆gen	Leiden	1 December 2012		123123.123123"
*					

Tabel 1 - Resultaat case 1, output in GUI

Correct:

- ✓ Rij 1, kolom 1 en kolom 2: naam en datum correct
- ✓ Rij 1, kolom 2: **'s Gravenhage** correct
- ✓ Rij 1, kolom 4: **123123.123123** correct geconverteerd (komma naar punt)

Onjuist:

- ❖ Rij 2, kolom 1: speciaal teken **ä** onjuist weergeven, wordt niet herkend
- ❖ Rij 2, kolom 4: dubbele quotes

	id	data
1	1	"Naam","Adres","Geboortedatum","Lege kolom","Getallen",
2	2	"Axel Polet","'s Gravenhage","01-12-2012",,"123123.123123",
3	3	"Mw. Verh◆gen","Leiden","1 December 2012",,"123123.123123" ",

Tabel 2 - Resultaat case 1, output in database

Correct:

- ✓ Zie output GUI
- ✓ Lege velden correct overgenomen (**,,**)

Onjuist:

- ❖ Speciaal teken **ä** onjuist weergeven
- ❖ Komma aan het einde van elke rij

4.2 Resultaat case 2

	Column1	Column2	Column3	Column4	Column5
▶	Naam	Adres	Geboortedatum	Lege kolom	Getallen
	Axel Polet	's Gravenhage	01-12-2012		123123,123123
	Mw. Verh◆gen	Leiden	1 December 2012		123123.123123"
*					

Tabel 3 - Resultaat case 2

Correct:

- ✓ Eerste rij niet geconverteerd naar kolomnamen
- ✓ Rij 1, kolom 4: **123123,123123** niet geconverteerd (van punt naar komma)

4.3 Resultaat case 3

In figuur 2 staat de output in de GUI van de conversieapplicatie na het uitvoeren van de conversie en de query op de database.

```
Delete all rows from table info: OK
Number of columns in DataTable: 155
Number of rows in DataTable: 20569
Number of rows stored in ChangeSet: 20569
```

Figuur 2: Aantallen ingevoerde rijen en kolommen volgens applicatie

Hieronder staat de uitgevoerde query in de SQL Server Management studio om de hoeveelheid rijen in de betreffende database en tabel te controleren.

Query: `SELECT COUNT(data) FROM ConversionDb.dbo.deelnemer_import`
Resultaat: 20569

Correct:

- ✓ Aantal kolommen: 155
- ✓ Aantal rijen: 20569
- ✓ Aantal rijen in db: 20569

Incorrect:

- ❖ Aan het einde van een rij in de database: één komma teveel
- ❖ Umlaut (ü) in sommige rijen niet leesbaar

5. Conclusie

De verwachte resultaten en daadwerkelijke resultaten van testcase 1 en testcase 3 komen niet allemaal overeen. De huidige implementatie kan momenteel dus niet worden geaccepteerd als oplossing voor het uitlezen en converteren van CSV bestanden.

Er moeten dus wijzigingen worden aangebracht aan de implementatie voor het uitlezen van CSV bestanden in de gebruikte versie van de applicatie.

Nadat de nodige wijzigingen zijn gemaakt moeten de in dit document beschreven handelingen van elke testcase opnieuw worden uitgevoerd om te controleren of de conversie van CSV bestanden dan wel aan de verwachtingen voldoet.

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage I

Ontwerprapport Iteratie 1 - Conversiesysteem

Versie 1.2

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 02-03-2012

Documentbeheer

Versie	Datum	Status	Approvedby	Omschrijving
0.1	22-02-2012	Concept		Initiële versie ontwerp Toegevoegde onderdelen: - Database ontwerp (EER) - Class diagram - Verwijzingen naar eisen requirementsdocument
1.0	24-02-2012	Eerste versie		+ Beschrijving DLL + Beschrijving plug-in systeem
1.1	27-02-2012	Tweede versie		+ RRM, RIM + Abstracte klasse ArtenaFileReader
1.2	02-03-2012	Derde versie		+ O/R mapper beschrijvng + Beschrijvingen: - Callbacks - Try-catch statements + Triggers voor referentiele integriteit vanwege multiple cascade paths

Gerelateerde documenten	Versie	Datum
Analyserapport Iteratie 1: Conversiesysteem	1.0	23-02-2012

Inhoudsopgave

1. Inleiding 4
2. Datastructuur 5
2.1 Bestandstructuur 5
2.2 Invoer en uitvoer conversiesysteem 5
2.3 Databaseontwerp 6
2.3.1 (Enhanced) Entity Relationship Diagram 6
2.3.2 Relationeel Representatie Model 8
2.3.3 Relationeel Implementatie Model & Referentiele Integriteit 8
2.3.4 Databasediagram 8
3. O/R mapper 10
3.1 LINQ-to-SQL model 10
3.2 Relaties tussen entiteiten 11
3.3 Uitvoeren 'queries' 11
4. Programmatuur 12
4.1 Dynamic Linked Library 12
4.2 Plug-in systeem 12
4.3 Klassendiagram 13
4.3.1 Beschrijving klassen 14
4.4 Gebruikers interface 14
4.5 Programmatuurkenmerken 15
4.5.1 Callbacks 15
4.5.2 Try-catch statements 16
4.5.3 Karakterset(s) 16
4.6 Conversieproces 16
5. Begrippenlijst 18
Bijlage I.1 - ERD conversiesysteem 19
Bijlage I.2 - Relationeel Implementatie Model 20
Bijlage I.3 – Triggers 22
T1. Delete_Customer 22
T2. Delete_File_Template 22
T3. Delete_Table_Template 23
T4. Delete_Import 23
T5. Delete_Import_File 24

1. Inleiding

Op basis van de vastgelegde functionele en niet-functionele eisen voor het conversiesysteem wordt in dit document een ontwerp opgesteld voor het conversiesysteem. In dit document moeten alle beschreven eisen uit het Analyserapport verwerkt zijn.

In **hoofdstuk 2** wordt de datastructuur beschreven waarin de templates (zoals beschreven in eisen) voor te converteren bestanden in vastgelegd zullen worden. Het databaseontwerp geschiedt door een EER diagram (ERD) te maken waarin het conceptueel databaseontwerp verwerkt zit. Vanuit het EER diagram zal een Relationeel Representatie Model en een Relationeel Implementatie worden opgesteld, waarmee de daadwerkelijke database te implementeren zal zijn.

Hoofdstuk 3 beschrijft de gebruikte O/R (object-relational) mapper waarmee de database-entiteiten worden omgezet naar objecten zodat deze bruikbaar zijn in de applicatie. Met behulp van de O/R mapper is het dus niet nodig om voor bijvoorbeeld de templates klassen te ontwerpen.

Hoofdstuk 4 beschrijft tot slot de programmatuur waarin het conversieproces geïmplementeerd zal zijn Voor deze beschrijving worden twee soorten UML modellen gebruikt:

- **Klassendiagram**
  Beschrijving van de verschillende klassen die gebruikt zullen worden bij de implementatie van het conversiesysteem en de relaties tussen deze klassen. Dit diagram toont de structuur van de programmatuur.

- **Sequence diagram**
  Beschrijving van de volgorde van acties betrokken bij de afhandeling van één of meerdere interacties die vanuit de gebruiker zijn gestart. Het diagram beschrijft dus het gedrag van de programmatuur.

NB1 Functionele eis F10 wordt niet behandeld in dit document, maar zal worden behandeld in het ontwerprapport van de derde iteratie waarin de implementatie van het log systeem is vastgelegd.

NB2 Functionele eis NF1 wordt niet behandeld in dit document, omdat de uitvoer van de conversie wordt geregeld door een correcte implementatie in bestandslezers. In deze implementatie zullen uitgelezen velden worden weggeschreven in CSV formaat.

2. Datastructuur

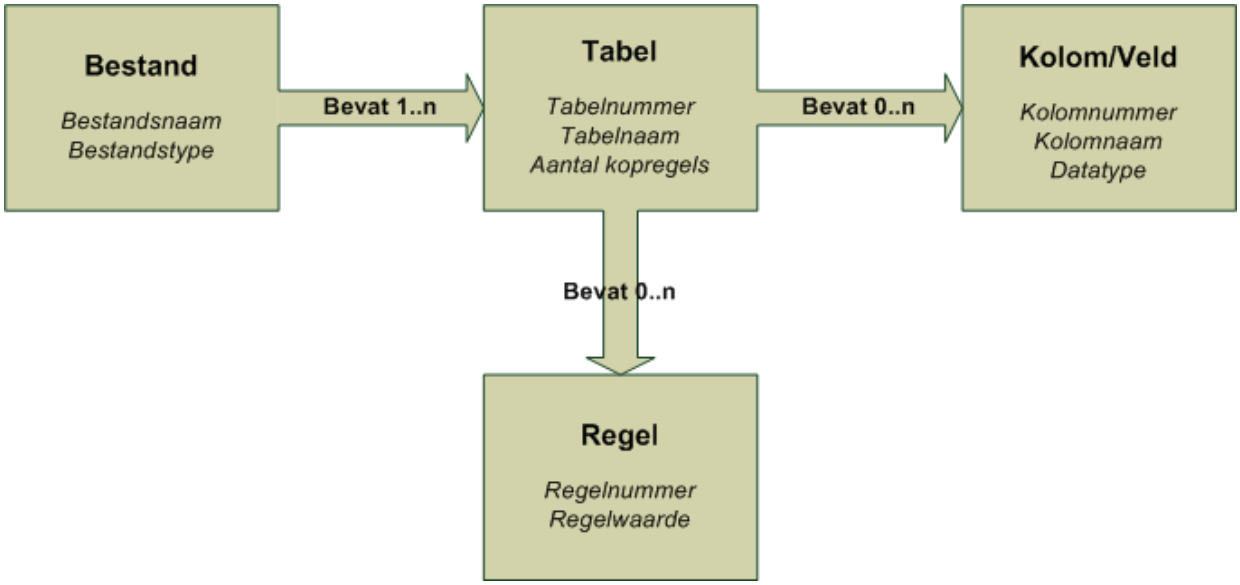
Dit hoofdstuk beschrijft de database waarin belangrijke gegevens voor het conversiesysteem worden vastgelegd. De twee belangrijkste functionaliteiten van de database:

1. Het vastleggen van de templates voor de te converteren bestanden
2. Mogelijkheid tot wegschrijven van geconverteerde data en deze koppelen aan de template(s)

2.1 Bestandstructuur

*Functionele eis **F4***

De bestandstructuur beschrijft aan welke voorwaarden een aangeleverd klantbestand moet voldoen wil het geconverteerd kunnen worden door het conversiesystem. Figuur 1 beschrijft deze bestandsstructuur.



Figuur 1: Bestandstructuur

Elk aangeleverd bestand bevat één of meerdere tabellen welke weer nul of meerdere kolommen/ velden bevat. Daarnaast bevat een tabel ook nul of meerdere regels waarin de daadwerkelijke data staat (de bestandsregels).

Bijvoorbeeld:

- Een Excel bestand bevat twee sheets (tabellen) met elk een verschillend aantal kolommen/velden. Een kolom/veld hoort bij één tabel en een tabel hoort bij één bestand.
- Een CSV bestand bevat maar één tabel omdat de mogelijkheid tot meerdere sheets niet aanwezig is in CSV. De tabel bevat nul of meerdere kolommen/velden en regels

2.2 Invoer en uitvoer conversiesysteem

*Functionele eisen **F1 – F7***

Zoals beschreven in de requirements en de bestandsstructuur bestaat de invoer van het conversiesysteem uit één of meerdere klantbestanden en één of meerdere templates. Ter verduidelijking hieronder een beschrijving in de vorm van logische expressies.

Invoer conversie: ((klantbestand)+) + ((template)+)

Bestand: (tabel)+
Tabel: (rij)*
Rij (Excel)**:** (veld)*
Rij (CSV)**:** (veld + delimiter)* + veld
Delimeter: “,”
Veld: (datum | getal | double | valuta | integer)

Template: aantal tabellen in bestand + bestandstype
Aantal tabellen: integer
Bestandstype: (xls | xlsx | csv)

De uitvoer van een conversie bestaat uit database (= een dataset van gegevens die wordt weggeschreven naar een database) met daarin een verwijzing naar het klantbestand, de identificatie van de klant en de bestandsregels uit dat bestand.

Uitvoer conversie: naam klantbestand + identificatie klant + (rij)*

Rij: tabelnummer + rijnummer + ((veld + “,”)* + veld)
Veld(datum): ‘dd-mm-yyyy hh:mm:ss’
Veld(integer): ‘xxxx’
Veld(double): ‘xx.xx’

2.3 Databaseontwerp

*Functionele eisen **F1, F4-F7, F11***

Een belangrijke voorwaarde aan een databaseontwerp bij Artena is het feit dat elke tabel één unieke primaire sleutel is die alleen gebruikt voor identificatie en verder geen informatie toevoegt aan datgene wat hij identificeert (*een surrogaatsleutel*).

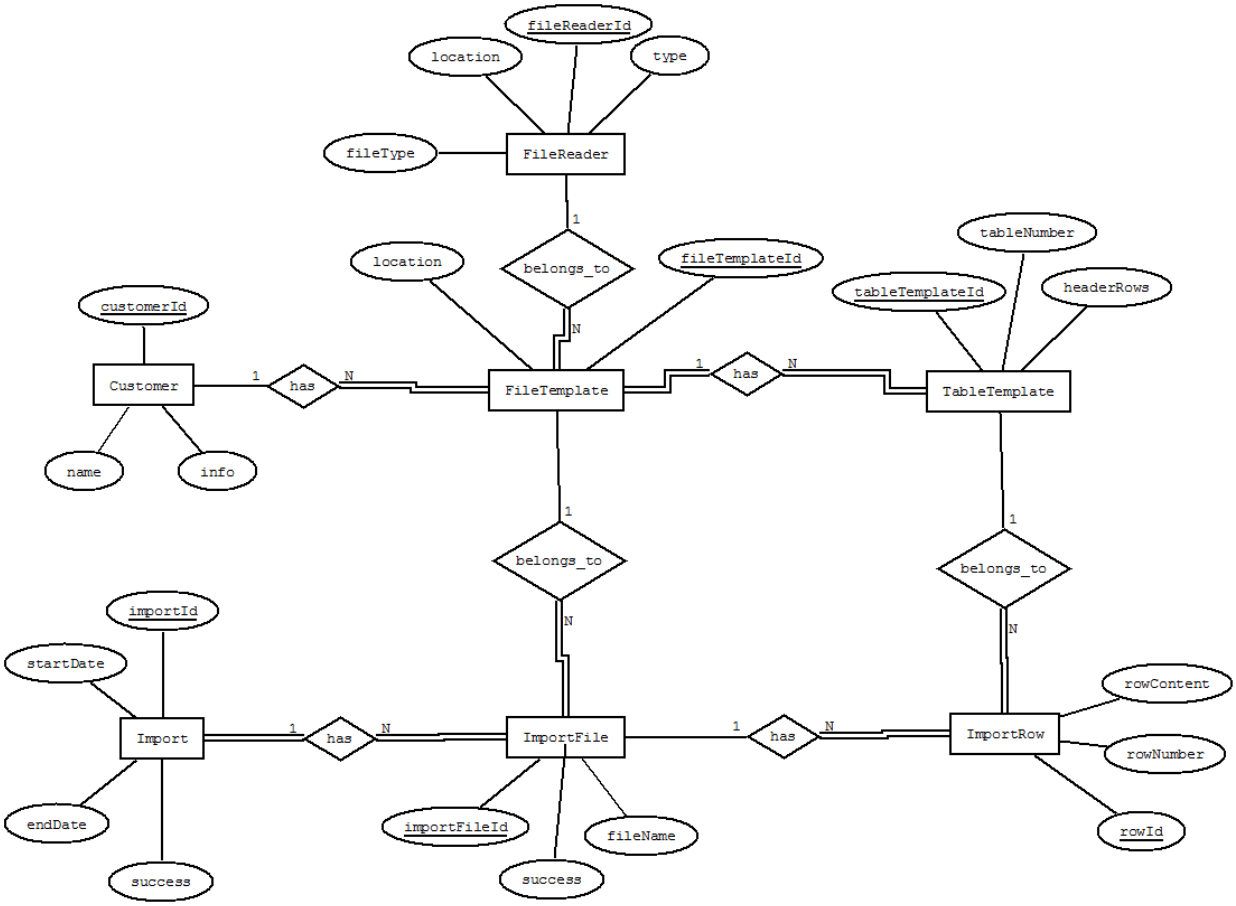
Het databaseontwerp bestaat uit vier ontwerpmodellen:

- (E)ER diagram (conceptueel databaseontwerp)
- Relationeel Representatie Model (logisch databaseontwerp)
- Relationeel Implementatie Model (logisch databaseontwerp – implementatie)
- Microsoft Visio 2010 databasediagram (logisch databaseontwerp – grafische weergave)

2.3.1 (Enhanced) Entity Relationship Diagram

In figuur 2 wordt het (E)ER diagram voor de database van het conversiesysteem getoond. In dit diagram zijn attributen en primaire sleutel per entiteittype aangegeven en worden de relaties tussen entiteittypen aangegeven.

Ook worden *kardinaliteiten* van de relaties aangegeven(zie de 1-N notaties) en wordt de verplichte (NULL/NOT NULL) invulling aangegeven (door middel van de dubbele en enkele relatielijnen/partiële participatie/totale participatie).



Figuur 2: ERD conversiesysteem

Voor de invoer voor een conversie wordt een template vastgelegd in de vorm van de entiteiten *FileTemplate* en *TableTemplate*. Deze template beschikt dus over het aantal tabellen dat in een aangeleverd klantbestand moet zitten. Ook beschikt deze tabel over een definitie van het bestandstype door de relatie met de entiteit *FileReader* (deze tabel bezit ook nog andere kenmerken benodigd voor het plug-in systeem, zie §4.2)

Voor de uitvoer van de een conversie zijn drie entiteiten beschreven: *Import, ImportFile en ImportRow.* In deze 'import' entiteiten is de uitvoer van de een conversie vastgelegd.

- Een nieuwe import is gekoppeld aan een klant
- Een import kan bestaan uit één of meerdere aangeleverde klantenbestanden. Elk bestand wordt vastgelegd in een *ImportFile* entiteit. Elk klantbestand is gekoppeld aan een template via de relatie *FileTemplate*
- Een rij uit een klantbestand (een bestandsregel) is vastgelegd in de vorm *ImportRow* entiteit en gekoppeld aan het klantbestand (*ImportFile*) en de bijbehorende tabel via de relatie met *TableTemplate* (van een tabel)

In **bijlage I.1** is het EER diagram van het conversiesysteem in het groot bijgevoegd.

2.3.2 Relationeel Representatie Model

Dit hoofdstuk beschrijft de tabellen van de volledige database die kan worden afgeleid uit het ERD van §2.3.1. Het model beschrijft de kolommen per tabel, de primaire sleutels en de vreemde sleutels die gebruikt zullen worden op de relaties tussen tabellen:

Customer(CustomerId, Name, Info)

FileReader(FileReaderId, FileType, ReaderLocation, ReaderFullType)

FileTemplate(FileTemplateId, FileLocation, *CustomerId*, *FileReaderId*)
CustomerId is foreign key, references CustomerId in Customer, NOT NULL
FileReaderId is foreign key, references FileReaderId in FileReader, NOT NULL

TableTemplate(TableTemplateId, TableNumber, IgnoreRows, *FileTemplateId)*
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NOT NULL

Import(ImportId, StartDate, EndDate, Success)

ImportFile(ImpFileId, FileName,Success, *ImportId, TemplatelFileId*)
ImportId is foreign key, references ImportId in Import, NOT NULL
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NOT NULL

ImportRow(ImpRowId, RowNumber, RowContent, *ImpFileId, TableTemplateId*)
ImpFileId is foreign key, references ImpFileId in ImportFile, NOT NULL
TableTemplateId is foreign key, references TableTemplateId in TableTemplate, NOT NULL

2.3.3 Relationeel Implementatie Model & Referentiele Integriteit

In **bijlage I.2** staat het **R**elationeel **I**mplementatie **M**odel voor SQL Server 2008 dat is gebaseerd op het RRM. In dat model zijn de datatypes van de kolommen vastgelegd.

Naast datatypen moet ook de referentiele integriteit worden vastgelegd voor elke vreemde sleutel relatie in de database. De volgende regels gelden:

1. Elke primaire sleutel is een surrogaat sleutel die automatisch wordt gegenereerd en opgehoogd door het SQL Server DBMS
2. ON UPDATE staat altijd op 'NO ACTION' (omdat de sleutels niet gewijzigd zullen worden, §2.3)
3. ON DELETE moet waar nodig op 'CASCADE' staan

In het RIM staan geen ON DELETE definities omdat deze ON DELETE definities niet geïmplementeerd konden worden. De reden hiervoor is dat SQL Server automatisch checkt op *'multiple cascade paths*[1]*'* die vanuit één DELETE query kunnen ontstaan en dit niet toestaat.

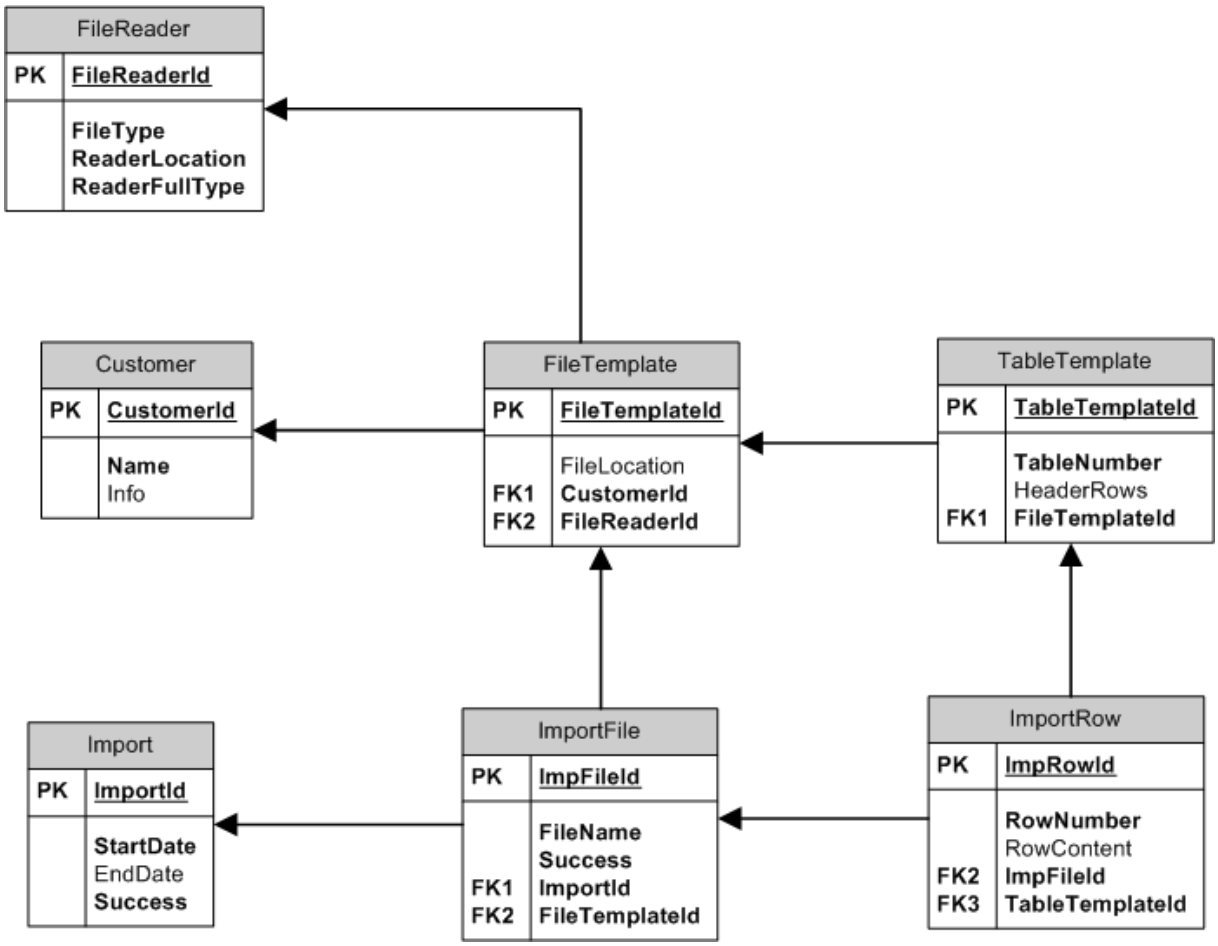
Om toch referentiële te verplichten moeten voor een aantal tabellen TRIGGERS geïmplementeerd worden waarin de DELETE query op een juiste wijze wordt uitgevoerd. Zie **bijlage I.3**.

2.3.4 Databasediagram

Aan de hand van het RRM en RIM is het mogelijk om een database te implementeren in Microsoft SQL Server 2008. Om een duidelijker beeld te geven van de database voor het conversiesysteem is een

[1] http://www.christian-etter.de/?tag=multiple-cascade-paths

databasediagram met behulp van Microsoft Visio opgesteld voor een grafische weergave van de in het RRM beschreven tabellen. In dit diagram staat de gehele database voor het conversiesysteem inclusief alle tabellen, kolommen, primaire sleutels en vreemde sleutels beschreven. Zie figuur 3.



Figuur 3: Databasediagram Conversiesysteem (visio)

3. O/R mapper

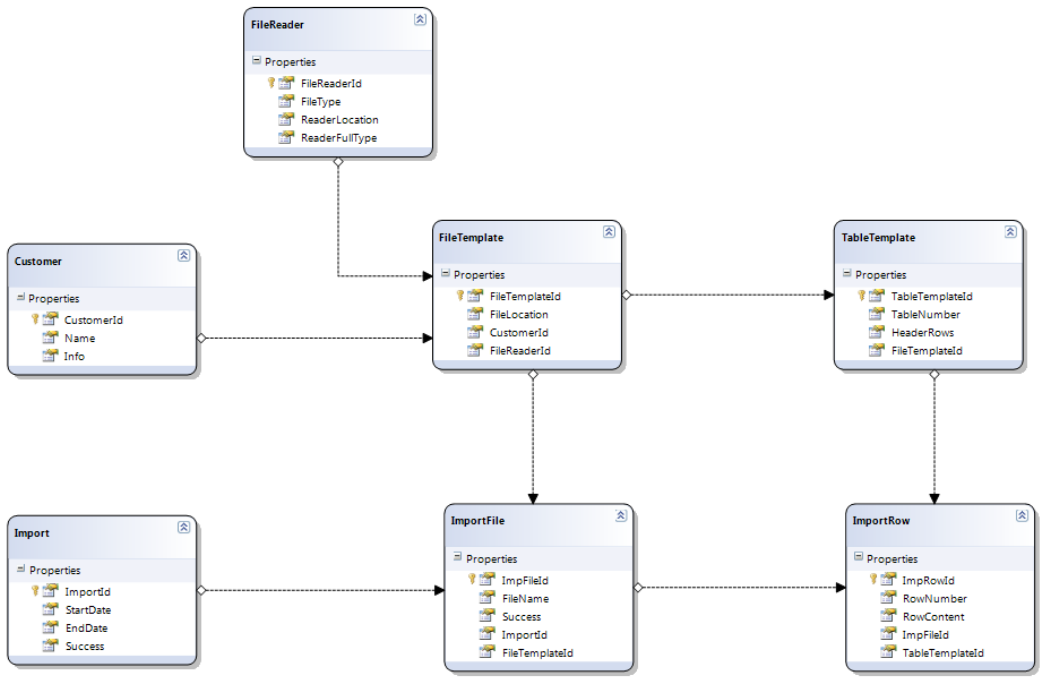
Zoals aangegeven in de inleiding is er gebruik gemaakt van een O/R mapper om de entiteiten uit de database van het conversiesysteem om te zetten naar objecten in C#. Hierdoor is het niet nodig om aparte objecten te implementeren in de programmatuur van het conversiesysteem.

Het conversiesysteem zal hiervoor **LINQ-to-SQL** gebruiken. Een **LINQ-to SQL Class** is een O/R mapping (object-relational) waarin de tabellen uit een SQL database worden omgezet naar entiteiten (entity classes). Hierdoor is het mogelijk om vanuit een willekeurige methode of class te communiceren is met data in de database via de instanties van die entity classes. De data in een tabel is daardoor direct aanspreekbaar en manipuleerbaar vanuit de code. Samengevat: het is met LINQ to SQL mogelijk om met objecten te communiceren i.p.v. het continue query-en van een database en de gegevens om te zetten naar eigen implementaties.

Het conversiesysteem zal voor dus voor de communicatie met de gebruikte SQL Server database en zijn tabellen gebruik maken van LINQ. LINQ staat voor Language Integrated Query en is een programmeermodel om query's op verschillende databronnen uit te voeren. Met behulp van LINQ is het mogelijk om een volledige query te vervangen door het aanroepen van verschillende LINQ methoden. Het gebruiken van LINQ is volledige gecontroleerd door de .NET omgeving waardoor bijvoorbeeld *type-casting*(een integer wordt bijvoorbeeld altijd direct als integer uitgelezen) van resultaten uit een query overbodig is. LINQ is onder andere bruikbaar op databases (SQL Server, MS Access) maar kan ook worden toegepast op XML bestanden of op C# objecten.[2]

3.1 LINQ-to-SQL model

Figuur 4 toont het LINQ-to-SQL na omzetting van de database naar objecten.



Figuur 4: LINQ-to-SQL representative database van conversiesysteem

[2] http://weblogs.asp.net/scottgu/archive/2007/05/19/using-linq-to-sql-part-1.aspx

3.2 Relaties tussen entiteiten

Zoals beschreven in de datastructuur van het conversiesysteem kan een import uit meerdere bestanden staan. Deze relatie wordt in de database gerealiseerd door een vreemde sleutel in de tabel **ImportFile** naar de tabel **Import**. Wanneer deze twee tabellen worden omgezet naar entiteiten bij LINQ to SQL wordt de relatie gerealiseerd door de entiteit **Import** een lijst (array) te geven met **ImportFiles**.

3.3 Uitvoeren 'queries'

Nadat een LINQ to SQL Class is gedefinieerd kan overal de applicatie een nieuwe instantie van deze class worden aangemaakt, waarna vervolgens gecommuniceerd kan worden. Een LINQ to SQL class wordt ook wel de *DataContext* genoemd.

Het uitvoeren van een INSERT query op bijvoorbeeld de tabel **Import** ziet er als volgt uit:

1. Maak één of meerdere instanties van de entiteit **Import** aan
2. Leg in elk van deze instanties de betreffende data (zoals start datum) vast
3. Voeg de instanties toe aan de *ChangeSet* van een DataContext instantie
4. Verzend alle changes in de ChangeSet van de DataContext instantie

4. Programmatuur

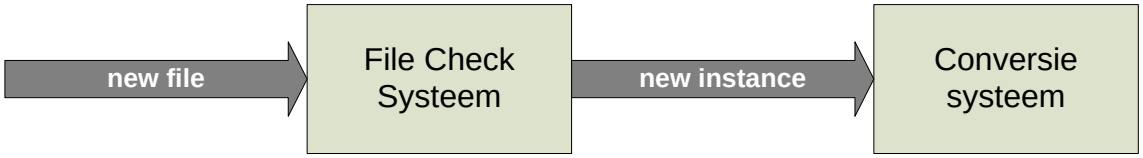
Dit hoofdstuk beschrijft de wijze waarop de software van het conversiesysteem is opgebouwd. Het onderzoek heeft bepaald dat er een implementatie in C# gebruikt gaat worden het uitlezen van de klantbestanden. Dit hoofdstuk beschrijft twee belangrijke onderdelen van het conversiesysteem:

1. Het plug-in systeem voor de bestandslezers
2. Het conversieproces

4.1 Dynamic Linked Library

*Functionele eisen **F9, F11, F12***

Het conversiesysteem is door een willekeurige applicatie te implementeren omdat het geïmplementeerd wordt als een DLL. Een gebruikersinterface kan deze DLL dan inladen en functionaliteiten van het systeem gebruiken. Figuur 5 toont een File Check systeem dat het conversiesysteem gebruikt.



Figuur 5: starten van conversiesysteem vanuit File Check Systeem

Indien twee conversie voor twee klanten tegelijkertijd uitgevoerd moeten worden, dan zullen twee instanties van het conversiesysteem aangemaakt moeten worden vanuit het *File Check Systeem.*

4.2 Plug-in systeem

*Eisen: **F2, NF3***

Voor de afhandeling van verschillende soorten bestandsregels in het conversiesysteem zal een plug-in[3] systeem geïmplementeerd worden. Dit houdt in dat het systeem voor elke bestandslezer een aparte plug-in zal inladen die een bestand zal inladen. Er zal dus een plug-in voor het uitlezen van Excel bestanden worden gemaakt en een plug-in voor CSV bestanden.

Om dit plug-in systeem te implementeren zal gebruik gemaakt worden van Reflection[4,5]. Reflection 'is het proces waarbij een applicatie zichzelf kan observeren en zijn eigen structuur en gedrag runtime (dynamisch) kan veranderen'.

Dit houdt dit dat het mogelijk is om op basis van bepaalde informatie een bepaald type object (de plug-in) in te laden die de gewenste functionaliteiten (die dan nodig zijn) uit te voeren. Deze plug-in zal geïmplementeerd moeten zijn als een DLL.

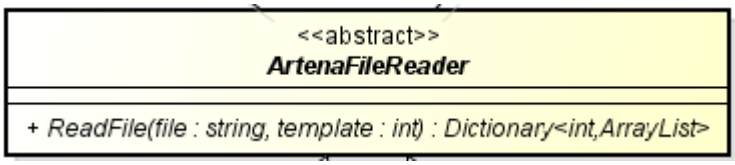
De gegevens die het systeem nodig heeft om een plug-in in te laden zijn:

- Locatie van het DLL bestand (bijv. C:\Software\ExcelFileReader.dll)
- *Volledige type* van het in te laden object (bijv. Artena.FileReaders.ExcelFileReader)

[3] http://en.wikipedia.org/wiki/Plug-in_(computing)
[4] http://en.wikipedia.org/wiki/Reflection_(computer_programming)
[5] http://www.codeguru.com/csharp/csharp/cs_misc/reflection/article.php/c4257

Deze twee gegevens zijn vastgelegd in de database van het conversiesysteem in de tabel *FileReader*. Door dit plug-in systeem wordt een grote mate van flexibiliteit gecreëerd omdat voor een extra bestandsformaat dat ondersteund moet worden enkel een nieuwe plug-in geschreven moet worden die geen impact zal hebben op het conversieproces.

Om te garanderen dat het conversiesysteem elke plug-in op dezelfde wijze behandeld (en het aantal plug-ins dus uitbreidbaar is) zal de plug-in (of liever: de concrete klasse in de plug-in) overerven van een abstracte klasse. Zie figuur 6.



Figuur 6: Abstracte klasse voor bestandslezer plug-in

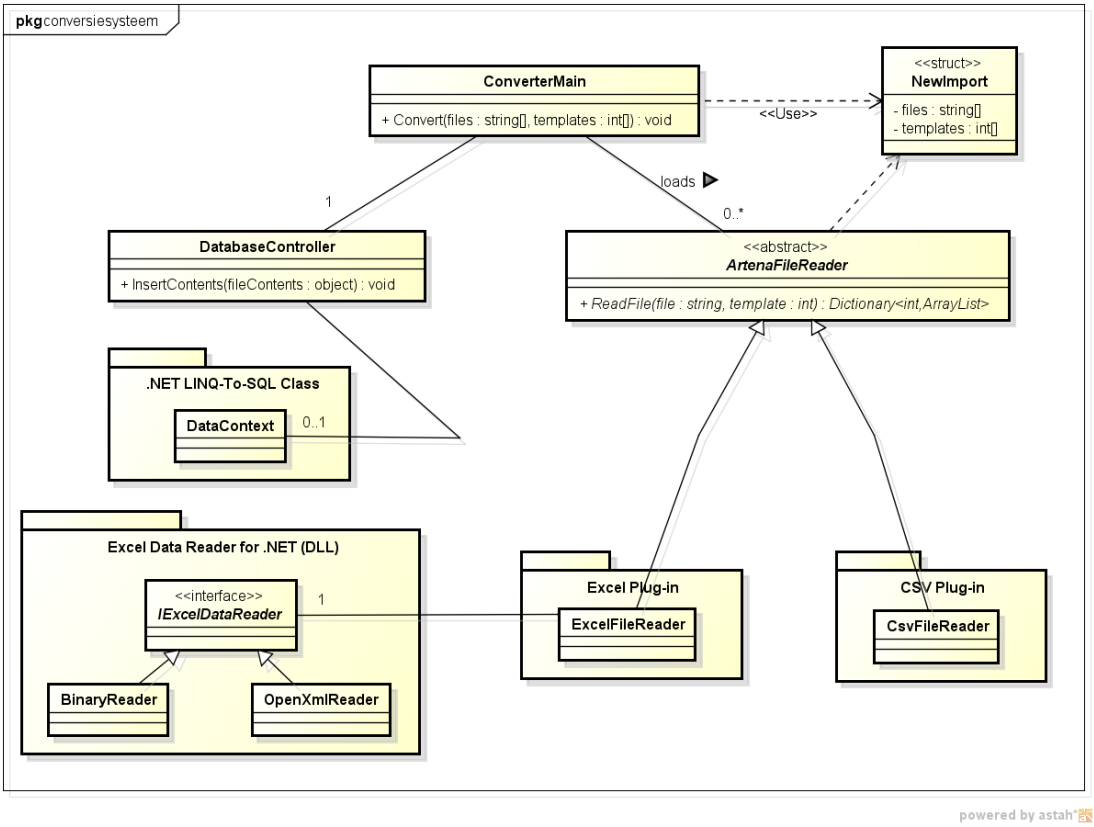
De methode **ReadFile()** moet per definitie geïmplementeerd worden door de overervende klassen en zorgt ervoor dat het conversiesysteem elke plug-in op dezelfde manier kan behandelen.

4.3 Klassendiagram

Het 'core' gedeelte bestaat globaal uit drie verschillende onderdelen:

- Een onderdeel dat toegang verleent aan een *client* en het conversieproces overziet
- Het plug-in systeem waarmee bestandslezers worden ingeladen
- Een onderdeel dat communiceert met de database en de verantwoordelijkheid heeft om het conversiesysteem te voorzien van de gedefinieerde in –en uitvoer.

Het klassendiagram van figuur 7 beschrijft deze onderdelen:



Figuur 7: domeinmodel conversiesysteem

4.3.1 Beschrijving klassen

NewImport is het object waarin de specifieke gegevens voor een nieuwe import worden bijgehouden, dit zijn identificaties van de templates (zoals gedefinieerd in de database) en de locaties van de, te 'converteren', klantbestanden.

ConverterMain heeft controle over het conversieproces van begin tot eind. Deze klasse zal de gewenste bestandslezers(s) kiezen, het conversieproces starten en de bestandsregels verkregen uit de bestandslezers moeten wegschrijven.

DatabaseController is verantwoordelijk voor de connectie met de database en de uitvoer van alle benodigde query's voor het conversiesysteem. Voor de connectie met de database maakt deze klasse gebruik van een DataContext klasse.

DataContext is een LINQ-To-SQL klasse waarmee met de database gecommuniceerd kan worden. Zie §3.3

ArtenaFileReader is de *abstracte klasse* waarin de een aantal noodzakelijke functionaliteiten van een datareader zijn vastgelegd, maar niet geïmplementeerd. Deze functionaliteiten hebben betrekking op het uitlezen en eventueel manipuleren van data uit een aangeleverd bestand van een bepaald bestandsformaat.

ExcelFileReader is een concrete klasse die overerft van ArtenaFileReader, deze klasse is verantwoordelijk voor het uitlezen van Excel 97-2003 en Excel2007-2010 klantbestanden en de data om te zetten naar het juiste formaat (CSV, zie requirements). Deze klasse is ook verantwoordelijk voor het manipuleren van Excel97-2003 bestanden, zodat de inhoud van deze data hetzelfde formaat (functionele eis **F8)** heeft als de inhoud van de Excel2007-2010 data. Deze klasse maakt gebruik van de Excel Data Reader for .NET bibliotheek.

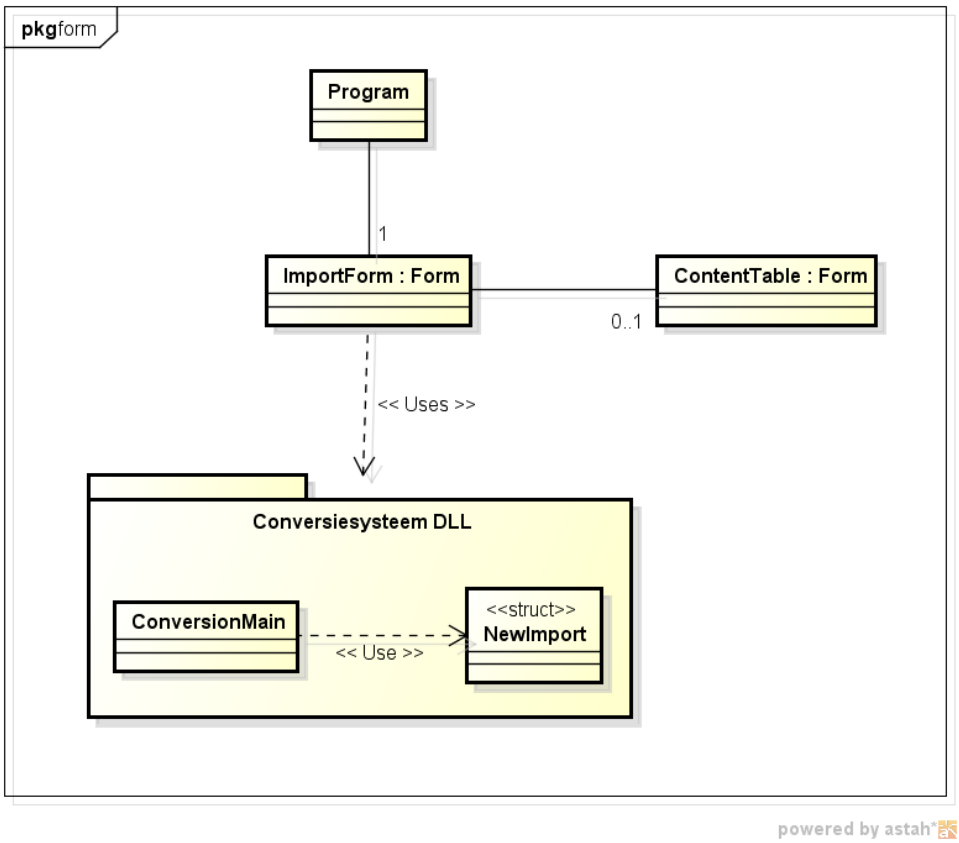
CsvFileReader is de eigen implementatie voor het, regel voor regel, uitlezen van een Comma Separated Values klantbestand. Eventuele manipulaties aan de data kunnen hier ook worden uitgevoerd.

Excel Data Reader for .NET is de gebruikte (en gekozen) bibliotheek om de data van een Excel bestand uit te lezen en op te slaan in een *DataSet* object.

4.4 Gebruikers interface

Voor de ontwikkeling van het conversiesysteem zal een gebruikersinterface gemaakt worden waarmee de ontwikkelaar het conversiesysteem kan inladen en gebruiken. De wijze waarop een dergelijke interface geïmplementeerd moet worden is niet opgenomen in de requirements (zie analyserapport) en zal dus volledig ingericht zijn om het conversiesysteem eventueel te testen of te debuggen.

Figuur 8 toont de klassen die gebruikt zullen worden bij de implementatie van de gebruikers interface. Zoals aangegeven maakt de interface gebruik van de core van het conversiesysteem door de DLL in te laden. Via de ConversionMain klasse kan via de gebruikersinterface een conversie opgestart worden.



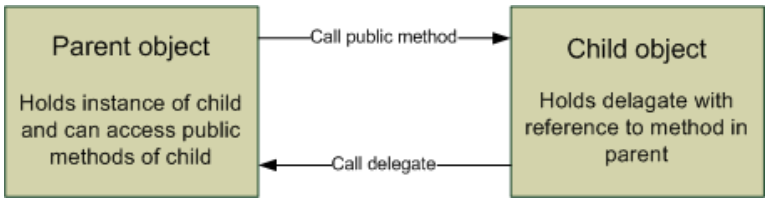
Figuur 8: domeinmodel gebruikersinterface

4.5 Programmatuurkenmerken

In de applicatie zal gebruik gemaakt worden van een aantal oplossingen om bepaalde

4.5.1 Callbacks

Tussen de verschillende objecten in het systeem wordt data uitgewisseld. Sommige objecten (children) worden aangemaakt door andere objecten(parents). De parents kunnen altijd direct data doorgeven aan de hun kinderen. Echter wanneer kinderen data aan hun parents willen geven dan zal dit gebeuren door middel van een callback, zie figuur 9.



Figuur 9: Callback principe

Met behulp van een callback hoeft het kind namelijk niet te weten naar wie hij zijn data doorgeeft, dus het kind hoeft geen referentie naar zijn ouder te hebben. In C# is deze constructie te implementeren met behulp van een *delegate*[6]. Deze delegate wordt bij het instantiëren van het kind aangemaakt. De delegate bevat een referentie naar een methode van een parent.

[6] http://msdn.microsoft.com/en-us/library/900fyy8e(v=vs.71).aspx

4.5.2 Try-catch statements

In C# is het mogelijk om try-catch[7] statements te implementeren. Met behulp van deze statements is het mogelijk om een fout af te vangen terwijl een methode wordt doorlopen (wanneer zich een fout voordoet) zonder dat de applicatie crasht.

```
Try
{
        Code to be executed….
}
Catch(Exception ex)
{
        In case of exception, catch it
        and it's possible to access
        specific information about the
        exception through 'ex'
}
```

Figuur 10: Try-catch implementatie

De kenmerken van de afgevangen fout kunnen vervolgens worden opgeslagen in een database of worden weergeven in de gebruikersinterface Voor bepaalde processen worden speciale try-catch statements gemaakt die bij het afvangen van een exceptie ervoor zorgen dat het conversieproces afbreekt.

4.5.3 Karakterset(s)

Niet functionele eis: ***NF2***

Het conversiesysteem moet de ANSI standaard volledig ondersteunen. Bij het uitlezen van de te ondersteunen bestandsformaten moet het systeem alle tekens in deze standaard[8] ondersteunen.

- Excel bestanden: de Excel Data Reader for .NET ondersteund deze standaard
- CSV bestanden: standaard worden bestanden uitgelezen op basis van de UTF-8 tekenset[9]

Voor de CsvFileReader moet de tekenset 1252 (Windows ANSI) ingesteld worden.

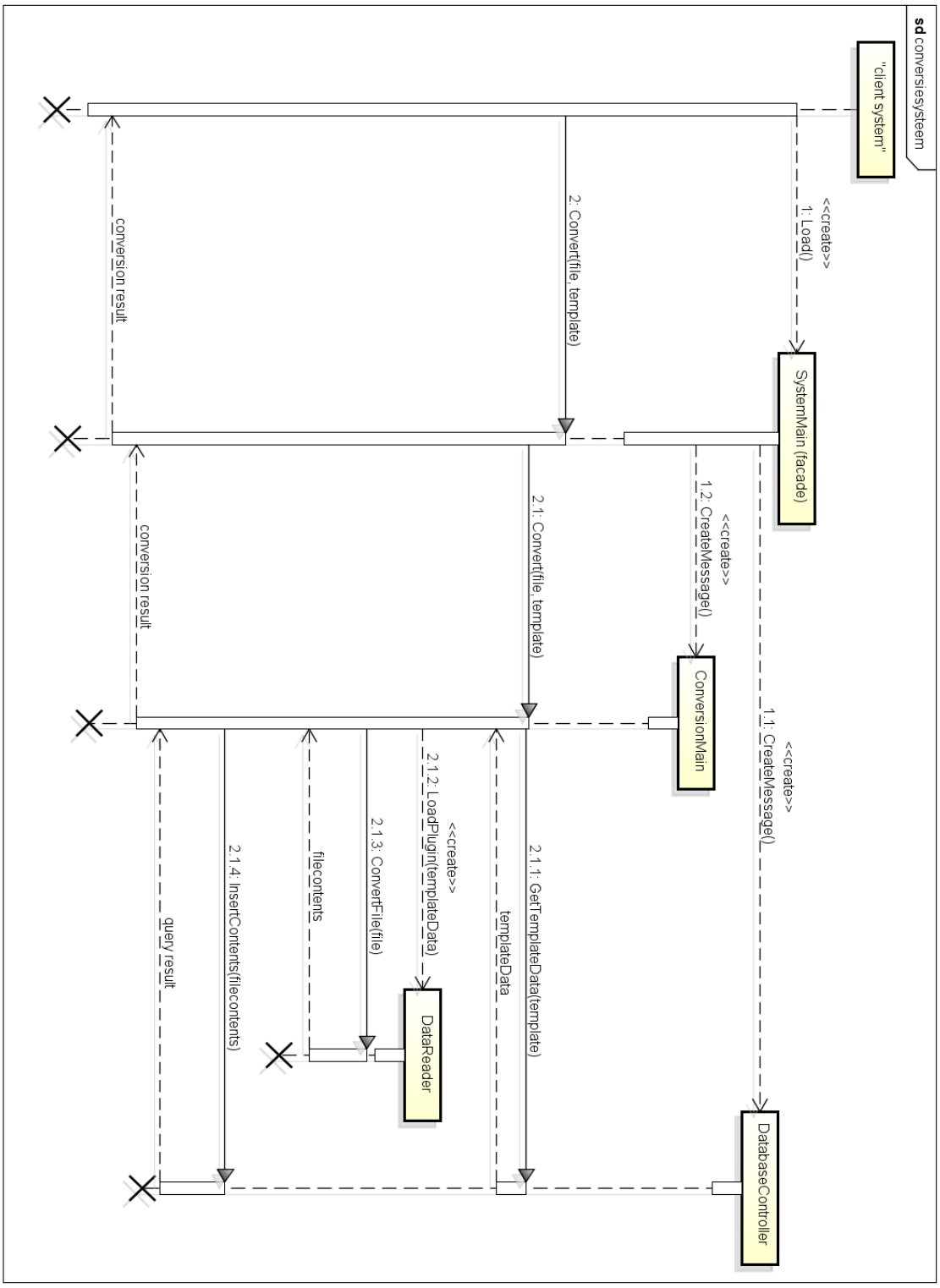
4.6 Conversieproces

Het conversieproces bestaat globaal gezien uit drie stappen:

1. Verkrijgen van verwijzingen naar het/de klantbestand(en) en de template(s)
2. Inladen van de juiste plug-ins om de bestanden uit te lezen
3. Bestanden uitlezen en geconverteerde data wegschrijven naar de database

Deze stappen zijn in detail uitgewerkt in een UML sequencediagram, zie figuur 11.

[7] http://msdn.microsoft.com/en-us/library/0yd65esw.aspx
[8] http://ascii-table.com/ansi-codes.php
[9] http://en.wikipedia.org/wiki/UTF-8



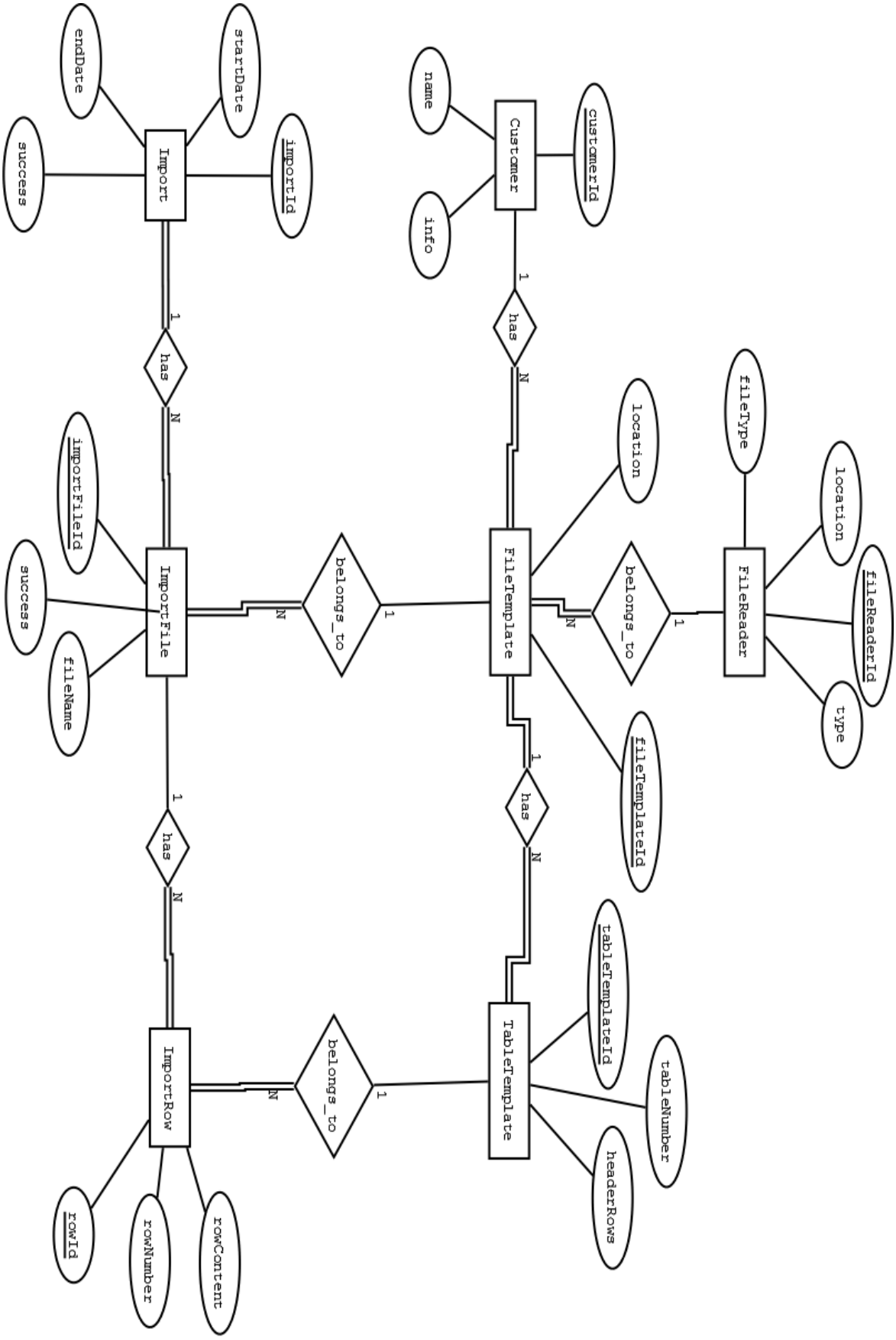
Figuur 11: Conversieproces

5. Begrippenlijst

- Abstract class
  - *Een klasse waarin minimaal één methode abstract is. Dit houdt in dat is aangeven hoe de methode eruit ziet, maar niet wat hij doet (de implementatie). Elke klasse die overerft van de abstracte klasse moet de abstracte methode altijd implementeren*
- ChangeSet
  - *Een onderdeel van de DataContext. In de ChangeSet worden alle nieuwe wijzigingen, zoals Insert, Updates en Deletes toegevoegd. Vanuit de DataContext kan een gehele ChangeSet in één keer worden doorgevoerd op de database*
- DataContext
  - *De representatie van een LINQ-to-SQL class. Van dit object kunnen meerdere instanties aangemaakt worden zodat de applicatie met de betrokken database kan communiceren.*
- File Check Systeem
  - *Een systeem dat één of meerdere locaties continu scant of pollt en bij upload van een nieuw bestand op één van deze locaties een actie uitvoert*
- Instantie
  - *Een nieuwe kopie van een object dat gebruikt kan worden door een systeem, object of methode*
- Type-casting
  - *Het opslaan van een kopie van één type variabele (zoals int) in een andere type variabele (zoals double). Er zijn verschillende mogelijkheden tot casten[10] en casten kan ook gevaren opleveren wanneer onjuiste of incompatibele types worden gebruikt.*

[10] http://msdn.microsoft.com/en-us/library/ms173105.aspx

Bijlage I.1 - ERD conversiesysteem



Bijlage I.2 - Relationeel Implementatie Model

```
CREATE TABLE [Import](
	[ImportId] [int] IDENTITY(1,1) NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[Success] [int] NOT NULL,
 CONSTRAINT [PK_import] PRIMARY KEY CLUSTERED
(
	[ImportId] ASC
))
```

```
CREATE TABLE [FileReader](
	[FileReaderId] [int] IDENTITY(1,1) NOT NULL,
	[FileType] [varchar](50) NULL,
	[ReaderLocation] [varchar](500) NULL,
	[ReaderFullType] [varchar](200) NULL,
 CONSTRAINT [PK_DataReader] PRIMARY KEY CLUSTERED
(
	[FileReaderId] ASC
))
```

```
CREATE TABLE [TableTemplate](
	[TableTemplateId] [int] IDENTITY(1,1) NOT NULL,
	[TableNumber] [int] NOT NULL,
	[HeaderRows] [int] NULL,
	[FileTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Template_Data_Definition] PRIMARY KEY CLUSTERED
(
	[TableTemplateId] ASC
))
```

```
CREATE TABLE [FileTemplate](
	[FileTemplateId] [int] IDENTITY(1,1) NOT NULL,
	[FileLocation] [varchar](500) NULL,
	[CustomerId] [int] NOT NULL,
	[FileReaderId] [int] NULL,
 CONSTRAINT [PK_TemplateFile] PRIMARY KEY CLUSTERED
(
	[FileTemplateId] ASC
))
```

```
CREATE TABLE [ImportFile](
	[ImpFileId] [int] IDENTITY(1,1) NOT NULL,
	[FileName] [varchar](500) NOT NULL,
	[Success] [int] NOT NULL,
	[ImportId] [int] NOT NULL,
	[FileTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Import_File] PRIMARY KEY CLUSTERED
(
	[ImpFileId] ASC
))
```

```
CREATE TABLE [ImportRow](
	[ImpRowId] [int] IDENTITY(1,1) NOT NULL,
	[RowNumber] [int] NOT NULL,
	[RowContent] [varchar](max) NULL,
	[ImpFileId] [int] NOT NULL,
	[TableTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Import_Content] PRIMARY KEY CLUSTERED
(
	[ImpRowId] ASC
))
```

```
CREATE TABLE [Customer](
	[CustomerId] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](200) NOT NULL,
	[Info] [varchar](1000) NULL,
 CONSTRAINT [PK_Client_Main_Template] PRIMARY KEY CLUSTERED
(
	[CustomerId] ASC
))
```

```
ALTER TABLE [FileTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateFile_FileReader] FOREIGN KEY([FileReaderId])
REFERENCES [FileReader] ([FileReaderId])
ALTER TABLE [FileTemplate] CHECK CONSTRAINT [FK_TemplateFile_FileReader]

ALTER TABLE [FileTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateFile_TemplateCustomer] FOREIGN KEY([CustomerId])
REFERENCES [Customer] ([CustomerId])
ALTER TABLE [FileTemplate] CHECK CONSTRAINT
[FK_TemplateFile_TemplateCustomer]

ALTER TABLE [ImportFile]  WITH CHECK ADD  CONSTRAINT [FK_ImportFile_Import]
FOREIGN KEY([ImportId])
REFERENCES [Import] ([ImportId])
ALTER TABLE [ImportFile] CHECK CONSTRAINT [FK_ImportFile_Import]

ALTER TABLE [ImportFile]  WITH CHECK ADD  CONSTRAINT
[FK_ImportFile_TemplateFile] FOREIGN KEY([FileTemplateId])
REFERENCES [FileTemplate] ([FileTemplateId])
ALTER TABLE [ImportFile] CHECK CONSTRAINT [FK_ImportFile_TemplateFile]

ALTER TABLE [ImportRow]  WITH CHECK ADD  CONSTRAINT
[FK_ImportRow_ImportFile] FOREIGN KEY([ImpFileId])
REFERENCES [ImportFile] ([ImpFileId])
ALTER TABLE [ImportRow] CHECK CONSTRAINT [FK_ImportRow_ImportFile]

ALTER TABLE [ImportRow]  WITH CHECK ADD  CONSTRAINT
[FK_ImportRow_TemplateTable] FOREIGN KEY([TableTemplateId])
REFERENCES [TableTemplate] ([TableTemplateId])
ALTER TABLE [ImportRow] CHECK CONSTRAINT [FK_ImportRow_TemplateTable]

ALTER TABLE [TableTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateTable_TemplateFile] FOREIGN KEY([FileTemplateId])
REFERENCES [FileTemplate] ([FileTemplateId])
ALTER TABLE [TableTemplate] CHECK CONSTRAINT [FK_TemplateTable_TemplateFile]
```

Bijlage I.3 – Triggers

T1. Delete_Customer

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between Customer and FileTemplate
-- =============================================
ALTER TRIGGER [dbo].[Delete_Customer]
   ON  [ConversionImport].[dbo].[Customer]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      BEGIN
            -- Delete fileTemplate entries linked to the customer(enforces
CASCADE)
            DELETE FROM [ConversionImport].[dbo].[FileTemplate]
            FROM [ConversionImport].[dbo].[FileTemplate] AS ft
            INNER JOIN deleted d ON ft.CustomerId = d.CustomerId

            -- Delete the customer entry itself
            DELETE FROM [ConversionImport].[dbo].[Customer]
            WHERE CustomerId IN (SELECT d.CustomerId FROM deleted d)
      END
END
```

T2. Delete_File_Template

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between FileTemplate and TableTemplate & between
FileTemplate & ImportFile
-- =============================================
ALTER TRIGGER [dbo].[Delete_File_Template]
   ON  [ConversionImport].[dbo].[FileTemplate]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      -- Delete importFile entries linked to the fileTemplate
      entry(enforces CASCADE)
      DELETE FROM [ConversionImport].[dbo].[ImportFile]
      FROM [ConversionImport].[dbo].[importFile] AS imf
      INNER JOIN deleted d ON imf.FileTemplateId = d.FileTemplateId
```

```
      -- Delete tableTemplate entries linked to the
      fileTemplate (enforces CASCADE 1)
      DELETE FROM [ConversionImport].[dbo].[TableTemplate]
      FROM [ConversionImport].[dbo].[TableTemplate] AS tt
      INNER JOIN deleted d ON tt.FileTemplateId =
      d.FileTemplateId

      -- Delete the fileTemplate entry itself
      DELETE FROM [ConversionImport].[dbo].[FileTemplate]
      WHERE FileTemplateId IN (SELECT d.FileTemplateId FROM deleted d)
END
```

T3. Delete_Table_Template

```
-- ============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between TableTemplate and ImportRow
-- ============================================
ALTER TRIGGER [dbo].[Delete_Table_Template]
   ON  [ConversionImport].[dbo].[TableTemplate]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      --Delete importRow entries linked to the tableTemplate
      DELETE FROM [ConversionImport].[dbo].[ImportRow]
      FROM [ConversionImport].[dbo].[ImportRow] AS ir
      INNER JOIN deleted d ON ir.TableTemplateId = d.TableTemplateId

      -- Delete the tableTemplate entry itself
      DELETE FROM [ConversionImport].[dbo].[TableTemplate]
      WHERE TableTemplateId IN (SELECT d.TableTemplateId FROM deleted d)
END
```

T4. Delete_Import

```
-- ============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between Import and ImportFile
ALTER TRIGGER [dbo].[Delete_Import]
   ON  [ConversionImport].[dbo].[Import]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;
```

```
      IF @@ROWCOUNT = 0 RETURN

      BEGIN
            -- Delete importFile entries linked to the import entry(enforces
CASCADE)
            DELETE FROM [ConversionImport].[dbo].[ImportFile]
            FROM [ConversionImport].[dbo].[importFile] AS imf
            INNER JOIN deleted d ON imf.ImportId = d.ImportId

            -- Delete the import entry itself
            DELETE FROM [ConversionImport].[dbo].[Import]
            WHERE ImportId IN (SELECT d.ImportId FROM deleted d)
      END
END
```

T5. Delete_Import_File

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between ImportFile and ImportRow
-- =============================================

ALTER TRIGGER [dbo].[Delete_Import_File]
   ON  [ConversionImport].[dbo].[ImportFile]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN


                  -- Delete importRow entries linked to the
importFile(enforces first CASCADE)
                  DELETE FROM [ConversionImport].[dbo].[ImportRow]
                  FROM [ConversionImport].[dbo].[ImportRow] AS ir
                  INNER JOIN deleted d ON ir.ImpFileId = d.ImpFileId

                  -- Delete the importFile entry itself
                  DELETE FROM [ConversionImport].[dbo].[ImportFile]
                  WHERE ImpFileId IN (SELECT d.ImpFileId FROM deleted d)
END
```

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage J

Analyserapport Iteratie 2: Validatie-engine

Versie 1.0

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 26-03-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	16-03-2012	Concept	+ Inleiding + Hoofdstukindeling
0.2	18-03-2012	Concept	+ Eis m.b.t. invoer validatiesysteem + Functionele eisen m.b.t. soorten validatieregels
0.3	20-03-2012	Concept	+ Validatieregels ingedeeld indeling in 3 validatieniveaus + Eerste versie use case diagram
1.0	26-03-2012	Eerste versie	+ Use case diagram conform analyserapport iteratie 1 + Vier validatieniveas ipv drie validatieniveaus + Alleen foutieve resultaten vastleggen (F23)

Gerelateerde documenten	Versie	Datum
Analyserapport Iteratie 1: Conversiesysteem	1.0	23-02-2011

Inhoudsopgave

1. Inleiding 4
2. Requirements 5
2.1 Functionele eisen 5
2.2 Niet-functionele eisen 9
3. Use cases (UML) 11
3.1 Use case diagram 11
3.2 Use case beschrijvingen 11
3.2.1 Valideer import 11
3.2.2 Ophalen bestandsregels 12
3.2.3 Inladen validatieregels 12
3.2.4 Vastleggen resultaten 13
4. Data quality framework Artena 14
4.1 Definitie datakwaliteit 14
4.2 Kwaliteitsaspecten 14
Bijlage J.1 IDQ model 16
Bijlage J.2 PSP-IQ model 20

1. Inleiding

In dit document wordt de uitgevoerde analyse van de tweede iteratie (de validatie-engine) beschreven. Dit tweede onderdeel van de Artena Data Validator zal de grootste stap zijn in het verbeteren van de gegevenstransfer bij Artena.

Het grootste deel van deze analyse omvat de definitie van requirements. Door de requirements vast te leggen voor de validatie-engine is vooral veel duidelijk geworden over de verwachte invoer en uitvoer van het systeem en de soorten validaties die uitgevoerd moeten worden.

De requirements voor de validatie-engine zijn vastgesteld door overleg te voeren met meerdere stakeholders, zodat veel duidelijk is geworden over hun visie op de validatie-engine en de mogelijkheden die het systeem moet hebben.

In dit document zullen eerst de requirements zoals deze zijn opgesteld in samenwerking met de stakeholders uiteengezet worden (**hoofdstuk 2**). Vervolgens worden de UML use case diagrammen die zijn opgesteld vanuit deze requirements beschreven (**hoofdstuk 3**). Tot slot wordt een beschrijving gegeven van de theoretische onderbouwing voor een validatie-engine vanuit het oogpunt van datakwaliteit een mogelijke data-quality-framework (**hoofdstuk 4**).

NB1.*Dit rapport bevat ook de functionele eisen van het conversiesysteem zoals deze zijn gedefinieerd in de eerste iteratie omdat de samenhang tussen de validatie-engine en het conversiesysteem zeer groot is.*

NB2. *Dit rapport bevat de nieuwe requirements structuur die is ingevoerd na feedback van de begeleidende examinator. De nieuwe structuur zorgde voor een veel duidelijkere en meer SMART beschrijving van de requirements. Deze feedback heeft plaatsgevonden op 28-03-2012. In het analyserapport van de laatste iteratie (3) waren de requirements dan ook op deze manier ingedeeld. Om het verhaal consistent te houden zijn alle analyserapporten op deze manier bijgewerkt.*

2. Requirements

De validatie-engine is het tweede en tevens belangrijkste onderdeel in het verbeteren van gegevenstransfer bij Artena. De validatie-engine zal de inhoud van een import (van een klantbestand) die door het conversiesysteem gegenereerd is, moeten valideren. De validatie-engine zal de import moeten kunnen valideren met behulp van een set van validatieregels en zal de resultaten van deze validaties moeten vastleggen.

De eerste (functionele) eisen zijn vastgelegd door gesprekken te voeren met de probleemeigenaar (stakeholder 2) om zo de verschillende soorten validatieregels die uitgevoerd moesten worden vast te stellen. Met behulp van stakeholder 3 is bepaalde visie vastgesteld voor de wijze waarop de validatieregels uitgevoerd zouden kunnen worden in de programmatuur. Tot slot heeft stakeholder 4 vanuit de business intelligence kant nog enkele toevoegingen gedaan op de verschillende soorten validatieregels.

Vanuit de gegevens verkregen uit bovenstaande handelingen is vervolgens een lijst met requirements opgesteld en geprioriteerd. Met behulp van de requirements is dus een helder beeld ontstaan van de eisen waaraan de validatie-engine moest voldoen. Deze eisen komen in dat opzicht ook overeen met de wensen van alle stakeholders (opdrachtgever(S1), probleemeigenaar/projectbegeleider(S2), developers Artena(S3) en business intelligence(S4)).

Voor dit subsysteem geldt dat flexibiliteit een grote waarde heeft. Het systeem van validatieregels (en de afhandeling daarvan) moet zo zijn opgebouwd dat uitbreidingen en/of wijzigingen aan validatieregels geen grote impact hebben op de processen die in validatie-engine worden uitgevoerd.

2.1 Functionele eisen

Per functionele eis in dit hoofdstuk is aangegeven van welke stakeholder deze eis afkomstig is. Daarnaast worden de functionele eisen beschreven in dit hoofdstuk worden ingedeeld naar prioriteit. Om de eisen naar prioriteit in te delen zal gebruik gemaakt worden van de MoSCoW methode. Met behulp van deze methode wordt het mogelijk om de eisen in te delen in de volgende categorieën:

1. **Must have:** deze eisen worden in ieder geval binnen de gemaakte timebox/planning verwerkt in het systeem
2. **Should have**: deze eisen zijn net zo belangrijk als de must have eisen. Echter is het voor deze eisen niet verplicht om dat binnen vastgelegde timebox te doen.
3. **Could have:** een dergelijke eis is gewenst maar niet noodzakelijk, deze eis kan meegenomen als tijd en kosten dat toelaten.
4. **Won't have**: deze eisen zullen niet worden verwerkt in het te implementeren systeem.

Bovenstaande categorieën zijn voorgelegd aan de stakeholders terwijl door de eisen is heen gelopen. Per eis is vervolgens een prioriteit toegekend.

De eisen hieronder zijn genummerd zodat er vanuit andere documenten (zoals het ontwerprapport) kan worden terugverwezen naar de eisen.
Onderstaande eisenlijst bevat ook de eisen van het conversiesysteem aangezien dat systeem veel samenhang heeft met de validatie-engine (de uitvoer van het conversiesysteem wordt gebruikt voor de invoer van de validatie-engine).

LEGENDA

Nr.	= nummer van de eis
Systeem	= subsysteem of deliverable waar de eis op van toepassing is
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Eis	= de eis zelf

Nr.	Systeem	St.	Pr.	Eis
F1	Conversie	S1, S2	1	Invoer van conversiesysteem: • Één of meerdere klantbestanden **{vervolg in F2, F3, F4}** • Één of meerdere verwijzingen naar de templates waarin de benodigde gegevens voor een conversie van een klantbestand staan **{vervolg in F5}**
F2	Conversie	S2	1	Soorten klantbestanden te ondersteunen door conversiesysteem: • Excel (xls of xlsx) • CSV
F3	Conversie	S3	3	Het conversiesysteem moet de conversie van XML klantbestanden uit kunnen voeren wanneer deze genoeg gegevens bevatten en complex genoeg zijn zodat kan worden voldaan aan de uitvoer van **{vervolg in F6}.**

F4	Conversie	S2	1	Structuur klantbestanden voor invoer conversiesysteem:
				• Een bestand bestaat uit één of meerdere tabellen
				  ○ Een tabel bestaat uit één of meerdere kopregels plus nul of meerdere rijen
				• Structuur rij in een tabel:
				  ○ (xls/xlsx) nul of meerdere velden
				  ○ (csv) nul of meerdere velden gescheiden door een scheidingsteken
				• Een veld heeft een datatype:
				  ○ Datum of getal of double of string

F5	Conversie	S2	1	Gegevens benodigd in een template om conversie uit te voeren: • Aantal tabellen aanwezig in bestand (minimaal één) • Bestandstype (xls of xlsx of csv)

F6	Conversie	S2	1	Uitvoer van een conversieproces (de import):
				• Database met daarin:
				  ○ Naam klantbestand
				  ○ Nul of meerdere rijen uit klantbestand **{vervolg in F7}**

F7	Conversie	S2	1	Kenmerken van een rij in uitvoer van een conversie: • Nummer van de tabel

				• Nummer van de rij • Nul of meerdere velden gescheiden door een komma (volgens de RFC-4180 standaard[1])
F8	Conversie	S2	1	Formaten van uitvoervelden van xls/xlsx conversies moeten voldoen aan onderstaande definities, zodat dit gelijk zal zijn: 1. Datum – 'dd-mm-yyyy hh:mm:ss' 2. Integer – 'xxxxxxx'(geen duizendtal separatoren) 3. Double – 'xx.xx' (geen duizendtal separatoren & gescheiden door punt i.p.v. komma)
F9	Conversie	S2, S3	1	Het conversiesysteem moet in staat zijn conversies voor bestanden van verschillende klanten parallel aan elkaar uit te voeren.
F10	Conversie	S2	1	Indien zich fouten voordoen tijdens een conversieproces (=één run van een conversie) moet het conversiesysteem de gegevens/kenmerken van deze fout kunnen vastleggen/loggen.
F11	Conversie	S2	1	Het conversiesysteem moet in staat zijn het bestandsformaat van het klantbestand automatisch te bepalen aan de hand van de inhoud van de meegeleverde template (of extensie) wanneer een klantbestand is geselecteerd.
F12	Conversie	S2	3	Het opstartmechanisme van het conversiesysteem moet gebaseerd zijn op een systeem dat een nieuwe aangeleverd klantbestand detecteert.
F13	Validatie	S2, S3	1	De invoer voor een uit te voeren validatie door de validatie engine bestaat uit onderstaande onderdelen: • Een deel van de uitvoer van een conversie (de import): ○ Nul of meerdere rijen met de structuur van **{F7}** • Een set met validatieregels op verschillende niveaus en van verschillende typen **{vervolg in F14, F15, F16 en F17}**
F14	Validatie	S2	1	Benodigde gegevens (naast condities) voor **table-level** (tabel niveau) validatie: • Tabelnummer Typen validaties op dit niveau: • Rows Range (aantal regels) ○ Condities: aantal regels in tabel ## is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####'
F15	Validatie	S2	1	Benodigde gegevens (naast condities) voor **row-level** validatie: • Tabelnummer • Rijnummer Typen validaties op dit niveau: • Field Count (aantal velden in rij) ○ Conditie: de rij heeft evenveel velden als er kolommen zijn vastgelegd in de template

[1] http://tools.ietf.org/html/rfc4180

F16	Validatie	S2, S4	1	Benodigde gegevens (naast condities) voor **column-level** validatie: • Tabelnummer • Kolomnummer Typen validaties op dit niveau: • Fields Range (aantal ingevulde velden): o Conditie: het aantal velden in kolom ## dat ingevuld is, is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####' • Uniqueness o Conditie: de waarde in kolom ## mag maar één keer voorkomen
F17	Validatie	S2, S3, S4	1	Benodigde gegevens (naast condities) voor **field-level** validatie: • Tabelnummer • Rijnummer • Kolomnummer Typen validaties op dit niveau: • Datatype o Conditie: waarde in veld is van het datatype 'datum' of 'double' of 'integer' of 'string' • Presence o Conditie: waarde in veld mag niet NULL zijn • Range o Conditie: waarde in veld is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####' • Format o Conditie: waarde in moet van een bepaald formaat o Mogelijke formaten: ▪ Datum – 'dd-mm-yyyy hh:mm:ss' ▪ Double – 'xx.xx' (gescheiden door punt i.p.v. komma) ▪ Integer – 'xxxxxxx' (geen duizendtal separatoren) • Lookup o Conditie: waarde in veld moet voldoen aan een waarde in een 'database—tabel—veld' • Relationship o Conditie: als 'waarde in veld 1 gelijk is aan #### dan waarde is waarde in veld 2 ingevuld'
F18	Validatie	S1, S2	1	Het systeem moet de mogelijkheid bevatten om *quality control validation* uit te voeren. Dit type validatie slaat op de wijze waarop een 'fout' resultaat wordt geïnterpreteerd. In dit geval geeft de fout een indicatie van de waarschijnlijkheid van de gevonden waarde. **{zie ook F23}**

F19	Validatie	S2	1	Indien een table-level validatie niet succesvol is, dat wil zeggen fout, dan zullen de row-level, column-level –en field-level validaties voor die tabel niet meer worden uitgevoerd.
F20	Validatie	S2	1	Indien een row-level –of column-level validatie niet succesvol is, dat wil zeggen fout, dan zullen field-level validaties voor die rij of kolom niet meer worden uitgevoerd.
F21	Validatie	S2	1	Het moet mogelijk zijn om naast de in **{F14, F15, F16 en F17}** beschreven validatieregels nieuwe soorten/typen validatieregels op elk gewenst niveau toe te voegen wanneer dat gewenst is **{zie ook NF5}**.
F22	Validatie	S2	1	De validatie-engine moet in staat zijn validaties van verschillende klantbestanden parallel aan elkaar uit te voeren.
F23	Validatie	S2	1	Een validatieresultaat wordt alleen vastgelegd als de validatie een fout genereerd. (en het gevalideerde dus niet voldoet aan de condities)
F24	Validatie	S2	2	Een validatieresultaat kan van twee niveaus zijn: • Error: er is niet voldaan aan de validatie condities • Warning: er is niet voldaan aan de validatie condities maar het gecontroleerde gegeven is niet perse fout (waarschijnlijkheid)

F25	Validatie	S1, S2	1	Gegevens van een fout resultaat van een validatie moeten voldoen aan onderstaande criteria:
				• De validatieregel (type en niveau) en condities waar niet aan is voldaan
				• De reden waarom er niet is voldaan aan de condities
				• Het tabelnummer waarop de validatie betrekking had
				  ○ Table-level
				  ○ Row-level
				  ○ Column-level
				  ○ Field-level
				• Het rijnummer waarop de validatie betrekking had
				  ○ Row-level
				  ○ Field-level
				• Het kolomnummer waarop de validatie betrekking had
				  ○ Column-level
				  ○ Field-level
				• Het resultaat moet in zijn geheel leesbaar zijn voor de gebruikers (de klanten)

2.2 Niet-functionele eisen

In deze paragraaf worden de niet-functionele eisen (ook wel kwaliteitseisen genoemd) beschreven. De niet-functionele eisen beschrijven vaak extra eisen aan de wijze waarop bepaalde functionaliteiten uitgevoerd worden door het systeem. Een complete lijst van soorten niet-functionele eisen is te vinden in de ISO-9126 standaard[2]

[2] http://www.testconsultancy-groep.nl/index.php/downloads - zie ISO-9126 toegelicht

De categorieën gebruikt in de niet-functionele eisen:

1. Functionality compliance
2. Security
3. Changeability

LEGENDA

Nr. = nummer van de eis
Systeem = subsysteem of deliverable waar de eis op van toepassing is
St. = stakeholder van wie de eis afkomstig is
Pr. = prioriteit van de eis (MoSCoW)
Cat. = categorie van de niet-functionele eis (ISO-9126)
Eis = de eis zelf

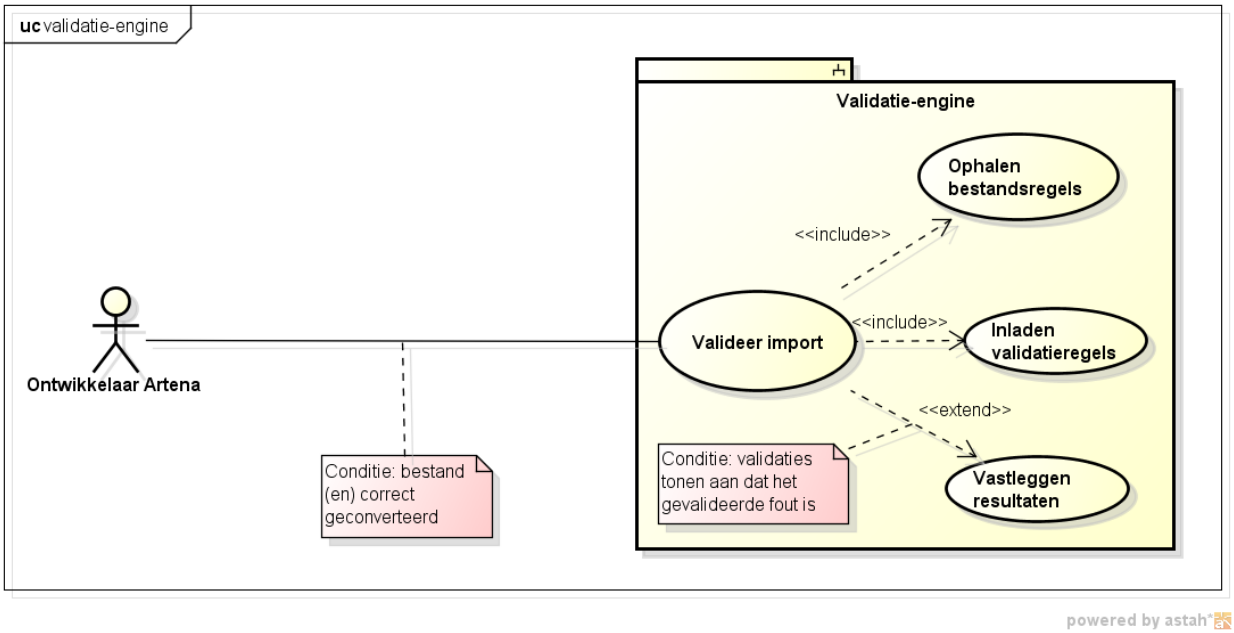
Nr.	Systeem	St.	Pr.	Cat.	Eis
NF1	Conversie	S2	1	1	De uitvoer van een conversie voldoet zoals in **{F7}** beschreven aan de RFC-4180[3] standaard voor Comma Separated Values. Dit houdt in dat: • Individuele velden in een rij zijn omgeven door dubbele quotes (''item'') • Individuele velden zijn van elkaar gescheiden door een komma (,) • Het laatste veld wordt niet afgesloten met een komma
NF2	Conversie	S2	1	1	Het conversiesysteem moet de ASCII standaard[4] volledig ondersteunen. Dit houdt in dat het systeem de tekens uit deze karakterset kan herkennen en verwerken.
NF3	Conversie	S2	1	3	De flexibiliteit van het conversiesysteem moet zodanig zijn dat een ondersteuning van een nieuw bestandsformaat geen grote impact moet hebben op het conversieproces, de wijze waarop het conversieproces omgaat met verschillende soorten bestanden zal dus generiek moeten zijn.
NF4	Conversie & Validatie	S3	1	2	**Algemene security eis van Artena:** De gegevens van de uitvoer van een conversie moeten na validatie verwijderd kunnen worden (omdat deze gegevens meestal privacy gevoelige informatie bevatten)
NF5	Validatie	S2	1	3	De flexibiliteit van de validatie-engine moet zodanig zijn dat het toevoegen/wijzigen van de validatieregels en validatietypen geen grote impact heeft op het proces dat doorlopen wordt in de validatie-engine bij de uitvoer van een validatie

[3] http://tools.ietf.org/html/rfc4180
[4] http://ascii-table.com/ansi-codes.php

3. Use cases (UML)

UML use cases worden gebruikt om aan te geven welke interacties bepaalde actoren zullen hebben met de validatie-engine. Met behulp van de use case beschrijvingen is het mogelijk om bepaalde functionele eisen te specificeren. §3.1 toont het use case diagram van toepassing op de validatie-engine. In §3.2 worden per use case de use case beschrijvingen gegeven.

3.1 Use case diagram



Figuur 1: Use case diagram

De use cases beschreven in het use case diagram zijn alleen uit te voeren wanneer het conversiesysteem een correcte conversie heeft uitgevoerd op de aangeleverde bestanden. Alleen dan kan de validatie op de *import* daadwerkelijk worden uitgevoerd.

3.2 Use case beschrijvingen

Bij de use case beschrijvingen in deze paragraaf wordt bij elke beschrijving vermeld op welke functionele eisen de beschrijving van toepassing is.

3.2.1 Valideer import

De eerste use case is van toepassing op alle eisen gesteld voor de validatie-engine. Deze use case omvat dan ook het gehele validatieproces waaronder de uitvoer van validaties op de verschillende validatieniveaus.

Naam	Valideer import (functionele eisen **F13 – F25)**
Aannamen	De import is correct weggeschreven door het conversiesysteem en de import(verwijzing) is bekend bij de ontwikkelaar van Artena
Beschrijving	(1) De ontwikkelaar selecteert de te valideren import en start de validatie m.b.v. een button in een interface

	(2) **Use case 'ophalen bestandsregels'** (3) **Use case 'inladen validatieregels'** (4) De validatie-engine start de uit te voeren validaties op table-level (5) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (6) **Use case 'inladen validatieregels'** (7) De validatie-engine start de uit te voeren validaties op row-level **[I]** (8) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (9) **Use case 'inladen validatieregels'** (10)De validatie-engine start de uit te voeren validaties op column-level **[I]** (11)**Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (12)**Use case 'inladen validatieregels'** (13)De validatie-engine start de uit te voeren validaties op field-level **[I] [II]** (14)**Use case 'vastleggen resultaten' [Extension point: fout na validatie]**
Uitzondering	**[I]** Validatie op table-level is niet succesvol, waardoor de validatie-engine de validatieregels op de andere validatieniveaus niet zal uitvoeren en het validatieproces zal afronden. **[II]** Validaties uitgevoerd op bijbehorend column –of row-level zijn niet succesvol, waardoor de validatie-engine de validatieregels op het betreffende veld niet uit zal voeren.
Resultaat	De bestandsregels uit van de import zijn gevalideerd en eventuele resultaten van de validaties zijn vastgelegd in de database.

3.2.2 Ophalen bestandsregels

Naam	Ophalen bestandsregels (functionele eis **F13**)
Aannamen	De database waarin de bestandsregels aanwezig in zijn is beschikbaar en bereikbaar
Beschrijving	(1) Het systeem haalt de bestandsregels op met behulp van verkregen verwijzing naar de import (2) Het systeem retourneert de opgehaalde gegevens
Uitzondering	
Resultaat	De gegevens zijn opgehaald uit de database en geretourneerd

3.2.3 Inladen validatieregels

Naam	Inladen validatieregels (functionele eis **F13**)
Aannamen	De validatieregels zijn beschikbaar en bereikbaar in de database
Beschrijving	(1) Op basis van de het validatieniveau (table, row, column of field) en de identificatie van tabellen, regels, kolommen en rijen worden de juiste validatieregels opgehaald (2) Het systeem zet de validatieregels om in bepaalde objecten zodat de validatie-engine de regels kan uitvoeren (3) Het systeem retourneert de validatie 'objecten' naar de validatie-engine
Uitzondering	
Resultaat	Validatieregels zijn ingeladen en bruikbaar voor de validatie-engine

3.2.4 Vastleggen resultaten

Naam	Vastleggen resultaten (functionele eisen **F23 – F25**)
Aannamen	De database waarin de resultaten vastgelegd moeten worden is beschikbaar en bereikbaar
Beschrijving	(1) Het systeem vraagt aan de gebruikte validatie objecten om de resultaten van de validatie op de juiste wijze weg te schrijven (volgens de eisen) (2) De resultaten worden samengesteld weg geschreven naar de database
Uitzondering	
Resultaat	De resultaten van de validaties zijn vastgelegd in de database

4. Data quality framework Artena

Naast de gedane analyse door de definitie van de requirements kwam vanuit stakeholder 4 een andere vraag. Sinds enige tijd is stakeholder 4 bezig met het ontwikkelen van een data quality framework om aan te kunnen bieden bij klanten.

De reden hiervoor is de steeds grotere vraag vanuit klanten (veelal financiële instellingen) naar kwaliteit van data. Veel klanten weten niet wat de kwaliteit van hun data is of willen deze verbeteren. Het antwoord op deze vragen ligt in de ontwikkeling van een data quality framework, waarin alle aspecten (die van belang zijn voor de klanten) van datakwaliteit behandeld en verbeterd kunnen worden.

Vanuit de consultancykant wordt hiervoor bijvoorbeeld gekeken naar wet –en regelgeving op het gebied van datakwaliteit. Daarnaast wordt gekeken naar mogelijke oplossingen om op verschillende (business)niveaus de datakwaliteit te kunnen controleren. De validatie-engine (en de Artena Data Validator in zijn geheel) dragen ook bij aan verbeteren van datakwaliteit. Voor stakeholder 4 is vanuit een theoretisch oogpunt (m.b.v. bepaalde modellen) gekeken naar de aspecten van datakwaliteit die behandeld worden door de Artena Data Validator.

4.1 Definitie datakwaliteit

Het valideren van data is het (in meer of mindere mate) controleren van de kwaliteit van de data. Datakwaliteit kan worden omschreven als *'De mate waarin de gegevens/data geschikt zijn voor het doel waarvoor ze gebruikt moeten worden'.*

Bovenstaande definitie is grotendeels afgeleid van de definitie van kwaliteit volgens de Van Dale: '*Kwaliteit is de mate waarin iets geschikt is om voor een bepaald doel te gebruiken'.*

Kwaliteit van data kan meerdere manieren worden vastgesteld. Er zijn namelijk meerdere verschillende aspecten waarmee de geschiktheid van data wordt omschreven. Een veel gebruikt model dat datakwaliteit definieert is het datakwaliteitsmodel **IDQ**[5] wat weer een onderdeel van de **SmarTEST**[6] methode.

Aspecten in het IDQ model komen weer grotendeels overeen met de dimensies van informatie kwaliteit opgesteld door een groep onderzoekers aan de MIT (Massachusetts Institute of Technology; de heren Kahn, Strong en Wang). Het kwaliteitsmodel door hun opgesteld heet het **PSP/IQ** [7]model en is onderdeel van het Total Data Quality Management[8].

In bijlage J.1 is het IDQ model bijgevoegd en in bijlage J.2 is het PSQ/IQ model bijgevoegd. In §4.2 worden de aspecten van de datakwaliteit uit deze modellen die van toepassing zijn op de validatie-engine beschreven

4.2 Kwaliteitsaspecten

[5] http://www.smartest.nl/verdieping/kwaliteitsmodellen/idqinformatiekwaliteit
[6] http://www.smartest.nl/smartest
[7] http://web.mit.edu/tdqm/www/tdqmpub/KahnStrongWangCACMApr02.pdf
[8] http://mitiq.mit.edu/toTDQM.aspx

De aspecten waar de validatie-engine de data op zal valideren (op basis van het IDQ model):

- Integriteit (Mate waarin relaties tussen de data-elementen technisch en syntactisch juist en volledig zijn. Zie ook Structuur.)
- Volledigheid (Mate waarin alle gewenste en relevante gegevens beschikbaar zijn in de informatieverzameling)
- Nauwkeurigheid (Mate van detail waarmee de informatiekenmerken (attributen) worden beschreven)
- Plausibiliteit (Mate waarin de informatie geloofwaardig is en overeenkomt met de werkelijkheid buiten het informatiesysteem)
- Syntax (Mate waarin de informatie volgens de afgesproken conventies en domeinen is gerepresenteerd)
- Structuur (Mate waarin de informatie logisch is geordend en relaties tussen elementen zijn gelegd. Zie ook Integriteit.)
- Consistentie (Mate waarin al dan niet vastgelegde keuzes en principes consequent worden gehanteerd)
- Eenduidigheid (Mate waarin verschillende interpretaties van informatie uitgesloten zijn)

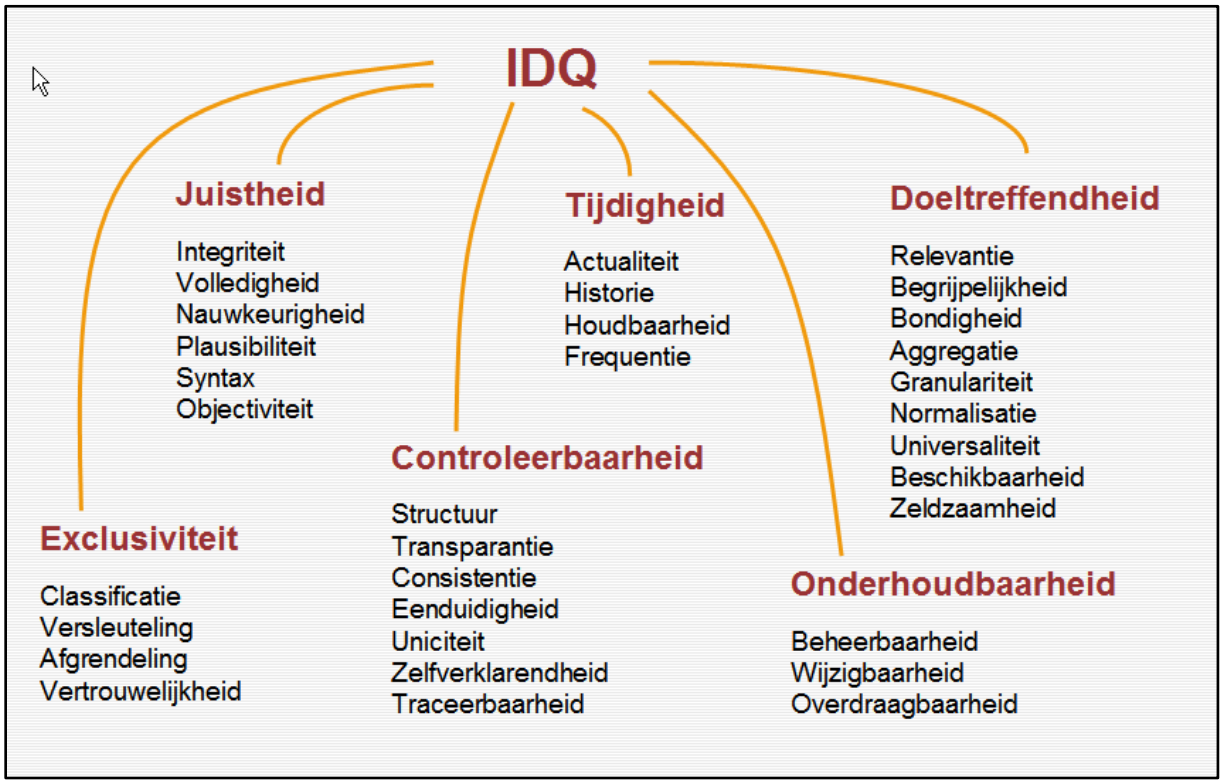
Op bovenstaande kwaliteitsaspecten faalt de aangeleverde data door klanten van Artena het meest (99%) en is ook de aanleiding tot constructie van de validatie-engine. Deze aspecten behoren vooral tot de *juistheid* en *controleerbaarheid* van de data. De tabel hieronder toont de overeenkomsten tussen soorten validatieregels gedefinieerd in de requirements en de hierboven beschreven aspecten van datakwaliteit.

Aspect	Validatieregel(s)	
Integriteit	Datatype	(field-level)
	Presence	(field-level)
	Range	(field-level)
	Lookup	(field-level)
Volledigheid	Rows Range	(table-level)
	Row Count	(row-level)
	Fields Range	(column-level)
Nauwkeurigheid	Datatype	(field-level)
	Range	(field-level)
Syntax	Datatype	(field-level)
	Format	(field-level)
	Lookup	(field-level)
Structuur	Uniqueness	(column-level)
	Relationship	(field-level)
Consistentie	Format	(field-level)
	Lookup	(field-level)
Eenduidigheid	Format	(field-level)
	Lookup	(field-level)

Het aspect plausibiliteit komt terug doordat een ondersteuning gegeven wordt voor deze fouten (**F18**).

Bijlage J.1 IDQ model

bron: http://www.smartest.nl/verdieping/kwaliteitsmodellen/idqinformatiekwaliteit



Kenmerk	Definitie	Synoniemen en verwante begrippen
JUISTHEID	Mate waarin de informatie qua presentatie en inhoud correct is vastgelegd	Betrouwbaarheid, Waarheid, Structuur
Integriteit	Mate waarin relaties tussen de data-elementen technisch en syntactisch juist en volledig zijn. Zie ook Structuur.	
Volledigheid	Mate waarin alle gewenste en relevante gegevens beschikbaar zijn in de informatieverzameling	
Nauwkeurigheid	Mate van detail waarmee de informatiekenmerken (attributen) worden beschreven	Accuraatheid. Zie ook Granulariteit.
Plausibiliteit	Mate waarin de informatie geloofwaardig is en overeenkomt met de werkelijkheid buiten het informatiesysteem	Waarheid, Juistheid
Syntax	Mate waarin de informatie volgens de afgesproken conventies, notatieregels en domeinen is gerepresenteerd	
Objectiviteit	Mate waarin de informatie objectief en neutraal is, vrij van subjectieve oogmerken en manipulatie	

Kenmerk	Definitie	Synoniemen en verwante begrippen
DOELTREFFENDHEID	Mate waarin de informatie zinvol en bruikbaar is voor een gegeven doel	Doelmatigheid, Effectiviteit (maar dit wordt soms bedoeld als efficiëntie, de kosten-batenverhouding dus).
Relevantie	Mate waarin de informatie nuttig is voor de gebruikers, gezien aard en doelstelling van de organisatie	Passendheid
Begrijpelijkheid	Mate waarin de informatie voor de doelgroep begrijpelijk en duidelijk is	
Bondigheid	Mate waarin de informatie beknopt en bondig is vastgelegd	
Aggregatie	Mate waarin de informatie op een zeker niveau is gegroepeerd, waaronder verder inzoomen niet mogelijk is	Tegenstelling: Granulariteit.
Granulariteit	Mate waarin individuele gegevenselementen tot een laag detailniveau afzonderlijk zijn vastgelegd	Atomariteit
Normalisatie	Mate waarin de verschillende informatietypen onafhankelijk van elkaar zijn vastgelegd	Gegevensordening
Universaliteit	Mate waarin de informatie gebruikt *kan* worden in een groter verband, binnen en buiten de organisatie (inherente eigenschap)	Standaardisatie
Verspreiding	Mate waarin de informatie *daadwerkelijk* gebruikt wordt in een groter verband, binnen en buiten de organisatie (niet-inherente eigenschap)	Persistentie, Ubiquiteit, (‘alomtegenwoordigheid’, denk aan *ubiquitous computing*)
Zeldzaamheid	Mate waarin de informatie ten opzichte van gegevensverzamelingen in andere systemen of organisaties uniek of zeldzaam is	Externe uniciteit
TIJDIGHEID	Mate waarin aan tijdgebonden aspecten van informatie recht is gedaan	
Actualiteit	Mate waarin de informatie actueel is en verouderde informatie is verwijderd	Tijdigheid
Historie	Mate waarin relevante historische informatie beschikbaar en toegankelijk is	
Houdbaarheid	Mate waarin de informatie voor langere tijd bruikbaar en relevant zal blijven	Levensduur, Geldigheid
Frequentie	Mate waarin gegevens met het geschikte interval zijn geregistreerd	Informatiefrequentie, Informatie-interval

Kenmerk	Definitie	Synoniemen en verwante begrippen
ONDERHOUD-BAARHEID	Mate waarin de informatie nu en in de toekomst beschikbaar en bruikbaar gehouden kan worden	
Beheerbaarheid	Mate waarin de informatie beheerbaar en onderhoudbaar is en zal blijven	
Wijzigbaarheid	Gemak waarmee de informatie gewijzigd kan worden zonder gevaar voor inconsistenties en onvolledigheid	
Overdraagbaarheid	Mate waarin de informatie overdraagbaar is naar andere systemen en organisaties	Verwerkbaarheid
EXCLUSIVITEIT	Mate waarin de informatie beveiligd is of kan worden tegen gebruik door onbevoegden	
Classificatie	Mate waarin gegevens van een (security-)classificatie zijn voorzien, waardoor diversificatie in toegang mogelijk wordt	
Versleuteling	Mate waarin de informatie is gecodeerd, waardoor zij voor onbevoegden geen waarde heeft	
Afgrendeling	Mate waarin de informatie, elektronisch of fysiek, is afgesloten voor benadering door onbevoegden	
Vertrouwelijkheid	Mate waarin de informatie voldoet aan normen en regels voor privacy en de daaraan verbonden regelgeving	
CONTROLEER-BAARHEID	Mate waarin de informatie geschikt is voor controle	
Structuur	Mate waarin de informatie logisch is geordend en relaties tussen elementen zijn gelegd. Zie ook Integriteit.	
Transparantie	Mate waarin de informatie overzichtelijk en toegankelijk is	
Consistentie	Mate waarin al dan niet vastgelegde keuzes en principes consequent worden gehanteerd	
Eenduidigheid	Mate waarin verschillende interpretatie uitgesloten is	
Uniciteit	Mate waarin redundantie en doublures binnen de informatieverzameling afwezig zijn	

Kenmerk	Definitie	Synoniemen en verwante begrippen
Zelfverklarendheid	Mate waarin de informatieverzameling als geheel zichzelf verklaart en beschrijft	Meta-informatie, metagegevens
Traceerbaarheid	Mate waarin de informatie herleid kan worden tot de bron, de locatie en het moment van ontstaan	

Bijlage J.2 PSP-IQ model

bron: http://web.mit.edu/tdqm/www/tdqmpub/KahnStrongWangCACMApr02.pdf

Belangrijkste gedeelte is vertaald naar Nederlands vanuit de bron

	In overeenstemming met de eisen	**Voldoet of overtreft klant verwachtingen**
Product kwaliteit	**Degelijke Informatie** De karakteristieken van de verschafte informatie komen overeen met informatiekwaliteit standaarden.	**Nuttige Informatie** De verschafte informatie voldoet aan de behoefte van de klanttaken.
Service kwaliteit	**Betrouwbare Informatie** Het proces van het converteren van data naar informatie maakt kennis met standaarden.	**Bruikbare Informatie** Het proces om data te converteren naar informatie overtreft de klantbehoeften.

Degelijke informatie:

De aspecten van kwaliteit in dit gebied zijn tastbaar en meetbaar tegenover een eisen of regels. Een informatiegebruiker heeft informatie nodig, dat representatief is en geen fouten bevat. Ontbrekende informatie kan leiden tot incorrecte conclusies en slechte beslissingen. Informatiegebruikers moeten weten van welke formaten gebruikt worden gemaakt om data representatief weer te geven.

Bijvoorbeeld: Een datum, zoals 01-01-2012 kan op verschillende manieren weergeven worden (1 January 2012 of January 1, 2012). .

Betrouwbare informatie:

De aspecten van datakwaliteit in dit gebied kunnen in het algemeen niet worden direct geëvalueerd, wanneer er geen overige informatie van andere systemen bekend is. Betrouwbare informatie is up-to-date, veilig en wordt bijvoorbeeld periodiek aangeleverd.

Nuttige informatie:

De aspecten in dit gebied kunnen alleen worden geëvalueerd vanuit een gebruikersoogpunt. Wanneer gebruikers niet alles over een persoon willen weten maar alleen voornaam en achternaam dan is een volledige uitdraai van een persoonsdossier niet nuttig. Informatie is dus nuttig wanneer deze is afgestemd op de gebruiker van de informatie.

Bruikbare informatie:

De kwaliteitsaspecten in dit gebied zijn taakafhankelijke karakteristieken. De informatie is relevant wanneer de gebruikers hun vereiste taken ermee kunnen uitvoeren.

Er zijn meerdere kwaliteitsaspecten (dimensies) per gebied in het PSP/IQ model toe te passen. Alle dimensies behorende tot dit model zijn in de tabel hieronder vastgelegd.

DIMENSIES	DEFINITIES
Toegankelijkheid	De mate waarin informatie beschikbaar of makkelijk en snel te verkrijgen is.
Geschikte hoeveelheid aan informatie	De mate waarin de hoeveelheid informatie geschikt is voor het te uit te voeren werkzaamheid.
Geloofwaardigheid	De mate waarin informatie gezien wordt als waar en geloofwaardig.
Compleet	De mate waarin er geen informatie wordt vermist en de informatie voldoende breedte- en dieptegang bied voor de te uit te voeren werkzaamheden.
Beknopte weergave	De mate waarin informatie beschikbaar of makkelijk en snel te verkrijgen is.
Consistente weergave	De mate waarin informatie in dezelfde formaat wordt gepresenteerd.
Makkelijk te manipuleren	De mate waarin informatie makkelijk te manipuleren is en toepasbaar is op andere taken.
Vrij van fouten	De mate waarin informatie correct en betrouwbaar is.
Verklarend	De mate waarin informatie in de juiste taal, symbolen, en eenheid staan en de definities helder zijn.
Objectiviteit	De mate waarin informatie onbevooroordeeld is.
Relevantie	De mate waarin informatie toepasselijk en behulpzaam is voor de taak waarvoor de informatie gebruikt wordt.
Reputatie	De mate waarin informatie hoog in het vaandel staat in termen van de bron of inhoud.
Beveiliging	De mate waarin de toegang tot informatie begrensd is om de veiligheid ervan te behouden.
Tijdigheid	De mate waarin informatie genoeg up-to-date is voor de taak die voorhanden ligt.
Begrijpelijkheid	De mate waarin informatie makkelijk te begrijpen is.
Waarde toevoegend	De mate waarin informatie gunstig is en voorziet van profijt door het gebruik ervan.

In onderstaande tabel zijn alle kwaliteitsaspecten/dimensies ingevuld per gebied.

	In overeenstemming met de eisen	**Voldoet of overtreft klant verwachtingen**
Product kwaliteit	**Degelijke Informatie** ● Vrij van fouten ● Compleet ● Beknopte weergave ● Consistente weergave	**Nuttige Informatie** ● Geschikte hoeveelheid aan informatie ● Relevantie ● Begrijpelijkheid ● *Objectiviteit* ● *Verklarend*
Service kwaliteit	**Betrouwbare Informatie** ● Tijdigheid ● Beveiliging	**Bruikbare Informatie** ● Geloofwaardigheid ● Toegankelijkheid ● Makkelijk te manipuleren ● Reputatie ● Waarde toevoegend

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage K

Ontwerprapport Iteratie 2: Validatie-engine

Versie 1.0

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 03-04-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	26-03-2012	Concept	Eerste versie ontwerpdocument op basis van analyserapport iteratie 2. + Specificatie eisen uit requirementsdocument m.b.t. validatieregels
0.2	29-03-2012	Concept	+ validatieflow + databaseontwerp validatieregels (EER & RRM) + databaseontwerp valdiatieresultaten (EER & RRM) + eerste versie klassendiagram
1.0	03-04-2012	Eerste versie	**Paragraaf 5.1:** De interface verantwoordelijk voor het definiëren van de functionaliteiten van een validatie plug-in is gewijzigd naar een abstracte klasse vanwege gebrek aan definities van attributen in een interface **Paragraaf 5.3:** Concrete implementaties van een validatie plug-in toegevoegd aan klassendiagram **Hoofdstuk 3**: Benamingen validaties aangepast + Uniqueness(kolom), presence(field) en datatype(field) validaties hebben geen conditie nodig + validatieflow op veld niveau beter beschreven en uitgebreid (**paragraaf 3.6.1**). **Hoofdstuk 4**: Relatie tussen Import en Customer verwijdert, bestaat al via de tabel FileTemplate + DefaultOrder toegevoegd aan ValidationType + referentiële integriteitsregels toegevoegd aan vreemde sleutel relaties in RIM dmv Triggers

Gerelateerde documenten	Versie	Datum
Analyserapport Iteratie 2: Validatie-engine	1.0	26-03-2012
Ontwerprapport Iteratie 1: Conversiesysteem	1.2	02-03-2012

Inhoudsopgave

1. Inleiding 5
2. Invoer en uitvoer validatie-engine 6
2.1 Invoer 6
2.2 Uitvoer 6
3. Validatieregels & Validatieresultaten 7
3.1 Table-level definities 8
3.1.1 Rows Range validatie 8
3.2 Row-level definities 8
3.2.1 Field Count validatie 8
3.3 Column-level definities 8
3.3.1 Fields Range validatie 8
3.3.2 Uniqueness validatie 9
3.4 Field-level definities 9
3.4.1 Datatype validatie 9
3.4.2 Presence validatie 9
3.4.3 Range validatie 10
3.4.4 Format validatie 10
3.4.5 Lookup validatie 10
3.4.6 Relationship validatie 10
3.5 Reguliere expressie 11
3.6 Validatieflow 11
3.6.1 Volgorde binnen één validatieniveau (field) 12
4. Databaseontwerp 14
4.1 Entity Relationship Diagram – Validatieregels 14
4.1.1 Relatie(s) met bestandstemplate 15
4.1.2 Volgorde 15
4.1.3 Condities 15
4.2 Entity Relationship Diagram – Validatieresultaten 16
4.3 Relationeel Representatie Model 17
4.4 Relationeel Implementatie Model & Referentiële integriteit 18
4.5 Databasediagram 18
5. Programmatuur 19
5.1 Plug-in systeem 19

5.2 LINQ-to-SQL & databasecommunicatie 20
5.3 Klassendiagram 20
5.3.1 Beschrijving klassen 21
5.4 Het validatieproces 22
5.4.1 Het inladen van een (validatie) plug-in 22
5.4.2 UML sequence diagram 22
6. Begrippenlijst 23
Bijlage K.1 - EER diagram validatieregels 24
Bijlage K.2 - EER Validatieresultaten 25
Bijlage K.3 - Relationeel Implementatie Model 26
Bijlage K.4 – Triggers 30
T1. Delete_Customer 30
T2. Delete_File_Template 30
T3. Delete_Table_Template 31
T4. Delete_Column_Template 32
T5. Delete_Import 33
T6. Delete_Import_File 34
T7. Delete_Import_Row 35
Bijlage K.5 – Databasediagram (visio) 36
Bijlage K.6 – UML sequence diagram validatieproces 37

1. Inleiding

In dit document wordt het ontwerp voor het tweede onderdeel van de Artena Data Validator beschreven, de validatie-engine. De validatie-engine controleert de uitvoer van het conversiesysteem (= de inhoud van de klantbestanden) met behulp van een set aan validatieregels. De validatie-engine is ook geïmplementeerd als een losse DLL (zie NB1).

Het analyserapport van deze tweede iteratie bevat ook de functionele eisen van het conversiesysteem aangezien de samenhang tussen dat systeem en de validatie-engine zeer zeker aanwezig is (beiden onderdeel van de Artena Data Validator). Deze samenhang zal pas volledig worden geïmplementeerd in de derde iteratie (afronding van de Artena Data Validator). De functionele eisen van het conversiesysteem zullen daarom niet worden meegenomen in dit ontwerprapport tenzij dat echt nodig is voor het ontwerp van de validatie-engine.

In **hoofdstuk 2** wordt een technische beschrijving van de invoer –en uitvoer van de validatie-engine gegeven in de vorm van logische expressies (ter verduidelijking van de eisen).

In **hoofdstuk 3** worden twee belangrijke onderdelen met betrekking tot het uitvoeren van validaties beschreven:

1. Uitgebreide beschrijving van de verschillende soorten validatieregels op verschillende niveaus, inclusief een mogelijk format van het resultaat dat een validatieregel kan genereren.
2. Uitgebreide beschrijving van de volgorde waarin validaties uitgevoerd zullen worden (flow)

Vervolgens zal in **hoofdstuk 5** de database structuur voor het vastleggen van validatieregels en validatieresultaten uiteengezet en gespecificeerd worden. De datastructuur voor de validatie-engine is een uitbreiding op de database van het conversiesysteem. Het (E)ER diagram zal dus ook een aantal tabellen van het conversiesysteem bevatten.

Nadat de datastructuren voor de validatie-engine volledig zijn gedefinieerd zal de programmatuur in **hoofdstuk 5** worden beschreven. Het plug-in systeem gebruikt in het conversiesysteem wordt in de validatie-engine ook toegepast (weliswaar op een andere manier). Daarnaast zal de structuur van de validatie-engine worden beschreven met behulp van UML klassendiagrammen en UML sequence diagrammen.

NB1 Functionele eis F22 wordt niet behandeld in dit document, maar is reeds behandeld in hoofdstuk 2 (‘Dynamic Linked Library’) ontwerprapport van de eerste iteratie waarin de implementatie van het conversiesysteem is vastgelegd.

NB2 Functionele eis NF4 wordt niet behandeld in dit document, omdat het verwijderen van de uitvoer van een conversie niet verplicht is om te implementeren (prioriteit 2) in de scope van dit project. Bovendien zou verwijdering van die gegevens ertoe leiden dat het samenstellen van de validatieresultaten uit de database niet meer mogelijk is.

2. Invoer en uitvoer validatie-engine

*Functionele eisen **F12 – F16***

In dit hoofdstuk wordt ter verduidelijking de gespecificeerde invoer en uitvoer (zie analyserapport) herhaalt en beschreven in de vorm van de logische expressies.

2.1 Invoer

De invoer van de validatie-engine bestaat uit een import (bestandsregels uit een bestand) die gegenereerd is door het conversiesysteem uit één of meerdere klantbestanden. Daarnaast behoren één of meerdere validatieregels tot de invoer van de validatie-engine.

Invoer validatie: (rij)* + ((validatieregel)+)

Rij: tabelnummer + rijnummer + ((veld + “,”)* + veld)
Validatieregel: validatieniveau + validatietype + ((conditie)+)
Validatieniveau: (file | table | row | column | field)
Validatietype: (datatype | presence | …)

2.2 Uitvoer

De uitvoer van de validatie-engine bestaat uit nul of meerdere validatieresultaten. Deze resultaten moeten door de validatie-engine worden gegenereerd wanneer validaties niet succesvol (oftewel fout) zijn.

Uitvoer validatie: (validatieresultaat)*

Validatieresultaat: validatieniveau + validatietype + resultaatbeschrijving

3. Validatieregels & Validatieresultaten

*Functionele eisen **F13 – F18, F25***

Dit hoofdstuk beschrijft de validatieregels en validatieresultaten die zijn gespecificeerd in het Analyserapport. §3.1 tot §3.4 geven gedetailleerde beschrijvingen van de validatieregels op de te ondersteunen niveaus (behalve file-niveau). In §3.5 wordt een uitleg gegeven over het gebruik van *reguliere expressies* bij sommige validatieregels. In §3.6 wordt de stroming/volgorde tussen deze validaties beschreven.

Er zijn vier niveaus van validaties vastgelegd in de requirements:

1. Table-level
2. Row-level
3. Column-level
4. Field-level

Naast deze vier niveaus wordt ook voor mogelijk gehouden om een vijfde niveau van validatie uit te voeren, namelijk op bestandsniveau (file-level). Het uitvoeren van een validatie over een heel bestand is echter nog niet nodig omdat de inhoud van bestand dan al is geconverteerd door het conversiesysteem. Een file-level validatie zal dan ook (nog) niet worden geïmplementeerd in de validatie-engine. De figuren 1, 2, 3 en 4 tonen de verschillende gebieden waarop de verschillende validatieniveaus van toepassing zijn.

	Kolom 1	**Kolom 2**
Rij 1	Veld 1	Veld 2
Rij 2	Veld 3	Veld 4

Figuur 1: Table-level

	Kolom 1	**Kolom 2**
Rij 1	Veld 1	Veld 2
Rij 2	Veld 3	Veld 4

Figuur 2: Row-level

	Kolom 1	**Kolom 2**
Rij 1	Veld 1	Veld 2
Rij 2	Veld 3	Veld 4

Figuur 3: Column-level

	Kolom 1	**Kolom 2**
Rij 1	Veld 1	Veld 2
Rij 2	Veld 3	Veld 4

Figuur 4: Field-level

Per validatieniveau zullen er één of meerdere validatieregels geïmplementeerd worden in de validatie-engine. Per validatieregel wordt aangegeven wat de kenmerken zijn (benodigde gegevens) en welk resultaat de regel moet genereren.

NB. *De quality control validation (functionele eis F19) is niet beschreven in de paragrafen 3.1-3.4 omdat deze wordt geïmplementeerd door een andere foutwaarde (warning i.p.v. error) toe te kennen aan een validatieresultaat.*

3.1 Table-level definities

3.1.1 Rows Range validatie

Kenmerken invoer *(overeenkomst met range validatie op field-level)*

- Tabelnummer: ##
- Aantal regels: ######
- Conditie: (== aantal) | (!=aantal) | (> aantal) | (<aantal) | (>= aantal) | (<= aantal) | (getal1 <> getal2) | (%(aantal)) | (== *regular expression*[1]- **zie §3.5**)

(Mogelijke) Kenmerken resultaat

"Het aantal regels in tabel ## van bestand ## voldoet niet aan conditie"
"Gevonden aantal regels: ##"
"Verwacht aantal regels: *conditie"*

Conditie: (== aantal) | (!=aantal) | (> aantal) | (<aantal) | (>= aantal) | (<= aantal) | (getal1 <> getal2) | (%(aantal)) | (== regular expression)

3.2 Row-level definities

3.2.1 Field Count validatie

Kenmerken invoer

- Tabelnummer: ##
- Regelnummer: ##
- Aantal velden in rij: ##
- Condities: (== aantal kolommen volgens template)

(Mogelijke) Kenmerken resultaat

"Het aantal velden in regel ## van tabel ## voldoet niet aan conditie"
"Gevonden aantal velden: ##"
"Verwacht aantal velden: *conditie"*

Conditie:(== aantal kolommen volgens template)

3.3 Column-level definities

3.3.1 Fields Range validatie

[1] http://www.regular-expressions.info/tutorial.html

Kenmerken invoer *(overeenkomst met range validatie op field-level)*

- Tabelnummer: ##
- Kolomnummer: ##
- Aantal ingevulde velden in kolom: ##
- Conditie: (== aantal) | (!=aantal) | (> aantal) | (<aantal) | (>= aantal) | (<= aantal) | (getal1 <> getal2) | (%(aantal)) | (== *regular expression*[2])- **zie §3.5**

(Mogelijke) Kenmerken resultaat
"Het aantal gevonden velden in kolom ## van tabel ## voldoet niet aan conditie"
"Gevonden aantal velden: ##"
"Verwacht aantal velden: *conditie"*

Conditie:(== aantal) | (!=aantal) | (> aantal) | (<aantal) | (>= aantal) | (<= aantal) | (getal1 <> getal2) | (%(aantal)) | (== *regular expression*)

3.3.2 Uniqueness validatie

Kenmerken invoer

- Tabelnummer: ##
- Kolomnummer: ##
- Waarden ingevuld in kolom: waarde1…waarde2...waarde3...waarde(n)

(Mogelijke) Kenmerken resultaat
"De waarde van het veld op regel ## in kolom ## van tabel ## is niet uniek"
"Gevonden waarde: ##"

3.4 Field-level definities

Voor elke field-level validatie is het tabelnummer, regelnummer en kolomnummer benodigd.

3.4.1 Datatype validatie

Kenmerken invoer

- Waarde in veld: mogelijk alle tekens uit ASCII of UNICODE tekenset
- Condities: (== date | double | integer | string)

(Mogelijke) Kenmerken resultaat
"De waarde in het veld op rij ##, kolom ## van tabel ## is niet van het juiste datatype"
"Verwacht datatype: *conditie"*

Conditie: (== date | double | integer| string)

3.4.2 Presence validatie

Kenmerken invoer

- Waarde in veld: mogelijk alle tekens uit ASCII of UNICODE tekenset

[2] http://www.regular-expressions.info/tutorial.html

- Conditie: *niet nodig, de aanwezigheid van dit type validatie geeft aan dat het veld verplicht ingevuld moet zijn.*

(Mogelijke) Kenmerken resultaat
"De waarde in het veld op rij ##, kolom ## van tabel ## is niet ingevuld terwijl dit verplicht is"

3.4.3 Range validatie

Kenmerken invoer

- Waarde in veld: mogelijk een getal of decimaal getal of datum
- Condities: (== getal1) | (!=getal1) | (>getal1) | (<getal1) | (>=getal1) | (<=getal1) | (getal1 <> getal2) | (%(getal1)) | (== regular expression) - **zie §3.5**

(Mogelijke) Kenmerken resultaat
"De waarde in het veld op rij ##, kolom ## van tabel ## past niet in de verwachte range"
"Gevonden waarde = ##"
"Verwacht range: *conditie"*

3.4.4 Format validatie

Kenmerken invoer

- Waarde in veld: mogelijk alle tekens uit ASCII of UNICODE tekenset
- Condities: (== regular expression) -**zie §3.5**

(Mogelijke) Kenmerken resultaat
"De waarde in het veld op rij ##, kolom ## van tabel ## is niet van het juiste formaat"
"Gevonden waarde/format = ##"
"Verwacht formaat: *conditie"*

Conditie: (== *regular expression*)

3.4.5 Lookup validatie

Kenmerken invoer

- Waarde in veld: mogelijk alle tekens uit ASCII of UNICODE tekenset
- Condities: (== waarde uit database—lookuptabel—veld)

(Mogelijke) Kenmerken resultaat
"De waarde in het veld op rij ##, kolom ## van tabel ## komt niet overeen met een lookup value"
"Gevonden waarde = ##"
"Verwachte waarde = *conditie"*

Conditie: (== waarde uit database—lookuptabel—veld)

3.4.6 Relationship validatie

Kenmerken invoer

- Waarde in veld: mogelijk alle tekens uit ASCII of UNICODE tekenset

- Condities: IF (waarde1 == range/format/lookup) THEN (waarde2, waardeN == NOT NULL)
  Hierbij is waarde 2 een waarde in een bepaalde tabel-regel-kolom-veld van een andere tabel uit dezelfde import.

(Mogelijke) Kenmerken resultaat

"De waarde in het veld2 op rij ##, kolom ## van tabel ## is niet ingevuld terwijl dat wel zo moet zijn omdat de waarde in veld1 op rij ##, kolom ## van tabel ## een bepaalde waarde heeft"
"Gevonden waarde (veld1) = conditie"

Conditie: IF (waarde1 == range/format/lookup) THEN (waarde2, waardeN == NOT NULL)
Hierbij is waarde 2 een waarde in een bepaalde tabel/regel/kolom/veld van een andere tabel uit dezelfde import

3.5 Reguliere expressie

Zoals beschreven kunnen format en range validaties ook worden uitgevoerd met behulp van een reguliere expressie. Het is uiteindelijk aan de daarvoor aangewezen plug-in om te bepalen of een reguliere expressie gebruikt wordt voor een validatie.

Met behulp van een reguliere expressie is het mogelijk om te controleren een stuk tekst (in dit geval een gegeven van een import van een klantbestand) voldoet aan een bepaald patroon. Het patroon is opgebouwd volgens bepaalde vooraf vastgestelde regels en is daarom vaak te gebruiken in meerdere programmeertalen.

- Een veld mag bijvoorbeeld alleen maar cijfers bevatten:
  - [0-9]*
- Een veld moet in het datum formaat dd/mm/yyyy zijn[3]:
  - (0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])[- /.](19|20)
- Een veld moet een waarde hebben tussen 0 en 999[4]:
  - ([0-9]|[1-9][0-9]|[1-9][0-9][0-9])

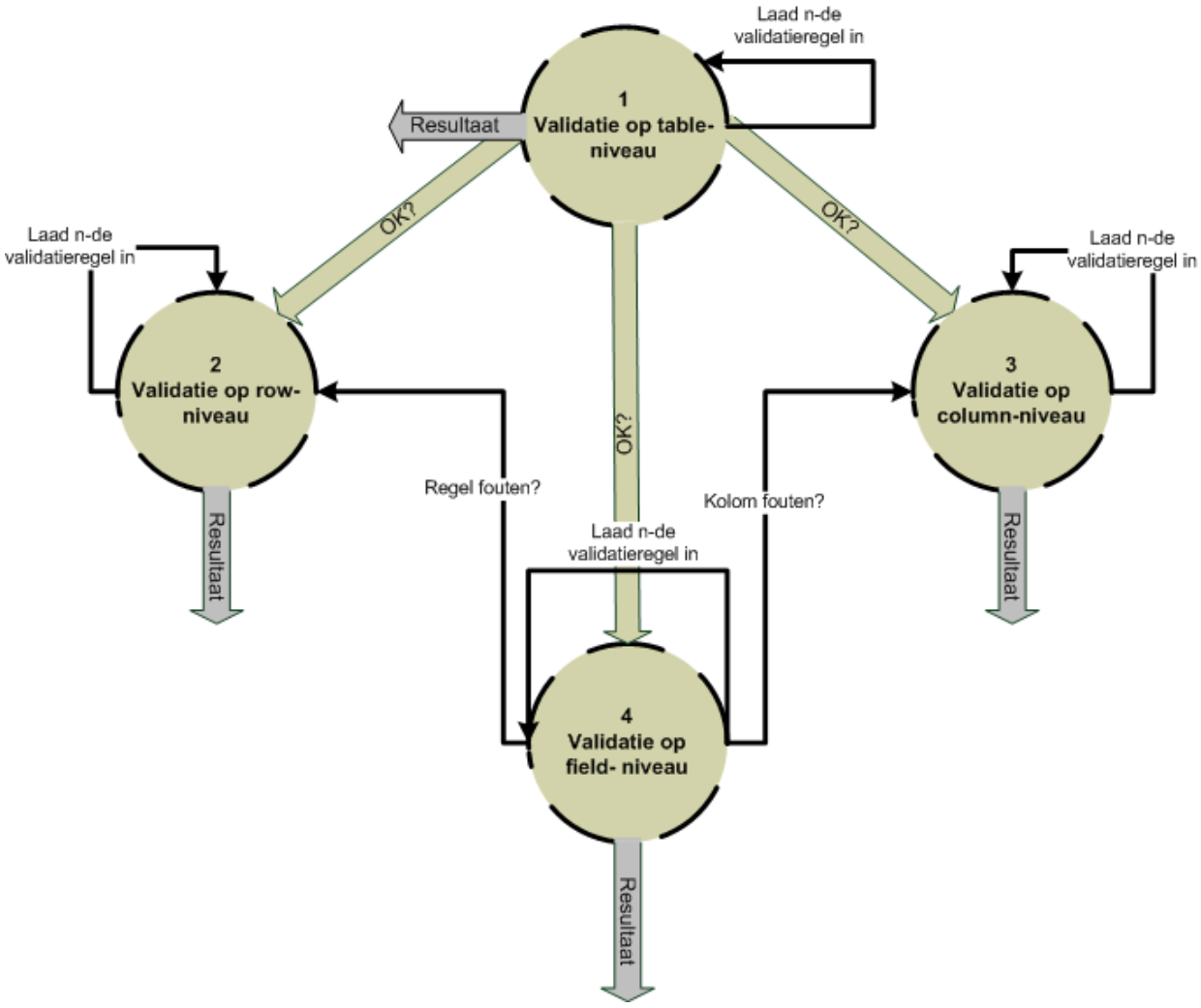
De validatie-engine kan met behulp van de daarvoor aangewezen functionaliteiten in het .NET framework een controle uitvoeren op deze expressies.

3.6 Validatieflow

*Functionele eisen **F19-F20***

Zoals beschreven worden validatieregels ingedeeld op verschillende niveaus. Sommige niveaus hebben een hogere prioriteit binnen een validatieproces dan andere validatieniveaus. Dit wil zeggen dat sommige validaties eerder dan andere validaties uitgevoerd moeten worden zodat een logische volgorde ontstaat. Deze volgorde noemen we de validatieflow. Zie figuur 5.

[3] http://www.regular-expressions.info/regexbuddy/datemmddyyyy.html
[4] http://www.regular-expressions.info/numericranges.html



Figuur 5: Validatieflow

De validatie-engine zal zoals te zien is de verschillende vormen van validatie op onderstaande wijze uitvoeren:

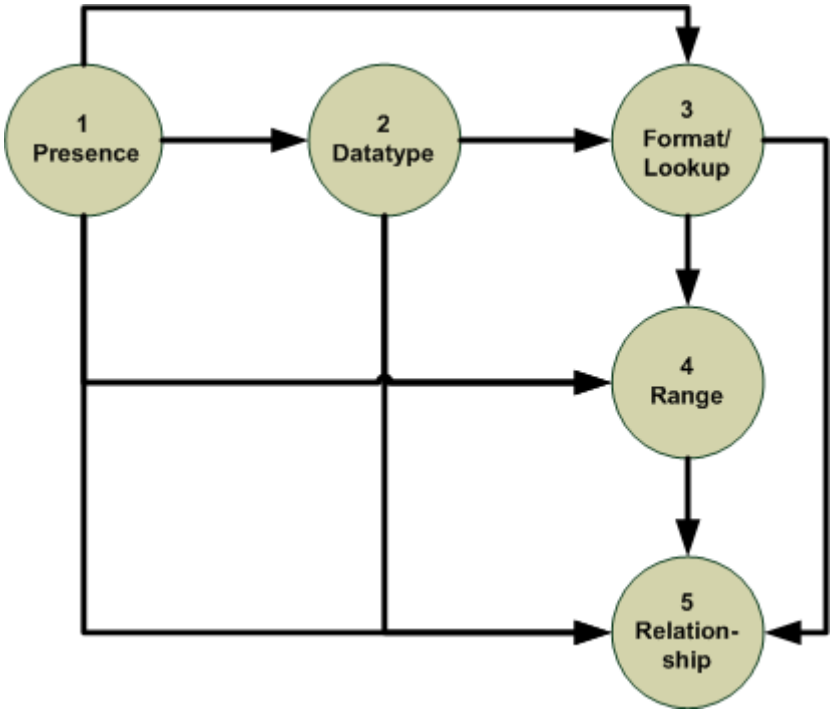
1. Validatie(s) op table- level
2. Validatie(s) op row-level /aantal kolommen in regel
3. Validatie(s) op column-level/aantal rijen in een kolom
4. Validatie(s) op field-level, waarbij wordt gekeken naar eventuele fouten gerapporteerd vanuit de row/column level validaties.

Op field-level wordt er gekeken naar de resultaten van validaties op row-level of column-level. Wanneer een bestandsregel niet goedgekeurd wordt omdat er bijvoorbeeld te weinig velden aanwezig zijn is dan is het ook niet mogelijk om de field-level validaties op die bestandsregel uit te voeren aangezien niet alle velden in de regel aanwezig zijn.

3.6.1 Volgorde binnen één validatieniveau (field)

Op field-level zijn er zeven typen/soorten validaties beschreven. Wanneer op één veld meerdere soorten validaties uitgevoerd zullen worden zal hierin ook een volgorde aanwezig zijn. Wanneer validatie immers aangeeft dat het datatype van het veld niet goed is dan heeft het ook geen zin de

andere validaties op dat veld uit te voeren. Elk van deze soorten validaties heeft een andere prioriteit en kan een volgende validatie overbodig maken. Zie figuur 7.



Figuur 6: volgorde field-level validaties

Zoals in figuur 7 is beschreven, is bijvoorbeeld het uitvoeren van een datatype validatie op een veld overbodig wanneer op datzelfde veld de presence validatie faalt. Wanneer bijvoorbeeld de presence validatie niet wordt toegepast op een veld (wat trouwens onwaarschijnlijk is) dan kan de datatype validatie als eerste uitgevoerd worden op een veld.

Afhankelijk van de regels die zijn toegevoegd voor een validatieroutine zal het systeem zelf de juiste volgorde moeten kiezen (op basis van figuur 7).

Deze volgorde is gebonden aan het type validatieregel. Het moet echter ook mogelijk zijn om de volgorde handmatig in te stellen bij het invoeren van validatieregels (zie *DefaultOrder* en *CustomOrder*, paragraaf 4.2)

4. Databaseontwerp

De database die gebruikt zal worden voor de opslag van gegevens benodigd door de validatie-engine en voor opslag van gegevens resulterend uit de validatieprocessen sluit aan op de database van het conversiesysteem. Dat gedeelte is reeds geïmplementeerd en beschreven in het ontwerprapport van de eerste iteratie. Het ERD van de database voor het conversiesysteem is bijgevoegd in bijlage A.

De database (het conversiesysteem en de validatie-engine) is dus verantwoordelijk voor het vastleggen/bijhouden van verschillende soorten gegevens:

- Klantgegevens
- Templates (invoer conversie/validatie)
- Geconverteerde bestandsregels (uitvoer conversie)
- Validatieregels (invoer validatie)
- Validatieresultaten (uitvoer validatie)

Het databaseontwerp van de datastructuur voor de validatie-engine ziet er wederom als volgt uit:

- *(E)ER diagram* (conceptueel databaseontwerp)
- *Relationeel Representatie Model* (logisch databaseontwerp)
- *Relationeel Implementatie Model* (logisch databaseontwerp – implementatie)
- Microsoft Visio 2010 databasediagram (logisch databaseontwerp – grafische weergave)

4.1 Entity Relationship Diagram – Validatieregels

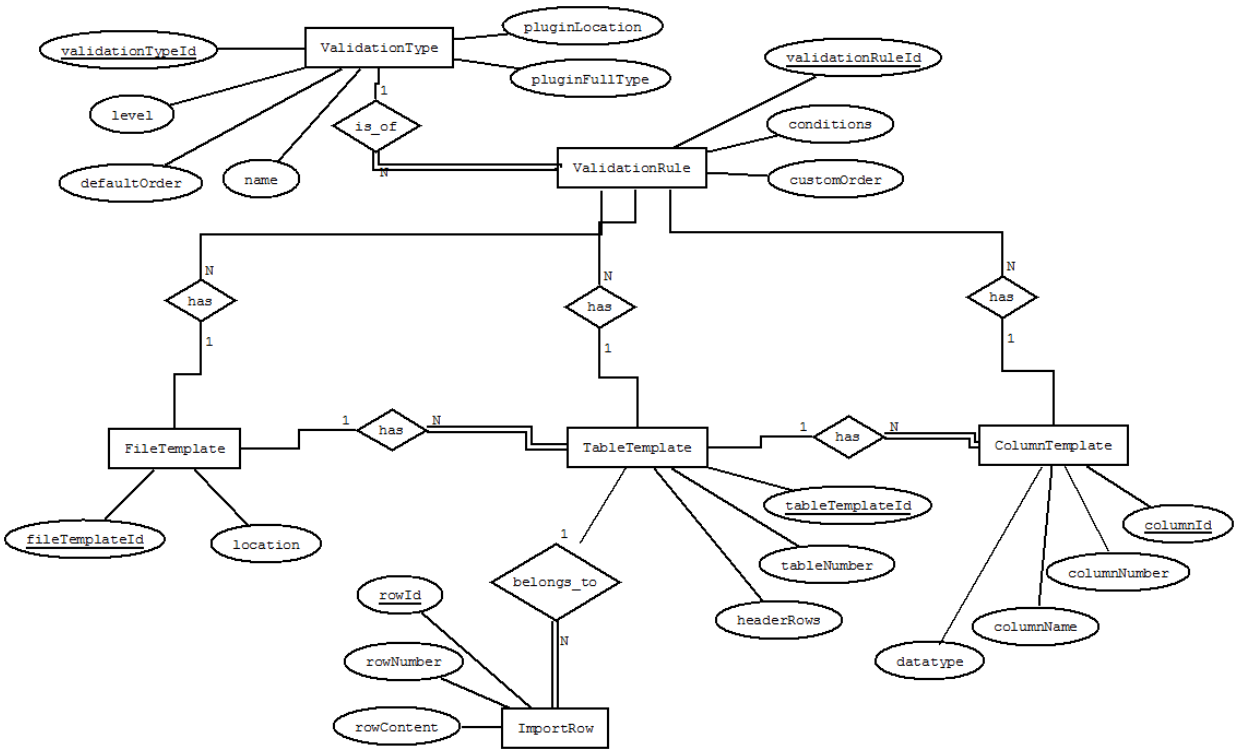
*Functionele eisen **F13 – F18***

Met behulp van de reeds bekende datastructuur van het conversiesysteem, de definities van validatieregels uit hoofdstuk 3 is de datastructuur te ontwerpen waarin de gewenste opslag van validatieregels en validatieresultaten kan worden verwerkt.

Deze database moet onderstaande gegevens kunnen vastleggen:

1. De aanwezige kolommen per tabel uit een aangeleverd klantbestand (onderdeel van een bestandstemplate en gekoppeld aan een tabel)
2. Een validatieregel met de volgende gegevens:
   - Naam en type van de validatieregel
   - Het niveau van de validatie (table, column, row of field)
   - Een koppeling naar een bestand, tabel, regel, kolom of veld (afhankelijk van het niveau) waar de regel op van toepassing is
   - De standaard en handmatige volgorde waarin de regel uitgevoerd moet worden (t.o.v. andere regels)
   - De specifieke condities waar datgene wat gevalideerd wordt aan moet voldoen

Voor de implementatie van de beschreven eisen is in figuur 8 een (E)ER diagram opgesteld. In dit diagram zijn drie nieuwe entiteittypen toegevoegd: *ColumnTemplate, ValidationRule, ValidationType.* Dit diagram is ook in het groot beschikbaar in bijlage K.1.



Figuur 7: ERD validatieregels

De entiteit *ColumnTemplate* is verantwoordelijk voor een beschrijving van een kolom die bij een bepaalde tabel hoort, inclusief kolomnummer, kolomnaam en datatype.

Een *ValidationType* is van een bepaald niveau en heeft een bepaalde naam. Per type kunnen er meerdere validatieregels worden toegevoegd (die allemaal van toepassing zijn op verschillende tabellen, kolommen enzovoorts en die verschillende condities kunnen bevatten).

4.1.1 Relatie(s) met bestandstemplate

Zoals te zien is aan de *partiële participatie* in de relaties tussen de validatieregel (*ValidationRule*) en de templates (*FileTemplate, TableTemplate en ColumnTemplate*) hoeft een template geen validatieregels te hebben en hoeft een validatieregel geen relatie te hebben met elke template. Op deze wijze worden de verschillende validatieniveaus geïmplementeerd:

- Table-level validatie → Relatie met TableTemplate
- Row-level validatie → Relatie met TableTemplate
- Column-level validatie → Relatie met ColumnTemplate
- Field-level validatie → Relatie met ColumnTemplate

4.1.2 Volgorde

De volgorde waarin een bepaalde regels worden uitgevoerd op een tabel, regel, kolom of veld is op twee manieren vast te leggen. In het entiteittype *ValidationType* is het attribuut *DefaultOrder* aanwezig waarin de standaard volgorde van validaties binnen een niveau is vastgelegd. Het is ook mogelijk om deze standaard volgorde niet te gebruiken maar een handmatige volgorde toe te passen. Hiervoor kan in het entiteittype *ValidationRule* het attribuut *CustomOrder* gebruikt worden.

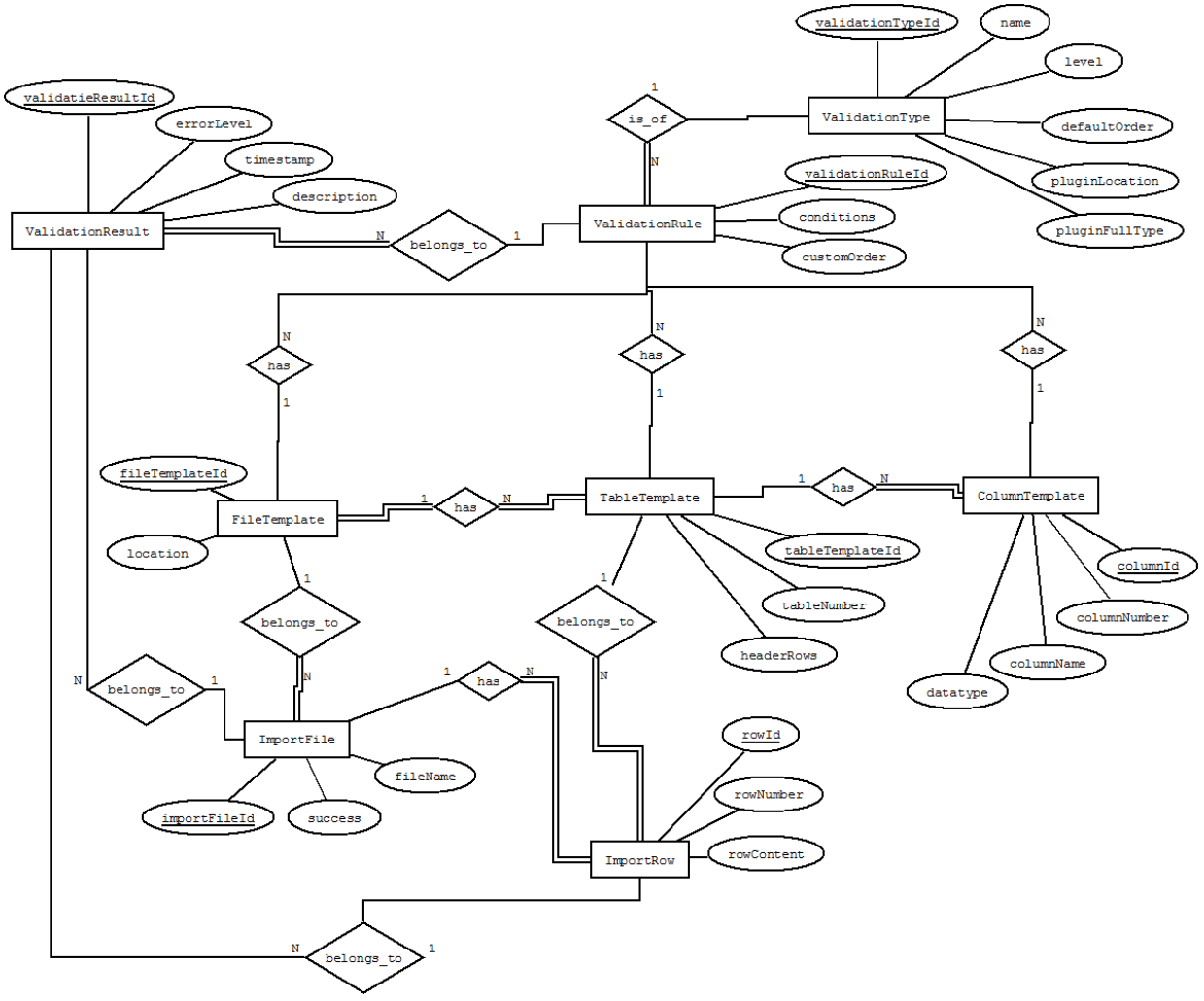
4.1.3 Condities

De condities waarin het gevalideerde moet voldoen wordt per validatieregel vastgelegd in de entiteit *ValidationRule*. De condities zullen worden opgeslagen in het attribuut *Conditions* wordt hiervoor gebruikt in de vorm van een CSV regel (“cond1”,”cond2”). Deze manier van data vastleggen gaat het (relationele) principe van databases tegen maar is een juiste keuze wanneer naar de implementatie van de programmatuur gekeken wordt omdat een plug-in systeem(zie hoofdstuk 5) gebruikt wordt. Ook zorgt deze oplossing dat de database niet nog complexer wordt.

4.2 Entity Relationship Diagram – Validatieresultaten

*Functionele eisen **F21, F23, F24***

Indien een validatie niet succesvol is dan moet dat resultaat worden vastgelegd in de database. Om dit te ondersteunen moet een extra entiteittype worden toegevoegd. In figuur 9 staat het (E)ER diagram waarin deze extra entiteit (*ValidationResult*) aanwezig is. Het ERD is groot bijgevoegd in bijlage K.2.



Figuur 8: ERD validatieresultaten

Het entiteittype *ValidationResult* is dus verantwoordelijk voor het vastleggen van een validatie die niet succesvol is afgerond. Elk resultaat is allereerst gekoppeld aan de validatieregel die de fout veroorzaakte en daarmee dus ook aan een template (*FileTemplate, TableTemplate of ColumnTemplate*).

Daarnaast zijn er relaties met een klantbestand (*ImportFile*) of een regel uit zo'n bestand (*ImportRow*) zodat de 'foutieve' waarde ook te herleiden is vanuit het resultaat.

Met deze structuur is het mogelijk om na de uitvoer van de validaties de resultaten op een juiste manier te structureren voor klanten omdat alle gegevens die daarvoor nodig zijn vanuit de *ValidationResult* template te bereiken zijn.

4.3 Relationeel Representatie Model

Dit hoofdstuk beschrijft de tabellen van de volledige database (voor het conversiesysteem en de validatie-engine) die kan worden afgeleid uit de EER diagrammen van de paragrafen 4.1-4.2 en uit bijlage A.

Dit model beschrijft de kolommen per tabel, de primaire sleutels en de vreemde sleutels die gebruikt zullen worden op de relaties tussen tabellen van het conversiesysteem en de validatie-engine.

Customer(CustomerId, Name, Info)

FileReader(FileReaderId, FileType, ReaderLocation, ReaderFullType)

ValidationType(ValidationTypeId, Name, ValidationLevel, DefaultOrder, Description, PluginLocation, PluginFullType)

FileTemplate(FileTemplateId, FileLocation, *CustomerId*, *FileReaderId*)
CustomerId is foreign key, references CustomerId in Customer, NOT NULL
FileReaderId is foreign key, references FileReaderId in FileReader, NOT NULL

TableTemplate(TableTemplateId, TableNumber, IgnoreRows, *FileTemplateId)*
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NOT NULL

ColumnTemplate(ColumnTemplateId, ColumnNumber, ColumnName, Datatype, *TableTemplateId*)
TableTemplateId is foreign key, references TableTemplateId in TableTemplate, NOT NULL

Import(ImportId, StartDate, EndDate, Success)

ImportFile(ImpFileId, FileName,Success, *ImportId, TemplatelFileId*)
ImportId is foreign key, references ImportId in Import, NOT NULL
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NOT NULL

ImportRow(ImpRowId, RowNumber, RowContent, *ImpFileId, TableTemplateId*)
ImpFileId is foreign key, references ImpFileId in ImportFile, NOT NULL
TableTemplateId is foreign key, references TableTemplateId in TableTemplate, NOT NULL

ValidationRule(ValidationRuleId, CustomOrder, Conditions, *ValidationTypeId*, *FileTemplateId*, *TableTemplateId*, *ColumnTemplateId*)
ValidationTypeId is foreign key, references ValidationTypeId in ValidationType, NOT NULL
FileTemplateId is foreign key, references FileTemplateId in FileTemplate, NULL ALLOWED
TableTemplateId is foreign key, references TableTemplateId in TableTemplate, NULL ALLOWED
ColumnTemplateId is foreign key, references ColumnTemplateId in ColumnTemplate, NULL ALLOWED

ValidationResult(ValidationResultId, Timestamp, ErrorLevel, Description, *ValidationRuleId*, *ImpFileId*, *ImpRowId*)
ValidationRuleId is foreign key, references ValidationRuleId in ValidationRule, NOT NULL
ImpFileId is foreign key, references ImpFileId in ImportFile, NULL ALLOWED
ImpRowId is foreign key, references ImpRowId in ImportRow, NULL ALLOWED

4.4 Relationeel Implementatie Model & Referentiële integriteit

In Bijlage K.3 staat het **R**elationeel **I**mplementatie **M**odel voor SQL Server 2008 dat is gebaseerd op het RRM van §4.3. In dat model zijn de ook datatypes van de kolommen vastgelegd.

Naast datatypen moet ook de referentiële integriteit worden vastgelegd voor elke vreemde sleutel relatie in de database. De volgende regels gelden:

1. Elke primaire sleutel is een surrogaat sleutel die automatisch wordt gegenereerd en opgehoogd door het SQL Server DBMS
2. ON UPDATE staat altijd op 'NO ACTION' (omdat de sleutels niet gewijzigd zullen worden, zie 1)
3. ON DELETE moet waar nodig op 'CASCADE' staan

In het RIM staan maar bij twee tabellen een ON DELETE definitie (ValidationRule en ValidationResult) omdat voor de andere tabellen de ON DELETE definitie niet geïmplementeerd kon worden. De reden hiervoor is dat SQL Server automatisch checkt op *'multiple cascade paths*[5]*'* die vanuit één DELETE query kunnen ontstaan en dit niet toestaat.

Om toch referentiële te verplichten zijn voor een aantal tabellen TRIGGERS geïmplementeerd waarin de DELETE query op een juiste wijze wordt uitgevoerd. De reeds geïmplementeerde TRIGGERS zijn ook geüpdate om met de nieuwe tabellen om te kunnen gaan. Zie Bijlage K.4.

4.5 Databasediagram

Aan de hand van het RRM en RIM is het mogelijk om een database te implementeren in Microsoft SQL Server 2008. Om een duidelijker beeld te geven van de database voor de Artena Data Validator is een databasediagram met behulp van Microsoft Visio opgesteld voor een grafische weergave van de in het RRM beschreven tabellen.

In dit diagram staat de gehele database van het conversiesysteem en de validatie-engine inclusief alle tabellen, kolommen, primaire sleutels en vreemde sleutels beschreven. Het diagram is bijgevoegd in bijlage K.5.

[5] http://www.christian-etter.de/?tag=multiple-cascade-paths

5. Programmatuur

Hoofdstuk 4 beschreef de datastructuren gebruikt om de benodigde gegevens voor conversie en validatie op te slaan. Wanneer gesproken wordt over een conversie dan wordt daarmee het uitlezen en eventueel converteren van klantbestanden bedoelt.

In dit hoofdstuk zal de programmatuur worden beschreven die in de validatie-engine aanwezig zal moeten zijn om de verschillende validatieregels op de verschillende niveaus uit te voeren.

Voor het opstellen van deze beschrijvingen zal gebruikt gemaakt worden van de requirements, de vorige hoofdstukken en de use cases(zie Analyserapport).

5.1 Plug-in systeem

Eisen: ***F21, NF5***

Voor de afhandeling van verschillende typen validatieregels in de validatie-engine zal een plug-in[6] systeem geïmplementeerd worden. Dit systeem wordt ook al toegepast in het conversiesysteem (ontwerprapport iteratie 1, §4.2). De validatie-engine zal voor elk type validatie een aparte plug-in zal inladen die een gegeven stuk data valideert met de gegeven condities. Er zal dus een plug-in zijn voor een datatype validatie en een plug-in voor een presence validatie enzovoorts.

De gegevens die het systeem nodig heeft om een plug-in in te laden zijn:

- Locatie van het DLL bestand (bijv. C:\Software\DatatypeValidator.dll)
- *Volledige type* van het in te laden object (bijv. Artena.Validator.DataTypeValidator)

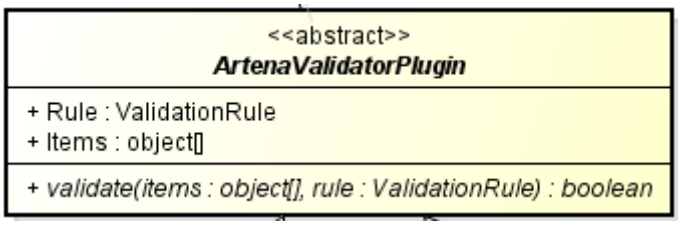
Deze gegevens zijn vastgelegd in twee tabellen in de database voor twee soorten plug-ins, één tabel voor de bestandslezer (filereader) van het conversiesysteem en één tabel voor de validatieregels van de validatie-engine:

- FileReader
- ValidationType

Met behulp van een dergelijk plug-in systeem wordt een grote mate van flexibiliteit gecreëerd omdat voor een extra type validatie alleen een stuk code(nieuwe plug-in) geschreven moet worden dat geen impact heeft op de code (de validatieroutine) van de validatie-engine zelf zal hebben.

Om te garanderen dat de validatie-engine elke plug-in op dezelfde wijze behandeld (en het aantal plug-ins dus uitbreidbaar is) zal de plug-in (of liever: de concrete klasse in de plug-in) overerven van een abstracte klasse. De abstractie klasse voor een validatie heet *ArtenaValidatorPlugin* is afgebeeld in figuur 9.

[6] http://en.wikipedia.org/wiki/Plug-in_(computing)



Figuur 9: Abstracte klasse validatie plug-in

Deze objectgeoriënteerde oplossing is dezelfde als die gebruikt wordt voor het conversiesysteem. De methode **validate()** moet per definitie geïmplementeerd worden door de overervende klassen (dit zijn dus de plug-ins voor de validatieregels) zodat de validatie-engine deze methode altijd kan aanroepen vanuit een validatieroutine.

5.2 LINQ-to-SQL & databasecommunicatie

De beschrijving van de O/R mapper in het ontwerprapport van het conversiesysteem (eerste iteratie) is wederom van toepassing op de validatie-engine. De drie nieuwe entiteiten (*ValidationRule, ValidationType en ValidationResult*) worden ook toegevoegd aan de LINQ-to-SQL implementatie.

De databasecontroller gebruikt in het conversiesysteem zal ook worden gebruikt door de validatie-engine om bijvoorbeeld de validatieregels voor bepaalde tabellen, rijen, kolommen of velden op te halen.

5.3 Klassendiagram

*Functionele eisen **F13-F25***

Het core gedeelte van de validatie-engine bestaat globaal gezien uit drie onderdelen:

1. Een controller die het gehele validatieproces (de validatieroutine) overziet
2. Het systeem dat de plug-ins van de validatieregels (op de juiste volgorde) inlaadt en de te controleren bestandsregels en condities meegeeft aan de plug-ins.
3. De validatie plug-in(s) die de bestandsregels op een juiste wijze valideren met de verkregen condities en een eventueel resultaat wegschrijven.

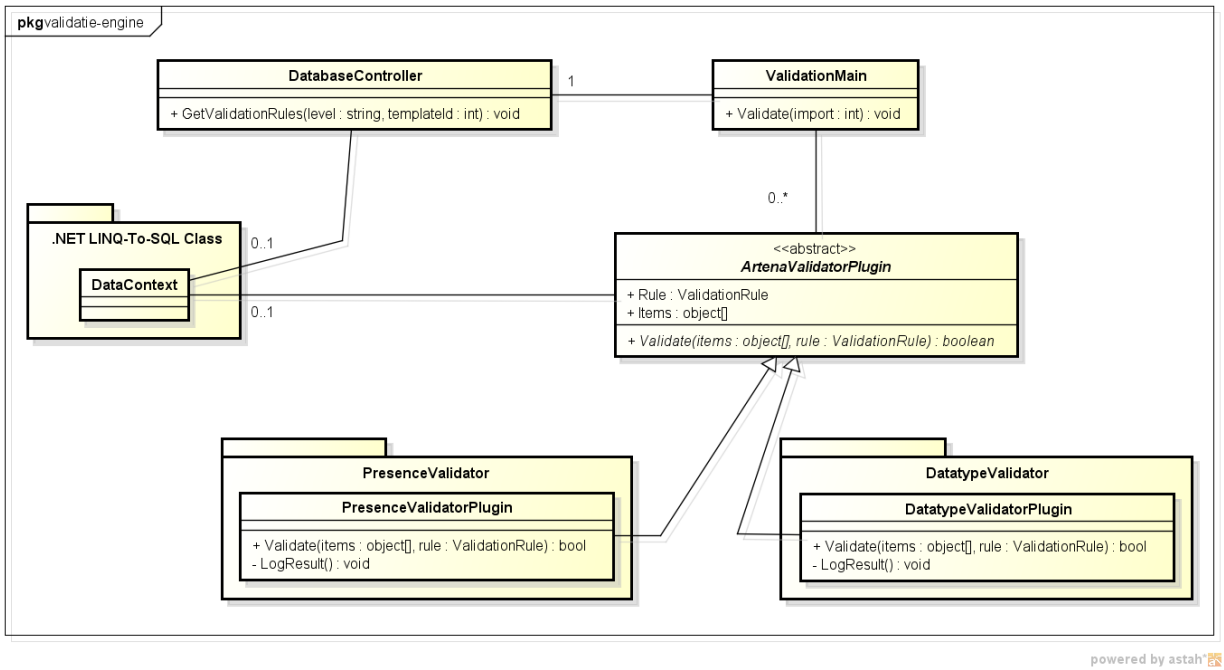
Naast deze onderdelen zal het object verantwoordelijk voor verbinding met de database (de DatabaseController) worden aangesproken om de in de requirements beschreven **invoer** te verkrijgen. De uitvoer (de validatieresultaten) wordt in de plug-ins zelf afgehandeld door gebruik te maken van de DataContext.

Om een validatieproces te starten zal de controller van de validatie-engine een verwijzing naar de import (die verwijst naar de, door het conversiesysteem, geïmporteerde bestanden en bestandsregels).

Met behulp van de kenmerken van de te valideren bestanden moet de engine de bijbehorende validatieregels inladen en vervolgens de regels op de juiste volgorde uitvoeren . Wanneer een validatie niet succesvol blijkt dan moet een resultaat worden weggeschreven naar de juiste tabel in de database.

Figuur 12 toont de klassen die gebruikt zullen worden voor de implementatie van de validatie-engine. In dit diagram worden ook twee validatie plug-ins getoond (voor datatype en presence). Andere

soorten validaties krijgen ook een dergelijke plug-in toegekend. De klasse *DatabaseController* is dezelfde klasse die gebruikt wordt door het conversiesysteem met een extra paar functionaliteiten benodigd voor de validatie-engine.



Figuur 10: structuur validatie-engine

5.3.1 Beschrijving klassen

ValidatorMain heeft controle over het validatie proces van begin tot eind. Dit object zal bestandsregels die gevalideerd moeten worden verkrijgen uit de database via de aangereikte verwijzing naar een import. Vervolgens zal het alle geregistreerde validatieregels voor de bestandsregels moeten ophalen. Met behulp van deze regels moeten de plug-ins verantwoordelijk voor de validaties worden ingeladen en moeten de validaties worden uitgevoerd.

DatabaseController is verantwoordelijk voor de connectie met de database en de uitvoer van alle benodigde query's voor de validatie-engine (en het conversiesysteem). Voor de connectie met de database maakt deze klasse gebruik van een DataContext klasse.

DataContext is een LINQ-To-SQL klasse waarmee de databasecontroller en validatie plug-ins op een gemakkelijke manier kunnen communiceren met de database door direct databaseobjecten aan te spreken (i.p.v. query's te bouwen)

ArtenaValidatorPlugin is de *abstracte klasse* waarin de een aantal noodzakelijke functionaliteiten van een validatieregel zijn vastgelegd, maar niet geïmplementeerd. Deze klasse is de basis waarvan elke plug-in (klasse) gebruik maakt. De functie **Validate()** is zoals eerder beschreven een verplichte implementatie voor een concrete klasse.

PresenceValidatorPlugin is de concrete implementatie van een plug-in waarmee een verplichte invulling van een veld gecontroleerd kan worden. Deze plug-in erft over van de ArtenaValidatorPlugin en wordt geïmplementeerd als een aparte Dynamic Linked Library (package).

DatatypeValidatorPlugin is de concrete implementatie van een plug-in waarmee de waarde in een veld vergeleken kan worden met een datatype. Deze plug-in erft over van de ArtenaValidatorPlugin en wordt geïmplementeerd als een aparte Dynamic Linked Library (package).

5.4 Het validatieproces

*Functionele eisen **F13, F14-F17, F23***

5.4.1 Het inladen van een (validatie) plug-in

Onderstaande opsomming toont de volgorde voor het inladen van een plug-in en welke tabellen uit de in hoofdstuk 4 beschreven database daarbij worden gebruikt. In onderstaand voorbeeld gaat de validatie-engine table-level validaties uitvoeren op een nieuwe import:

1. De validatie-engine haalt (via de *DatabaseController*) voor de eerste (en enige) tabel de een lijst met validatie(regel)s op uit de tabel **ValidationRule** die een verwijzing hebben naar de betreffende tabel(template).
2. De validatie-engine haalt voor de eerste regel uit de lijst met opgehaalde **ValidationRules** het **ValidationType** wat bij die regel hoort op (via de *DataContext*)
3. De validatie-engine laadt de plug-in in met behulp van de verkregen *PluginLocation* en *PluginFullType* uit het betreffende **ValidationType** record
4. De validatie-engine geeft de **ValidationRule** tezamen met de te controleren waarden mee aan de ingeladen plug-in zodat de plug-in de validatie kan uitvoeren. (m.b.v. van de **Validate()** methode).

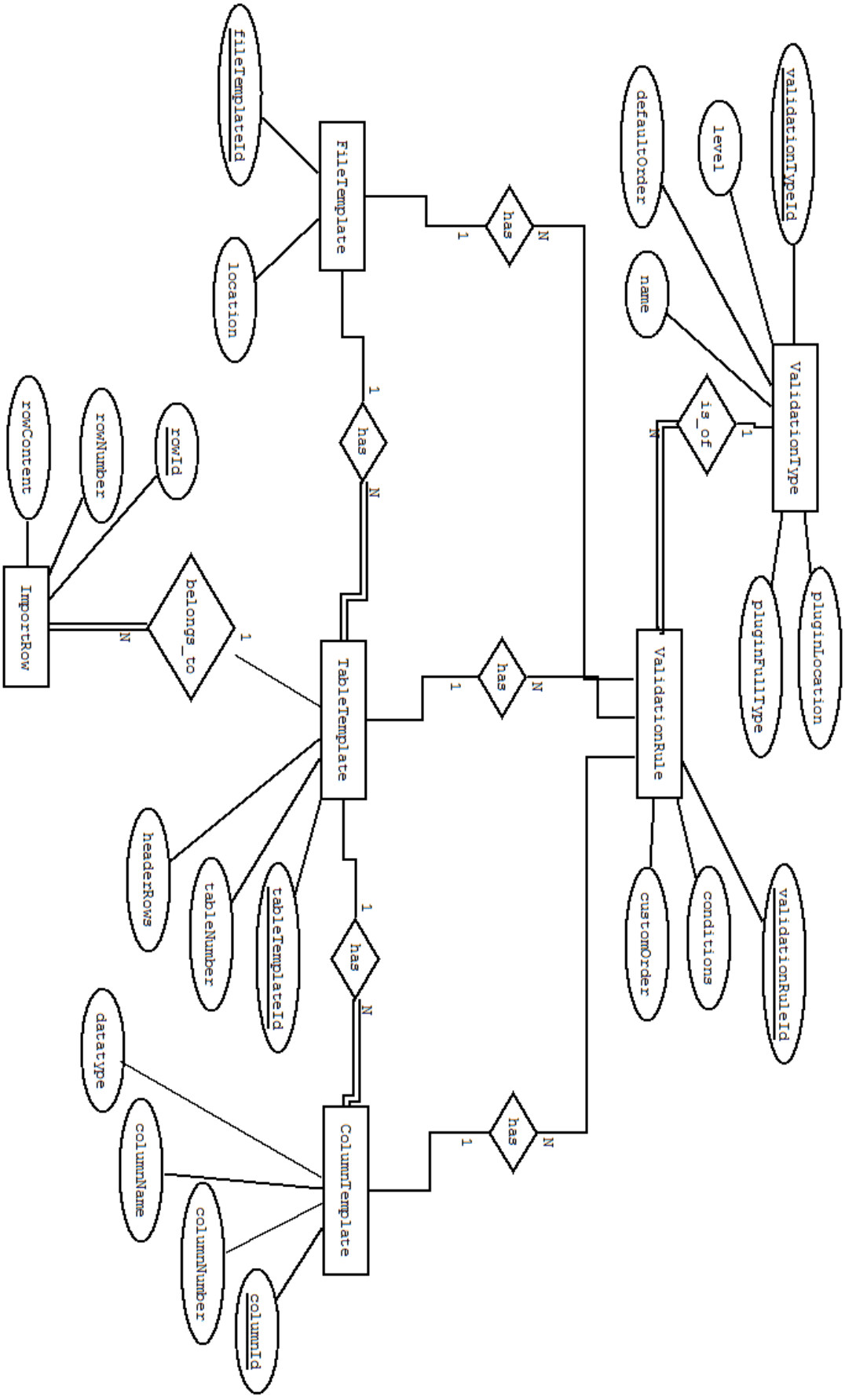
5.4.2 UML sequence diagram

Het validatieproces is al gedeeltelijk beschreven in de vorige paragraaf. Het validatieproces is ook beschreven in de use case *'Valideer import'* (zie analyserapport) en de volgorde waarin validaties uitgevoerd worden is ook uiteengezet in het hoofdstuk 3 van dit document. Het UML sequence diagram toont het proces nogmaals, maar dan overeenkomend met het klassendiagram. Zie bijlage K.6.

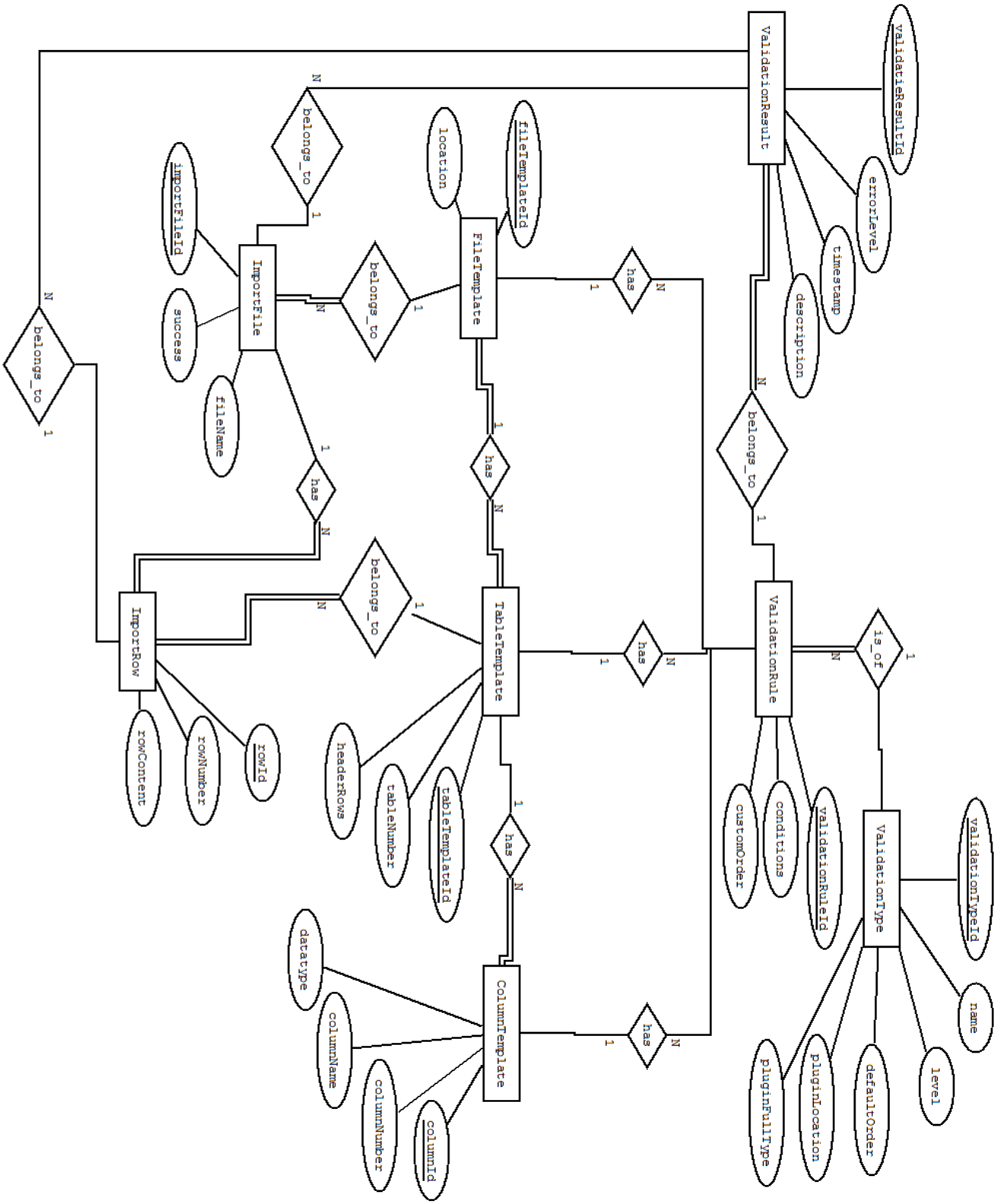
6. Begrippenlijst

- Abstract class
  - *Een klasse waarin minimaal één methode abstract is. Dit houdt in dat is aangeven hoe de methode eruit ziet, maar niet wat hij doet (de implementatie). Elke klasse die overerft van de abstracte klasse moet de abstracte methode altijd implementeren*
- DataContext
  - *De representatie van een LINQ-to-SQL class. Van dit object kunnen meerdere instanties aangemaakt worden zodat de applicatie met de betrokken database kan communiceren.*
- Multiple cascade paths
  - *Wanneer regel uit een tabel van een database via meer dan één cascading relatie gewijzigd of verwijdert kan worden. In dit geval bestaat de kans dat het DBMS in een lock terechtkomt. SQL Server 2008 ondersteunt deze functie daarom per definitie niet.*
- Partiële participatie
  - *Het feit dat in een relatie van een bepaald relatietype in een EER model elke entiteit van een bepaald entiteittype mag voorkomen (dus niet verplicht). Tegenovergesteld van totale participatie*.
- Polymorphisme
  - *Het feit dat een klasse van eenzelfde 'uiterlijk' of 'type' is als een andere klasse maar dat de onderliggende implementatie (en dus functionaliteit) kan verschillen.*

Bijlage K.1 - EER diagram validatieregels



Bijlage K.2 - EER Validatieresultaten



Bijlage K.3 - Relationeel Implementatie Model

```
CREATE TABLE [Import](
	[ImportId] [int] IDENTITY(1,1) NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[Success] [int] NOT NULL,
 CONSTRAINT [PK_import] PRIMARY KEY CLUSTERED
(
	[ImportId] ASC
))
```

```
CREATE TABLE [FileReader](
	[FileReaderId] [int] IDENTITY(1,1) NOT NULL,
	[FileType] [varchar](50) NULL,
	[ReaderLocation] [varchar](500) NULL,
	[ReaderFullType] [varchar](200) NULL,
 CONSTRAINT [PK_DataReader] PRIMARY KEY CLUSTERED
(
	[FileReaderId] ASC
))
```

```
CREATE TABLE [ValidationType](
	[ValidationTypeId] [int] IDENTITY(1,1) NOT NULL,
	[ValidationLevel] [varchar](50) NOT NULL,
	[DefaultOrder] [int] NULL,
	[Name] [varchar](50) NOT NULL,
	[Description] [varchar](1000) NULL,
	[PluginLocation] [varchar](500) NOT NULL,
	[PluginFullType] [varchar](200) NOT NULL,
 CONSTRAINT [PK_ValidationType] PRIMARY KEY CLUSTERED
(
	[ValidationTypeId] ASC
))
```

```
CREATE TABLE [ValidationRule](
	[ValidationRuleId] [int] IDENTITY(1,1) NOT NULL,
	[Conditions] [varchar](max) NULL,
	[CustomOrder] [int] NULL,
	[ValidationTypeId] [int] NOT NULL,
	[FileTemplateId] [int] NULL,
	[TableTemplateId] [int] NULL,
	[ColumnTemplateId] [int] NULL,
 CONSTRAINT [PK_ValidationRule] PRIMARY KEY CLUSTERED
(
	[ValidationRuleId] ASC
))
```

```
CREATE TABLE [ColumnTemplate](
	[ColumnTemplateId] [int] IDENTITY(1,1) NOT NULL,
	[ColumnNumber] [int] NOT NULL,
	[ColumnName] [varchar](50) NULL,
	[ColumnName] [varchar](50) NOT NULL,
	[TableTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_TemplateColumn] PRIMARY KEY CLUSTERED
```

```
(
	[ColumnTemplateId] ASC
))
```

```
CREATE TABLE [TableTemplate](
	[TableTemplateId] [int] IDENTITY(1,1) NOT NULL,
	[TableNumber] [int] NOT NULL,
	[HeaderRows] [int] NULL,
	[FileTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Template_Data_Definition] PRIMARY KEY CLUSTERED
(
	[TableTemplateId] ASC
))
```

```
CREATE TABLE [FileTemplate](
	[FileTemplateId] [int] IDENTITY(1,1) NOT NULL,
	[FileLocation] [varchar](500) NULL,
	[CustomerId] [int] NOT NULL,
	[FileReaderId] [int] NULL,
 CONSTRAINT [PK_TemplateFile] PRIMARY KEY CLUSTERED
(
	[FileTemplateId] ASC
))
```

```
CREATE TABLE [ImportFile](
	[ImpFileId] [int] IDENTITY(1,1) NOT NULL,
	[FileName] [varchar](500) NOT NULL,
	[Success] [int] NOT NULL,
	[ImportId] [int] NOT NULL,
	[FileTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Import_File] PRIMARY KEY CLUSTERED
(
	[ImpFileId] ASC
))
```

```
CREATE TABLE [ValidationResult](
	[ValidationResultId] [int] IDENTITY(1,1) NOT NULL,
	[Timestamp] [datetime] NOT NULL,
	[ErrorLevel] [varchar](50) NOT NULL,
	[Description] [varchar](max) NOT NULL,
	[ValidationRuleId] [int] NOT NULL,
	[ImpFileId] [int] NULL,
	[ImpRowId] [int] NULL,
 CONSTRAINT [PK_ValidationResult] PRIMARY KEY CLUSTERED
(
	[ValidationResultId] ASC
))
```

```
CREATE TABLE [ImportRow](
	[ImpRowId] [int] IDENTITY(1,1) NOT NULL,
	[RowNumber] [int] NOT NULL,
	[RowContent] [varchar](max) NULL,
	[ImpFileId] [int] NOT NULL,
	[TableTemplateId] [int] NOT NULL,
 CONSTRAINT [PK_Import_Content] PRIMARY KEY CLUSTERED
```

```
(
	[ImpRowId] ASC
))
```

```
CREATE TABLE [Customer](
	[CustomerId] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](200) NOT NULL,
	[Info] [varchar](1000) NULL,
 CONSTRAINT [PK_Client_Main_Template] PRIMARY KEY CLUSTERED
(
	[CustomerId] ASC
))
```

```
ALTER TABLE [ColumnTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateColumn_TemplateTable] FOREIGN KEY([TableTemplateId])
REFERENCES [TableTemplate] ([TableTemplateId])
ALTER TABLE [ColumnTemplate] CHECK CONSTRAINT
[FK_TemplateColumn_TemplateTable]

ALTER TABLE [FileTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateFile_FileReader] FOREIGN KEY([FileReaderId])
REFERENCES [FileReader] ([FileReaderId])
ALTER TABLE [FileTemplate] CHECK CONSTRAINT [FK_TemplateFile_FileReader]

ALTER TABLE [FileTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateFile_TemplateCustomer] FOREIGN KEY([CustomerId])
REFERENCES [Customer] ([CustomerId])
ALTER TABLE [FileTemplate] CHECK CONSTRAINT
[FK_TemplateFile_TemplateCustomer]

ALTER TABLE [ImportFile]  WITH CHECK ADD  CONSTRAINT [FK_ImportFile_Import]
FOREIGN KEY([ImportId])
REFERENCES [Import] ([ImportId])
ALTER TABLE [ImportFile] CHECK CONSTRAINT [FK_ImportFile_Import]

ALTER TABLE [ImportFile]  WITH CHECK ADD  CONSTRAINT
[FK_ImportFile_TemplateFile] FOREIGN KEY([FileTemplateId])
REFERENCES [FileTemplate] ([FileTemplateId])
ALTER TABLE [ImportFile] CHECK CONSTRAINT [FK_ImportFile_TemplateFile]

ALTER TABLE [ImportRow]  WITH CHECK ADD  CONSTRAINT
[FK_ImportRow_ImportFile] FOREIGN KEY([ImpFileId])
REFERENCES [ImportFile] ([ImpFileId])
ALTER TABLE [ImportRow] CHECK CONSTRAINT [FK_ImportRow_ImportFile]

ALTER TABLE [ImportRow]  WITH CHECK ADD  CONSTRAINT
[FK_ImportRow_TemplateTable] FOREIGN KEY([TableTemplateId])
REFERENCES [TableTemplate] ([TableTemplateId])
ALTER TABLE [ImportRow] CHECK CONSTRAINT [FK_ImportRow_TemplateTable]

ALTER TABLE [TableTemplate]  WITH CHECK ADD  CONSTRAINT
[FK_TemplateTable_TemplateFile] FOREIGN KEY([FileTemplateId])
REFERENCES [FileTemplate] ([FileTemplateId])
ALTER TABLE [TableTemplate] CHECK CONSTRAINT [FK_TemplateTable_TemplateFile]

ALTER TABLE [ValidationResult]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationResult_ImportFile] FOREIGN KEY([ImpFileId])
REFERENCES [ImportFile] ([ImpFileId])
```

```
ALTER TABLE [ValidationResult] CHECK CONSTRAINT
[FK_ValidationResult_ImportFile]

ALTER TABLE [ValidationResult]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationResult_ImportRow] FOREIGN KEY([ImpRowId])
REFERENCES [ImportRow] ([ImpRowId])
ALTER TABLE [ValidationResult] CHECK CONSTRAINT
[FK_ValidationResult_ImportRow]

ALTER TABLE [ValidationResult]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationResult_ValidationRule] FOREIGN KEY([ValidationRuleId])
REFERENCES [ValidationRule] ([ValidationRuleId])
ON DELETE CASCADE
ALTER TABLE [ValidationResult] CHECK CONSTRAINT
[FK_ValidationResult_ValidationRule]

ALTER TABLE [ValidationRule]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationRule_TemplateColumn] FOREIGN KEY([ColumnTemplateId])
REFERENCES [ColumnTemplate] ([ColumnTemplateId])
ALTER TABLE [ValidationRule] CHECK CONSTRAINT
[FK_ValidationRule_TemplateColumn]

ALTER TABLE [ValidationRule]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationRule_TemplateFile] FOREIGN KEY([FileTemplateId])
REFERENCES [FileTemplate] ([FileTemplateId])
ALTER TABLE [ValidationRule] CHECK CONSTRAINT
[FK_ValidationRule_TemplateFile]

ALTER TABLE [ValidationRule]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationRule_TemplateTable] FOREIGN KEY([TableTemplateId])
REFERENCES [TableTemplate] ([TableTemplateId])
ALTER TABLE [ValidationRule] CHECK CONSTRAINT
[FK_ValidationRule_TemplateTable]

ALTER TABLE [ValidationRule]  WITH CHECK ADD  CONSTRAINT
[FK_ValidationRule_ValidationType] FOREIGN KEY([ValidationTypeId])
REFERENCES [ValidationType] ([ValidationTypeId])
ON DELETE CASCADE
ALTER TABLE [ValidationRule] CHECK CONSTRAINT
[FK_ValidationRule_ValidationType]
```

Bijlage K.4 – Triggers

T1. Delete_Customer

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between Customer and FileTemplate
-- =============================================
ALTER TRIGGER [dbo].[Delete_Customer]
   ON  [ConversionImport].[dbo].[Customer]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      BEGIN
            -- Delete fileTemplate entries linked to the customer(enforces
CASCADE)
            DELETE FROM [ConversionImport].[dbo].[FileTemplate]
            FROM [ConversionImport].[dbo].[FileTemplate] AS ft
            INNER JOIN deleted d ON ft.CustomerId = d.CustomerId

            -- Delete the customer entry itself
            DELETE FROM [ConversionImport].[dbo].[Customer]
            WHERE CustomerId IN (SELECT d.CustomerId FROM deleted d)
      END
END
```

T2. Delete_File_Template

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between FileTemplate and TableTemplate & between
FileTemplate and ValidationRule
-- =============================================
ALTER TRIGGER [dbo].[Delete_File_Template]
   ON  [ConversionImport].[dbo].[FileTemplate]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      IF exists
      (
            SELECT * FROM [ConversionImport].[dbo].[ValidationRule] AS vr
```

```
            INNER JOIN deleted d ON vr.FileTemplateId = d.FileTemplateId
            WHERE vr.TableTemplateId is not null AND vr.ColumnTemplateId is
not null
      )
            BEGIN
                  RAISERROR ('One or more ValidationRule entries related to
the FileTemplate entry have a TableTemplateId or ColumnTemplateId which is
NOT NULL,
                                                aborting transaction to prevent
cyclic cascade', 1, 1)
                  ROLLBACK TRANSACTION
            END
      ELSE
            BEGIN

                  -- Delete importFile entries linked to the fileTemplate
 entry(enforces CASCADE 1)
                  DELETE FROM [ConversionImport].[dbo].[ImportFile]
                  FROM [ConversionImport].[dbo].[importFile] AS imf
                  INNER JOIN deleted d ON imf.FileTemplateId =
d.FileTemplateId

                  -- Delete tableTemplate entries linked to the fileTemplate
(enforces CASCADE 2)
                  DELETE FROM [ConversionImport].[dbo].[TableTemplate]
                  FROM [ConversionImport].[dbo].[TableTemplate] AS tt
                  INNER JOIN deleted d ON tt.FileTemplateId =
d.FileTemplateId

                  -- Delete validationRule entries linked to the fileTemplate
(enforces CASCADE 3)
                  DELETE FROM [ConversionImport].[dbo].[ValidationRule]
                  FROM [ConversionImport].[dbo].[ValidationRule] AS vr
                  INNER JOIN deleted d ON vr.FileTemplateId =
d.FileTemplateId

                  -- Delete the fileTemplate entry itself
                  DELETE FROM [ConversionImport].[dbo].[FileTemplate]
                  WHERE FileTemplateId IN (SELECT d.FileTemplateId FROM
deleted d)
            END
END
```

T3. Delete_Table_Template

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between TableTemplate and ColumnTemplate & TableTemplate and
ValidationRule
-- =============================================
ALTER TRIGGER [dbo].[Delete_Table_Template]
   ON  [ConversionImport].[dbo].[TableTemplate]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
```

```
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      IF exists
      (
            SELECT * FROM [ConversionImport].[dbo].[ValidationRule] AS vr
            INNER JOIN deleted d ON vr.TableTemplateId = d.TableTemplateId
            WHERE vr.FileTemplateId is not null AND vr.ColumnTemplateId is
not null
      )
            BEGIN
                  RAISERROR ('One or more ValidationRule entries related to
the TableTemplate entry have a FileTemplateId or ColumnTemplateId which is
NOT NULL,
                                          aborting transaction to prevent
cyclic cascade', 1, 1)
                  ROLLBACK TRANSACTION
            END
      ELSE
            BEGIN

                  -- Delete importRow entries linked to the tableTemplate
(enforces CASCADE 1)
                  DELETE FROM [ConversionImport].[dbo].[ImportRow]
                  FROM [ConversionImport].[dbo].[ImportRow] AS ir
                  INNER JOIN deleted d ON ir.TableTemplateId =
                  d.TableTemplateId

                  -- Delete columnTemplate entries linked to the
tableTemplate (enforces CASCADE 2)
                  DELETE FROM [ConversionImport].[dbo].[ColumnTemplate]
                  FROM [ConversionImport].[dbo].[ColumnTemplate] AS ct
                  INNER JOIN deleted d ON ct.TableTemplateId =
d.TableTemplateId

                  -- Delete validationRule entries linked to the
tableTemplate (enforces CASCADE 3)
                  DELETE FROM [ConversionImport].[dbo].[ValidationRule]
                  FROM [ConversionImport].[dbo].[ValidationRule] AS vr
                  INNER JOIN deleted d ON vr.TableTemplateId =
d.TableTemplateId

                  -- Delete the tableTemplate entry itself
                  DELETE FROM [ConversionImport].[dbo].[TableTemplate]
                  WHERE TableTemplateId IN (SELECT d.TableTemplateId FROM
deleted d)
            END
END
```

T4. Delete_Column_Template

```
-- ============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
```

```
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between ColumnTemplate and ValidationRule
-- ============================================

ALTER TRIGGER [dbo].[Delete_Column_Template]
   ON  [ConversionImport].[dbo].[ColumnTemplate]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      IF exists
      (
            SELECT * FROM [ConversionImport].[dbo].[ValidationRule] AS vr
            INNER JOIN deleted d ON vr.ColumnTemplateId = d.ColumnTemplateId
            WHERE vr.FileTemplateId is not null AND vr.TableTemplateID is not
null
      )
            BEGIN
                  RAISERROR ('One or more ValidationRule rows related to the
ColumnTemplate row has a FileTemplateId or TableTemplateId which is NOT
NULL,
                                          aborting transaction to prevent
cyclic cascade', 1, 1)
                  ROLLBACK TRANSACTION
            END
      ELSE
            BEGIN
                  -- Delete validationRules linked to the columnTemplate
                  DELETE FROM [ConversionImport].[dbo].[ValidationRule]
                  FROM [ConversionImport].[dbo].[ValidationRule] AS vr
                  INNER JOIN deleted d ON vr.ColumnTemplateId =
d.ColumnTemplateId

                  -- Delete the columnTemplate itself
                  DELETE FROM [ConversionImport].[dbo].[ColumnTemplate]
                  WHERE ColumnTemplateId IN (SELECT d.ColumnTemplateId FROM
deleted d)
            END
END
```

T5. Delete_Import

```
-- ============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between Import and ImportFile
ALTER TRIGGER [dbo].[Delete_Import]
   ON  [ConversionImport].[dbo].[Import]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
```

```
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      BEGIN
            -- Delete importFile entries linked to the import entry(enforces
CASCADE)
            DELETE FROM [ConversionImport].[dbo].[ImportFile]
            FROM [ConversionImport].[dbo].[importFile] AS imf
            INNER JOIN deleted d ON imf.ImportId = d.ImportId

            -- Delete the import entry itself
            DELETE FROM [ConversionImport].[dbo].[Import]
            WHERE ImportId IN (SELECT d.ImportId FROM deleted d)
      END
END
```

T6. Delete_Import_File

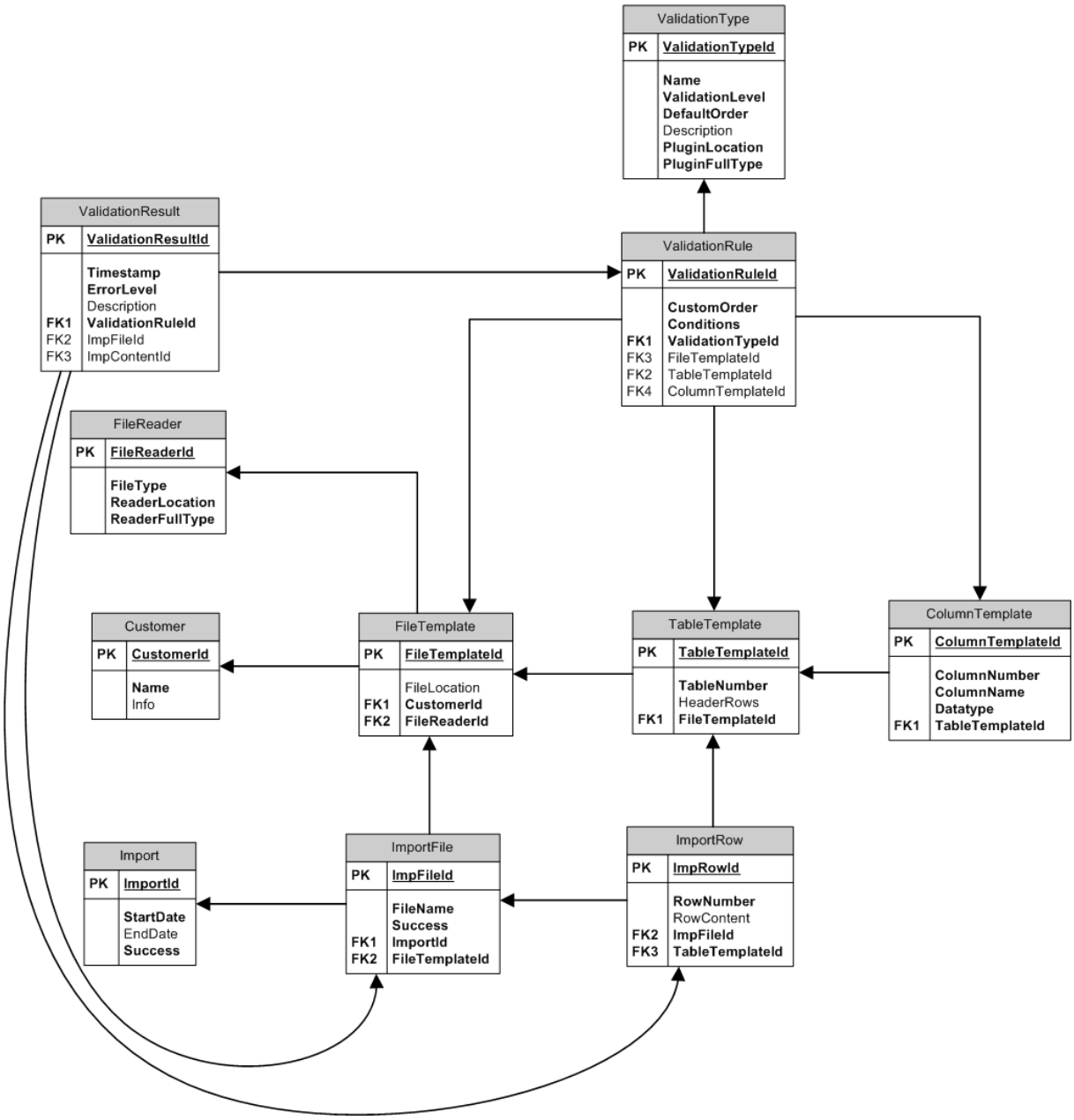
```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between ImportFile and ImportRow & between ImportFile and
ValidationResult
-- =============================================

ALTER TRIGGER [dbo].[Delete_Import_File]
   ON  [ConversionImport].[dbo].[ImportFile]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      IF exists
      (
            SELECT * FROM [ConversionImport].[dbo].[ValidationResult] AS vr
            INNER JOIN deleted d ON vr.ImpFileId = d.ImpFileId
            WHERE vr.ImpFileId is not null
      )
            BEGIN
                  RAISERROR ('One or more ValidationResult entries related to
the ImportFile entry have an ImpRowId which is NOT NULL,
                                          aborting transaction to prevent
cyclic cascade', 1, 1)
                  ROLLBACK TRANSACTION
            END
      ELSE
            BEGIN
                  -- Delete importRow entries linked to the
importFile(enforces first CASCADE)
                  DELETE FROM [ConversionImport].[dbo].[ImportRow]
                  FROM [ConversionImport].[dbo].[ImportRow] AS ir
```

```
                  INNER JOIN deleted d ON ir.ImpFileId = d.ImpFileId

                  -- Delete validationRule entries linked to the importFile
(enforces second CASCADE)
                  DELETE FROM [ConversionImport].[dbo].[ValidationResult]
                  FROM [ConversionImport].[dbo].[ValidationResult] AS vr
                  INNER JOIN deleted d ON vr.ImpFileId = d.ImpFileId

                  -- Delete the importFile entry itself
                  DELETE FROM [ConversionImport].[dbo].[ImportFile]
                  WHERE ImpFileId IN (SELECT d.ImpFileId FROM deleted d)
            END
END
```

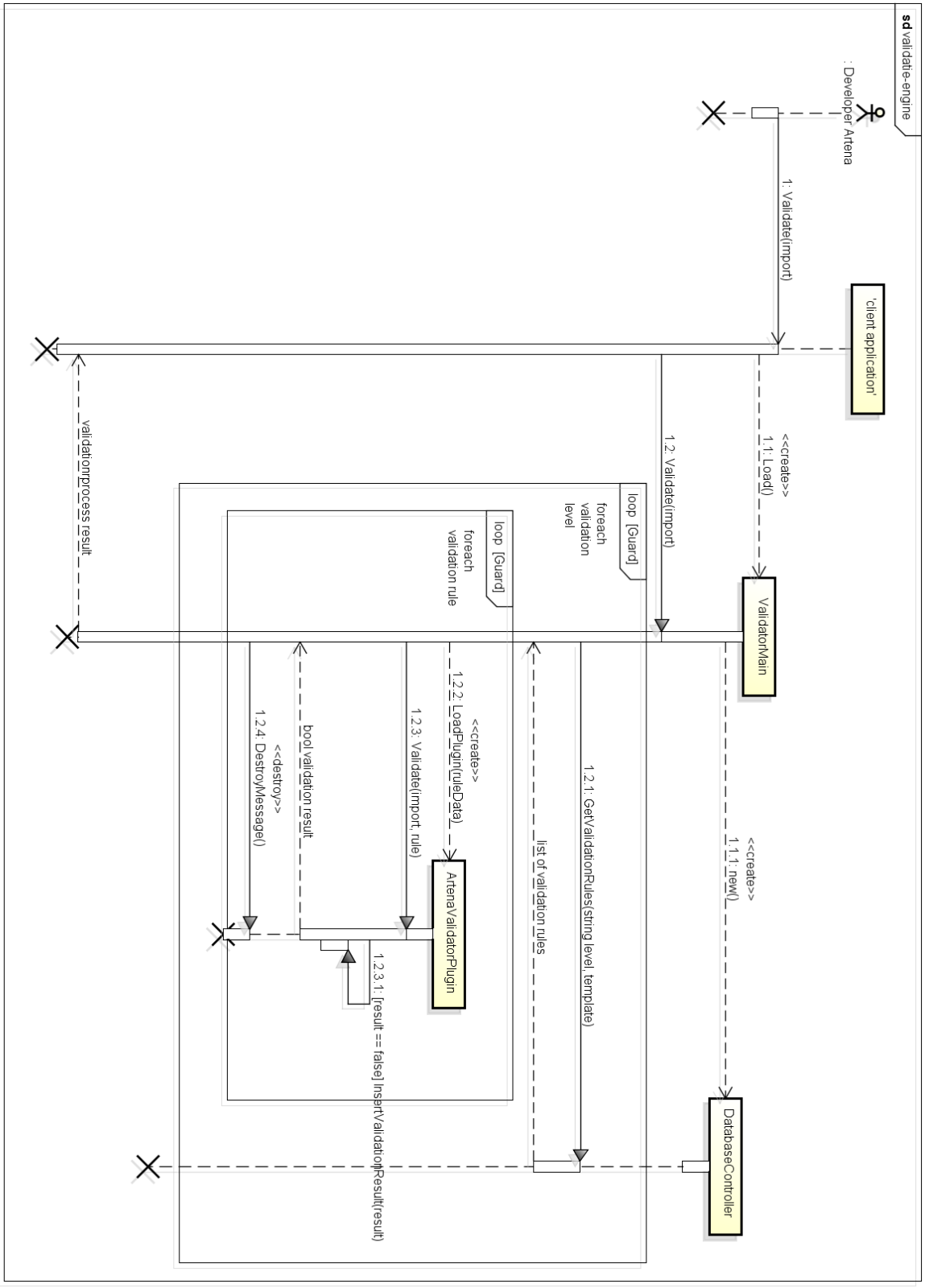
T7. Delete_Import_Row

```
-- =============================================
-- Author:        Axel Polet
-- Create date: 23-04-2012
-- Description:   This trigger is used to enforce referential integrity (ON
DELETE CASCADE) between ImportRow and ValidationResult
-- =============================================
ALTER TRIGGER [dbo].[Delete_Import_Row]
   ON  [ConversionImport].[dbo].[ImportRow]
   INSTEAD OF DELETE
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      -- SET NOCOUNT ON;

      IF @@ROWCOUNT = 0 RETURN

      IF exists
      (
            SELECT * FROM [ConversionImport].[dbo].[ValidationResult] AS vr
            INNER JOIN deleted d ON vr.ImpRowId = d.ImpRowId
            WHERE vr.ImpFileId is not null
      )
            BEGIN
                  RAISERROR ('One or more ValidationResult entries related to
the ImportRow entry have a ImpFileId which is NOT NULL,
                                          aborting transaction to prevent
cyclic cascade', 1, 1)
                  ROLLBACK TRANSACTION
            END
      ELSE
            BEGIN
                  -- Delete validationResults linked to the importRow
(enforces CASCADE)
                  DELETE FROM [ConversionImport].[dbo].[ValidationResult]
                  FROM [ConversionImport].[dbo].[ValidationResult] AS vr
                  INNER JOIN deleted d ON vr.ImpRowId = d.ImpRowId

                  -- Delete the importRow entry itself
                  DELETE FROM [ConversionImport].[dbo].[ImportRow]
                  WHERE ImpRowId IN (SELECT d.ImpRowId FROM deleted d)
            END
END
```

Bijlage K.5 – Databasediagram (visio)



Bijlage K.6 – UML sequence diagram validatieproces



Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage L

Analyserapport Iteratie 3: Artena Data Validator

Versie 1.0

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 13-04-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	10-04-2012	Concept	+ Inleiding + Eis m.b.t. conversie CSV bestanden (F26)
0.2	12-04-2012	Concept	+ Eisen logsysteem (F27, F28, NF6) + Eis m.b.t. Artena Data Validator als DLL (F9) + Use case diagram Artena Data Validator
1.0	13-04-2012	Eerste versie	+ Use case beschrijvingen

Gerelateerde documenten	Versie	Datum
Analyserapport Iteratie 1: Artena Data Validator	1.0	23-02-2012
Analyserapport Iteratie 2: Conversiesysteem	1.2	26-03-2012

Inhoudsopgave

1. Inleiding 4
2. Requirements 5
2.1 Functionele eisen 5
2.2 Niet-functionele eisen 9
3. UML Use cases 11
3.1.1 Upload nieuw(e) bestand(en) 11
3.1.2 Verwerk nieuw(e) bestand(en) 12
3.1.3 Conversiesysteem: Converteren klantbestand(en) 12
3.1.4 Conversiesysteem: Inladen bestandslezer 13
3.1.5 Validatie-engine: Valideer import 13
3.1.6 Validatie-engine: Ophalen bestandsregels 14
3.1.7 Validatie-engine: Inladen validatieregels 14
3.1.8 Validatie-engine: Vastleggen resultaten 15

1. Inleiding

Dit rapport is het derde en laatste analyserapport voor de Artena Data Validator. In de eerste en tweede iteraties zijn twee onderdelen van de Artena Data Validator afzonderlijk behandeld. In dit rapport zal een derde 'systeem' worden toegevoegd aan de analyse, het logsysteem. Daarnaast zal in dit rapport het samenkomen van de twee eerdere onderdelen (het conversiesysteem en de validatie-engine) behandeld worden.

De meeste requirements zijn al opgesteld in de eerste en tweede iteratie. In deze iteratie zijn extra requirements toegevoegd voor het logsysteem. De analyse door definitie van requirements is in deze iteratie dus veel kleiner geweest dan in voorgaande iteraties.

In dit document zullen eerst alle requirements zoals deze zijn opgesteld in samenwerking met de stakeholders uiteengezet worden (**hoofdstuk 2**). Tot slot wordt een nieuw UML use case diagram geïntroduceerd waarin de Artena Data Validator als één geheel wordt beschreven (**hoofdstuk 3**).

2. Requirements

De requirements in dit rapport kunnen worden beschouwd als de laatste en volledige versie van alle requirements voor de gehele Artena Data Validator. Het overgrote deel is afkomstig van de eerdere analyserapporten.

2.1 Functionele eisen

Per functionele eis in dit hoofdstuk is aangegeven van welke stakeholder deze eis afkomstig is. Daarnaast worden de functionele eisen beschreven in dit hoofdstuk worden ingedeeld naar prioriteit. Om de eisen naar prioriteit in te delen zal gebruik gemaakt worden van de MoSCoW methode. Met behulp van deze methode wordt het mogelijk om de eisen in te delen in de volgende categorieën:

1. **Must have:** deze eisen worden in ieder geval binnen de gemaakte timebox/planning verwerkt in het systeem
2. **Should have**: deze eisen zijn net zo belangrijk als de must have eisen. Echter is het voor deze eisen niet verplicht om dat binnen vastgelegde timebox te doen.
3. **Could have:** een dergelijke eis is gewenst maar niet noodzakelijk, deze eis kan meegenomen als tijd en kosten dat toelaten.
4. **Won't have**: deze eisen zullen niet worden verwerkt in het te implementeren systeem.

Bovenstaande categorieën zijn voorgelegd aan de stakeholders terwijl door de eisen is heen gelopen. Per eis is vervolgens een prioriteit toegekend.

De eisen hieronder zijn genummerd zodat er vanuit andere documenten (zoals het ontwerprapport) kan worden terugverwezen naar de eisen.

LEGENDA

Nr.	= nummer van de eis
Systeem	= subsysteem of deliverable waar de eis op van toepassing is
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Eis	= de eis zelf

Nr.	Systeem	St.	Pr.	Eis
F1	Conversie	S1, S2	1	Invoer van conversiesysteem: • Één of meerdere klantbestanden **{vervolg in F2, F3, F4}** • Één of meerdere verwijzingen naar de templates waarin de benodigde gegevens voor een conversie van een klantbestand staan **{vervolg in F5}**
F2	Conversie	S2	1	Soorten klantbestanden te ondersteunen door conversiesysteem: • Excel (xls of xlsx) • CSV
F3	Conversie	S3	3	Het conversiesysteem moet de conversie van XML klantbestanden uit kunnen voeren wanneer deze genoeg gegevens bevatten en complex genoeg zijn zodat kan worden voldaan aan de uitvoer van **{vervolg in F6}.**
F4	Conversie	S2	1	Structuur klantbestanden voor invoer conversiesysteem: • Een bestand bestaat uit één of meerdere tabellen

				○ Een tabel bestaat uit één of meerdere kopregels plus nul of meerdere rijen • Structuur rij in een tabel: ○ (xls/xlsx) nul of meerdere velden ○ (csv) nul of meerdere velden gescheiden door een scheidingsteken • Een veld heeft een datatype: ○ Datum of getal of double of string
F5	Conversie	S2	1	Gegevens benodigd in een template om conversie uit te voeren: • Aantal tabellen aanwezig in bestand (minimaal één) • Bestandstype (xls of xlsx of csv)
F6	Conversie	S2	1	Uitvoer van een conversieproces (de import): • Database met daarin: ○ Naam klantbestand ○ Nul of meerdere rijen uit klantbestand **{vervolg in F7}**
F7	Conversie	S2	1	Kenmerken van een rij in uitvoer van een conversie: • Nummer van de tabel • Nummer van de rij • Nul of meerdere velden gescheiden door een komma (volgens de RFC-4180 standaard[1])
F8	Conversie	S2	1	Formaten van uitvoervelden van xls/xlsx conversies moeten voldoen aan onderstaande definities, zodat dit gelijk zal zijn: 1. Datum – 'dd-mm-yyyy hh:mm:ss' 2. Integer – 'xxxxxxx'(geen duizendtal separatoren) 3. Double – 'xx.xx' (geen duizendtal separatoren & gescheiden door punt i.p.v. komma)
F9	Conversie	S2, S3	1	Het conversiesysteem moet in staat zijn conversies voor bestanden van verschillende klanten parallel aan elkaar uit te voeren.
F10	Conversie	S2	1	Indien zich fouten voordoen tijdens een conversieproces (=één run van een conversie) moet het conversiesysteem de gegevens/kenmerken van deze fout kunnen vastleggen/loggen. **{vervolg in F27}**
F11	Conversie	S2	1	Het conversiesysteem moet in staat zijn het bestandsformaat van het klantbestand automatisch te bepalen aan de hand van de inhoud van de meegeleverde template (of extensie) wanneer een klantbestand is geselecteerd.
F12	Conversie	S2	3	Het opstartmechanisme van het conversiesysteem moet gebaseerd zijn op een systeem dat een nieuwe aangeleverd klantbestand detecteert.
F13	Validatie	S2, S3	1	De invoer voor een uit te voeren validatie door de validatie engine bestaat uit onderstaande onderdelen: • Een deel van de uitvoer van een conversie (de import): ○ Nul of meerdere rijen met de structuur van **{F7}** • Een set met validatieregels op verschillende niveaus en van verschillende typen **{vervolg in F14, F15, F16 en F17}**

[1] http://tools.ietf.org/html/rfc4180

F14	Validatie	S2	1	Benodigde gegevens (naast condities) voor **table-level** (tabel niveau) validatie: • Tabelnummer Typen validaties op dit niveau: • Rows Range (aantal regels) o Condities: aantal regels in tabel ## is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####'
F15	Validatie	S2	1	Benodigde gegevens (naast condities) voor **row-level** validatie: • Tabelnummer • Rijnummer Typen validaties op dit niveau: • Field Count (aantal velden in rij) o Conditie: de rij heeft evenveel velden als er kolommen zijn vastgelegd in de template
F16	Validatie	S2, S4	1	Benodigde gegevens (naast condities) voor **column-level** validatie: • Tabelnummer • Kolomnummer Typen validaties op dit niveau: • Fields Range (aantal ingevulde velden): o Conditie: het aantal velden in kolom ## dat ingevuld is, is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####' • Uniqueness o Conditie: de waarde in kolom ## mag maar één keer voorkomen
F17	Validatie	S2, S3, S4	1	Benodigde gegevens (naast condities) voor **field-level** validatie: • Tabelnummer • Rijnummer • Kolomnummer Typen validaties op dit niveau: • Datatype o Conditie: waarde in veld is van het datatype 'datum' of 'double' of 'integer' of 'string' • Presence o Conditie: waarde in veld mag niet NULL zijn • Range o Conditie: waarde in veld is 'groter dan ####' of 'kleiner dan ####' of 'van #### tot ####' of 'percentage van ####' • Format o Conditie: waarde in moet van een bepaald formaat

				○ Mogelijke formaten: ▪ Datum – 'dd-mm-yyyy hh:mm:ss' ▪ Double – 'xx.xx' (gescheiden door punt i.p.v. komma) ▪ Integer – 'xxxxxxx' (geen duizendtal separatoren) • Lookup ○ Conditie: waarde in veld moet voldoen aan een waarde in een 'database—tabel—veld' • Relationship ○ Conditie: als 'waarde in veld 1 gelijk is aan #### dan waarde is waarde in veld 2 ingevuld'
F18	Validatie	S1, S2	1	Het systeem moet de mogelijkheid bevatten om *quality control validation* uit te voeren. Dit type validatie slaat op de wijze waarop een 'fout' resultaat wordt geïnterpreteerd. In dit geval geeft de fout een indicatie van de waarschijnlijkheid van de gevonden waarde. **{zie ook F23}**
F19	Validatie	S2	1	Indien een table-level validatie niet succesvol is, dat wil zeggen fout, dan zullen de row-level, column-level –en field-level validaties voor die tabel niet meer worden uitgevoerd.
F20	Validatie	S2	1	Indien een row-level –of column-level validatie niet succesvol is, dat wil zeggen fout, dan zullen field-level validaties voor die rij of kolom niet meer worden uitgevoerd.
F21	Validatie	S2	1	Het moet mogelijk zijn om naast de in **{F14, F15, F16 en F17}** beschreven validatieregels nieuwe soorten/typen validatieregels op elk gewenst niveau toe te voegen wanneer dat gewenst is **{zie ook NF5}**.
F22	Validatie	S2	1	De validatie-engine moet in staat zijn validaties van verschillende klantbestanden parallel aan elkaar uit te voeren.
F23	Validatie	S2	1	Een validatieresultaat wordt alleen vastgelegd als de validatie een fout genereerd. (en het gevalideerde dus niet voldoet aan de condities)
F24	Validatie	S2	2	Een validatieresultaat kan van twee niveaus zijn: • Error: er is niet voldaan aan de validatie condities • Warning: er is niet voldaan aan de validatie condities maar het gecontroleerde gegeven is niet perse fout (waarschijnlijkheid)
F25	Validatie	S1, S2	1	Gegevens van een fout resultaat van een validatie moeten voldoen aan onderstaande criteria: • De validatieregel (type en niveau) en condities waar niet aan is voldaan • De reden waarom er niet is voldaan aan de condities • Het tabelnummer waarop de validatie betrekking had ○ Table-level ○ Row-level ○ Column-level ○ Field-level • Het rijnummer waarop de validatie betrekking had ○ Row-level ○ Field-level

				• Het kolomnummer waarop de validatie betrekking had
				  ○ Column-level
				  ○ Field-level
				• Het resultaat moet in zijn geheel leesbaar zijn voor de gebruikers (de klanten)

F26	Conversie	S2, S3	2	Het moet mogelijk zijn meerdere CSV formaten te ondersteunen. Om dit af te handelen moet het systeem om kunnen gaan met meerdere *delimeters(, of ; of : of \t)* en *indicators(' of ")*.
F27	Log	S2	1	Fouten tijdens de uitvoer van een conversieproces of validatieproces moeten worden geregistreerd volgens onderstaande criteria: • Het deel van het systeem waarin de fout ontstond • Het proces waarin de fout ontstond • De kenmerken (details) van de fout
F28	Log	S2	1	Het systeem moet meerdere soorten log niveaus kunnen ondersteunen. De twee niveaus die in ieder geval ondersteund moeten worden: • Warning • Error
F29	Conversie ,Validatie, Log	S2, S3	1	De Artena Data Validator moet een op zich zelf staande softwarebibliotheek zijn die kan worden gebruikt door een willekeurig systeem via een interface.

2.2 Niet-functionele eisen

In deze paragraaf worden de niet-functionele eisen (ook wel kwaliteitseisen genoemd) beschreven. De niet-functionele eisen beschrijven vaak extra eisen aan de wijze waarop bepaalde functionaliteiten uitgevoerd worden door het systeem. Een complete lijst van soorten niet-functionele eisen is te vinden in de ISO-9126 standaard[2]

De categorieën gebruikt in de niet-functionele eisen:

1. Functionality compliance
2. Security
3. Changeability

LEGENDA

Nr.	= nummer van de eis
Systeem	= subsysteem of deliverable waar de eis op van toepassing is
St.	= stakeholder van wie de eis afkomstig is
Pr.	= prioriteit van de eis (MoSCoW)
Cat.	= categorie van de niet-functionele eis (ISO-9126)
Eis	= de eis zelf

Nr.	Systeem	St.	Pr.	Cat.	Eis
NF1	Conversie	S2	1	1	De uitvoer van een conversie voldoet zoals in **{F7}** beschreven aan de RFC-4180[3] standaard voor Comma Separated Values. Dit houdt in dat: • Individuele velden in een rij zijn omgeven door dubbele quotes (''item'')

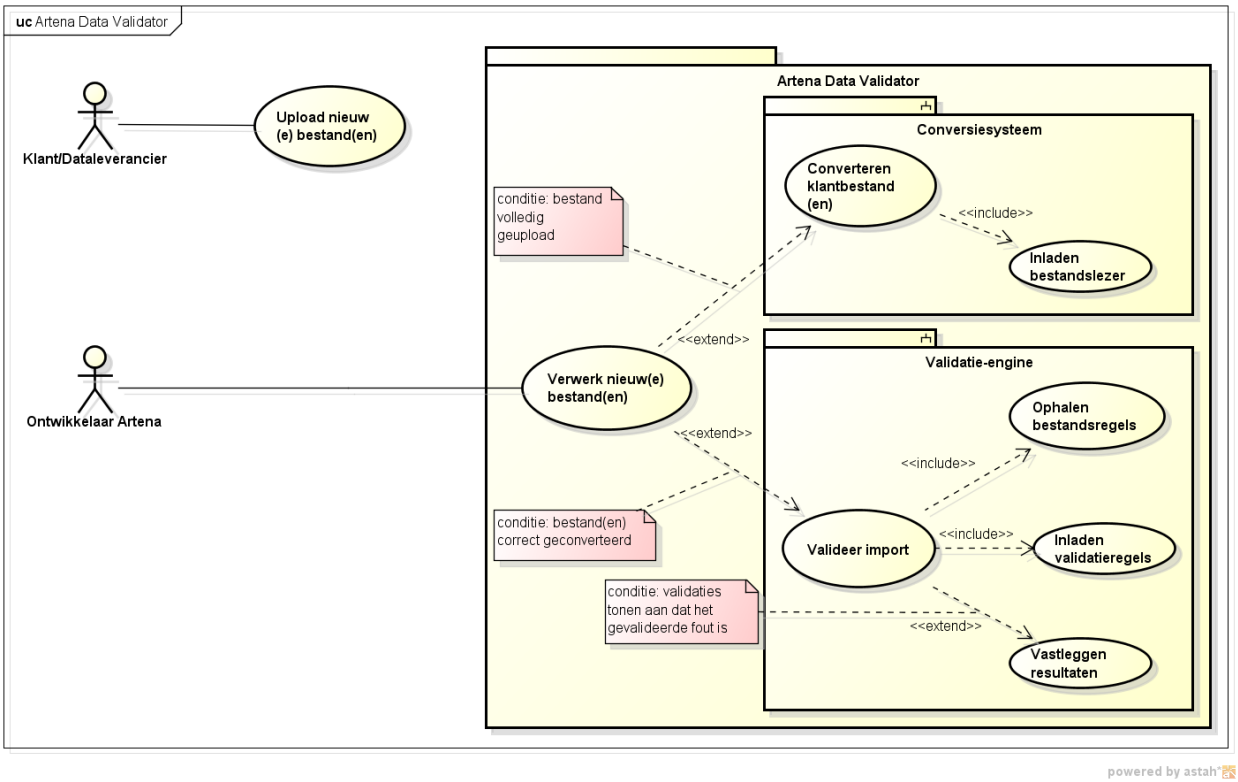
[2] http://www.testconsultancy-groep.nl/index.php/downloads - zie ISO-9126 toegelicht
[3] http://tools.ietf.org/html/rfc4180

					• Individuele velden zijn van elkaar gescheiden door een komma (,) • Het laatste veld wordt niet afgesloten met een komma
NF2	Conversie	S2	1	1	Het conversiesysteem moet de ASCII standaard[4] volledig ondersteunen. Dit houdt in dat het systeem de tekens uit deze karakterset kan herkennen en verwerken.
NF3	Conversie	S2	1	3	De flexibiliteit van het conversiesysteem moet zodanig zijn dat een ondersteuning van een nieuw bestandsformaat geen grote impact moet hebben op het conversieproces, de wijze waarop het conversieproces omgaat met verschillende soorten bestanden zal dus generiek moeten zijn.
NF4	Conversie & Validatie	S3	1	2	**Algemene security eis van Artena:** De gegevens van de uitvoer van een conversie moeten na validatie verwijderd kunnen worden (omdat deze gegevens meestal privacy gevoelige informatie bevatten)
NF5	Validatie	S2	1	3	De flexibiliteit van de validatie-engine moet zodanig zijn dat het toevoegen/wijzigen van de validatieregels en validatietypen geen grote impact heeft op het proces dat doorlopen wordt in de validatie-engine bij de uitvoer van een validatie
NF6	Log	S2	2	3	Het log systeem moet ook een mate van flexibiliteit bevatten waarbij een ondersteuning voor meerdere soorten loggers gewenst is.

[4] http://ascii-table.com/ansi-codes.php

3. UML Use cases

Dit hoofdstuk beschrijft het nieuwe use case diagram dat is gebaseerd op de samenkomst van de twee subsystemen uit de eerste twee iteraties. Het diagram beschrijft de interacties die zich kunnen voordoen in de Artena Data Validator.



Figuur 1: Use case diagram Artena Data Validator

Het diagram van figuur 1 beschrijft de interacties van de klant met het systeem en de interacties die ontwikkelaar van Artena met het systeem heeft wanneer de klant/dataleverancier één of meerdere nieuwe bestanden upload die gevalideerd moeten worden. De interacties zoals beschreven in het diagram zijn de acties en reacties die het systeem zal uitvoeren wanneer het in productie wordt genomen.

Zoals te zien is heeft de klant zelf weinig te maken het systeem; een volledige upload van het/de klantbestand(en) kan een ontwikkelaar ertoe zetten een nieuwe conversie en validatie te startten.

3.1.1 Upload nieuw(e) bestand(en)

Naam	Upload nieuw(e) bestand(en)
Aannamen	- De klant heeft de juiste autorisatie om het bestand te kunnen uploaden - De klant is in bezit van eventueel benodigde gebruikersnaam en wachtwoord
Beschrijving	(1) De klant logt op de (FTP) server of andere service **[I]** (2) De klant selecteert het te uploaden bestand

	(3) De klant start het uploaden van het bestand
Uitzondering	**[I]** FTP server niet bereikbaar of offline waardoor het bestand niet geüpload kan worden, de klant neemt dan waarschijnlijk contact op met Artena.
Resultaat	Het (deelnemer)bestand van de klant dat door Artena verwerkt moet worden is aanwezig op de juiste FTP locatie.

3.1.2 Verwerk nieuw(e) bestand(en)

Deze use case is de grootste toevoeging aan het use case diagram. Hierin komt namelijk de samenkomst van het conversiesysteem en de validatie-engine terug. Hoe het systeem met de conversie –en validatieprocessen om zal moeten gaan is hierin beschreven.

Naam	Verwerk nieuw(e) bestand(en)
Aannamen	- De gegevens van de betreffende klant zijn bekend bij de ontwikkelaar van Artena, zoals de bestandslocaties en de benodigde templates - Het/De nieuwe bestand(en) zijn bekend bij de ontwikkelaar van Artena
Beschrijving	(1) De ontwikkelaar voert de bestandsgegevens in en de identificaties van de templates en start de gehele import en validatie m.b.v. een button in de interface. (2) Het systeem vraagt de gegevens op van de betrokken klant inclusief de templates van bestanden die ondersteund worden voor die klant **[I][II]** (3) Het systeem bepaalt welke templates aan welke bestanden gekoppeld zijn en geeft verwijzingen naar deze templates samen met het/de klantbestand(en) door aan het conversiesysteem. (4) [Extension point: upload bestand volledig uitgevoerd] **Use case 'Converteren klantbestand(en)'** (5) Het systeem geeft een verwijzing naar de zojuist geconverteerde bestandsregels mee aan de validatie-engine. **[III]** (6) [Extension point: import klantbestand(en) correct uitgevoerd] **Use case 'Valideer import'**
Uitzondering	**[I]** Databasefout(*kan geen verbinding maken met database)*: het proces wordt afgebroken en een systeemfout wordt vastgelegd via het log systeem. **[II]** Het systeem herkent de ingevoerde template niet, waardoor het proces wordt afgebroken en een systeemfout wordt vastgelegd via het log systeem. **[III]** De conversie van de aangeleverde data is mislukt door een willekeurig voorgekomen systeemfout, waardoor het validatieproces niet opgestart zal worden.
Resultaat	Het/de aangeleverde bestand(en) is/zijn geconverteerd en gevalideerd en de resultaten van deze processen kunnen eventueel worden verwerkt tot een rapportage voor de klant (dit valt buiten de scope van het project).

3.1.3 Conversiesysteem: Converteren klantbestand(en)

De use case is van toepassing op vrijwel alle eisen die van toepassing zijn op het conversiesysteem en omvat dan ook het gehele conversieproces.

Naam	Converteren klantbestand(en) (functionele eisen **F1 – F12)**
Aannamen	- De aangeleverde locaties van de klantbestanden zijn beschikbaar - De aangeleverde templates zijn correct
Beschrijving	(1) Het conversiesysteem selecteert de juiste bestandslezer per bestand **Use case 'Inladen bestandslezer'** (2) De bestandslezer leest de regels van het bestand in en voert afhankelijk van het bestandsformaat enkele manipulaties uit. **[I]** (3) Het conversiesysteem schrijft de geconverteerde bestandsregels weg naar de database is het gewenste formaat **[II]** (4) Het conversiesysteem retourneert het resultaat van de conversie naar het systeem dat de conversiedata heeft aangeleverd
Uitzondering	**[I]** Bestandslezer kan het bestand niet inlezen, waardoor het conversieproces wordt afgebroken en een systeemfout wordt vastgelegd via het logsysteem **[II]** Databasefout(*kan geen verbinding maken met database)*, waardoor het conversieproces wordt afgebroken en een systeemfout wordt vastgelegd via het logsysteem
Resultaat	De data aanwezig in het geselecteerde bestand is geconverteerd en weggeschreven naar de betreffende database. Daarbij is ook het resultaat van het gehele proces geretourneerd naar het bovenliggende systeem.

3.1.4 Conversiesysteem: Inladen bestandslezer

Naam	Inladen bestandslezer (functionele eis **F2,** niet-functionele eis **NF5)**
Aannamen	Alle gegevens die benodigd zijn om een correcte bestandslezer te vinden en in te laden zijn bekend
Beschrijving	(1) Het systeem zoekt de locatie van de bestandslezer op **[I]** (2) Het systeem laadt de bestandslezer in het conversiesysteem en zorgt ervoor dat er mee gecommuniceerd kan worden
Uitzondering	**[I]** De locatie van de bestandslezer is niet beschikbaar/bereikbaar, waardoor een het conversieproces afgebroken zal worden en een systeemfout zal worden vastgelegd via het log systeem.
Resultaat	Een nieuw ingeladen bestandslezer die gebruikt kan worden voor het conversiesysteem

3.1.5 Validatie-engine: Valideer import

De eerste use case is van toepassing op alle eisen gesteld voor de validatie-engine. Deze use case omvat dan ook het gehele validatieproces waaronder de uitvoer van validaties op de verschillende validatieniveaus.

Naam	Valideer import (functionele eisen **F13 – F25)**
Aannamen	- De aangeleverde verwijzing naar de nieuwe import (dus de nieuw geconverteerde bestandsregels van de aangeleverde bestanden) is correct - De import is beschikbaar in de database van de Artena Data Validator

Beschrijving	(1) **Use case 'ophalen bestandsregels'** (2) **Use case 'inladen validatieregels'** (3) De validatie-engine start de uit te voeren validaties op table-level (4) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (5) **Use case 'inladen validatieregels'** (6) De validatie-engine start de uit te voeren validaties op row-level **[I]** (7) **Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (8) **Use case 'inladen validatieregels'** (9) De validatie-engine start de uit te voeren validaties op column-level **[I]** (10)**Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (11)**Use case 'inladen validatieregels'** (12)De validatie-engine start de uit te voeren validaties op field-level **[I] [II]** (13)**Use case 'vastleggen resultaten' [Extension point: fout na validatie]** (14)De validatie-engine retourneert het resultaat van het validatieproces naar het bovenliggende systeem dat de importdata heeft aangeleverd.
Uitzondering	**[I]** Validatie op table-level is niet succesvol, waardoor de validatie-engine de validatieregels op de andere validatieniveaus niet zal uitvoeren en het validatieproces zal afronden. **[II]** Validaties uitgevoerd op bijbehorend column –of row-level zijn niet succesvol, waardoor de validatie-engine de validatieregels op het betreffende veld niet uit zal voeren.
Resultaat	De bestandsregels uit van de import zijn gevalideerd en eventuele resultaten van de validaties zijn vastgelegd in een database. Het resultaat van het proces is geretourneerd naar het bovenliggende systeem.

3.1.6 Validatie-engine: Ophalen bestandsregels

Naam	Ophalen bestandsregels (functionele eis **F13)**
Aannamen	- De database waarin de bestandsregels aanwezig in zijn is beschikbaar en bereikbaar
Beschrijving	(1) Het systeem haalt de bestandsregels op met behulp van verkregen verwijzing naar de import (2) Het systeem retourneert de opgehaalde gegevens
Uitzondering	
Resultaat	De gegevens zijn opgehaald uit de database en geretourneerd

3.1.7 Validatie-engine: Inladen validatieregels

Naam	Inladen validatieregels (functionele eis **F13)**
Aannamen	- De validatieregels zijn beschikbaar en bereikbaar in de database
Beschrijving	(1) Op basis van de het validatieniveau (table, row, column of field) en de identificatie van tabellen, regels, kolommen en rijen worden de juiste validatieregels opgehaald (2) Het systeem zet de validatieregels om in validatie objecten* zodat de validatie-engine de regels kan uitvoeren

	(3) Het systeem retourneert de validatie objecten naar de validatie-engine
Uitzondering	
Resultaat	Validatieregels zijn ingeladen en bruikbaar voor de validatie-engine

* hetzelfde plug-in systeem als gebruikt voor het conversiesysteem wordt toegepast

3.1.8 Validatie-engine: Vastleggen resultaten

Naam	Vastleggen resultaten (functionele eisen **F23 – F25)**
Aannamen	- De database waarin de resultaten vastgelegd moeten worden is beschikbaar en bereikbaar
Beschrijving	(1) Het systeem vraagt aan de gebruikte validatie objecten om de resultaten van de validatie op de juiste wijze weg te schrijven (volgens de eisen) (2) De resultaten worden samengesteld weg geschreven naar de database
Uitzondering	
Resultaat	De resultaten van de validaties zijn vastgelegd in een database

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage M

Ontwerprapport Iteratie 3: Artena Data Validator

Versie 1.1

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 18-04-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
0.1	13-04-2012	Concept	Eerste versie ontwerprapport basis van de ontwerprapporten van de eerste twee iteraties en feedback verkregen vanuit vergaderingen en gesprekken met stakeholders + *SystemLog* tabel ontwerp
1.0	16-04-2012	Eerste versie	+ Technisch Ontwerp Artena Data Validator + Facade class + Klassendiagrammen subsystemen + Logging systeem inclusief Singleton en Observer
1.1	18-04-2012	Tweede versie	+ Belangrijke methoden in klassendiagram - Verwijdering *NewValidation* struct uit klassendiagram + beschrijving configuratiebestand en CsvFileReader + beschrijving importproces + beschrijving beheer

Gerelateerde documenten	Versie	Datum
Analyserapport Iteratie 3: Artena Data Validator	1.0	13-04-2012
Ontwerprapport Iteratie 1: Conversiesysteem	1.2	02-03-2012
Ontwerprapport Iteratie 2: Validatie-engine	1.0	03-04-2012

Inhoudsopgave

1. Inleiding 4
2. Databaseontwerp 5
2.1 (Enhanced) Entity Relationship Diagram – Log systeem 5
2.2 Relationeel Representatie Model 5
2.3 Relationeel Implementatie Model 5
2.4 Databasediagram 6
3. Programmatuur 7
3.1 Dynamic Linked Library 7
3.2 Entry point Artena Data Validator 7
3.3 Subsysteem 1: Conversiesysteem 9
3.3.1 Klassendiagram 9
3.3.2 Beschrijving klassen 10
3.4 Subsysteem 2: Validatie-engine 10
3.4.1 Klassendiagram 10
3.4.2 Beschrijving klassen 11
3.5 Log systeem 12
3.5.1 Beschikbaarheid logger 12
3.5.2 Soorten loggers 12
3.5.3 Implementatie (Singleton) 12
3.5.4 Observer pattern 13
3.6 Configuratie CSV File Reader 14
3.7 Het gehele importproces 14
4. Begrippenlijst 16

1. Inleiding

In dit document is het ontwerp van de Artena Data Validator vastgelegd. Dit ontwerp is ontstaan uit het samenkomen van het conversiesysteem en de validatie-engine zoals ook beschreven in het analyserapport(§3.1, use cases).

Het ontwerp in document beschrijft de Artena Data Validator als één systeem en het zal ook op die geïmplementeerd worden. De uiteindelijke implementatie zal één DLL (dynamic linked library) zijn die door een willekeurig systeem (zoals een gebruikersinterface) in te laden is. De interface (= entry-point) van de Artena Data Validator zal zo simpel mogelijk worden gehouden, zodat een dergelijke gebruikersinterface weinig hoeft af te weten van de processen die binnen de DLL uitgevoerd worden.

Naast het samenkomen van het conversiesysteem en validatie-engine zal het ontwerp worden uitgebreid met een (flexibel) logsysteem. Dit systeem dient als registratiemiddel om systeemfouten (die zich kunnen voordoen tijdens conversie –of validatieprocessen) vast te leggen in de database of ander log bestand. Het ontwerp bestaat uit een toevoeging aan het bestaande databaseontwerp en een extra klassendiagram waarin twee design patterns verwerkt zijn.

In **hoofdstuk 2** is de database-uitbreiding, voor het logsysteem, beschreven.

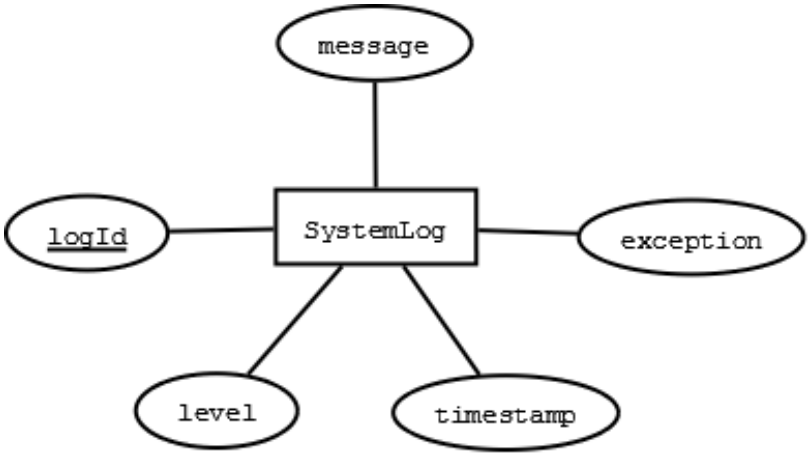
In **hoofdstuk 3** wordt de programmatuur van de Artena Data Validator in zijn geheel beschreven. In dit hoofdstuk is duidelijk te zien hoe de twee subsystemen (conversie en validatie) zijn samengekomen in de Artena Data Validator. In dit hoofdstuk zijn extra UML klassendiagrammen beschreven voor de Artena Data Validator en het logsysteem (en zijn design patterns). Tot slot wordt in het hoofdstuk een beschrijving gegeven van het gehele importproces.

2. Databaseontwerp

In dit hoofdstuk wordt een uitbreiding op de bestaande database ontworpen. Het ontwerp voor het log systeem bestaat uit een ERD, Relationeel Representatie Model en RIM.

2.1 (Enhanced) Entity Relationship Diagram – Log systeem

Naast validatieresultaten moet de Artena Data Validator ook de mogelijkheid hebben om fouten die zich voordoen tijdens de uitvoer van conversie –en/of validatieprocessen vast te leggen (te loggen). Om het vastleggen van deze fouten in een database te realiseren moet een entiteittype worden toegevoegd aan het bestaande ontwerp. Zie figuur 1.



Figuur 1: ERD log systeem

Het entiteittype *SystemLog* kan een fout zo duidelijk mogelijk specificeren. Als zich een exceptie (zie hoofdstuk 5) heeft voorgedaan waardoor de systeemfout is ontstaan, kunnen de kenmerken van deze exceptie worden vastgelegd in *exception*.

Daarnaast heeft de entiteit de mogelijkheid om een log bericht vast te leggen in *message*, de zwaarte van de voorgekomen fout vast te leggen in *level* en de precieze datum en tijd in *timestamp*.

2.2 Relationeel Representatie Model

SystemLog(LogId, Timestamp, Level, Message, Exception)

De tabel SystemLog heeft geen relatie met een andere tabel uit de database van de Artena Data Validator zodat vanuit elk onderdeel in het systeem een bericht weggeschreven kan worden zonder dat bepaalde informatie nodig is. Meer hierover in hoofdstuk 3.

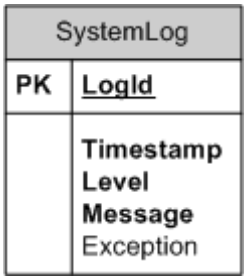
2.3 Relationeel Implementatie Model

Het onderstaande RIM toont ook de databasetypen en vereiste velden (NULL/NOT NULL). *LogId* is een identity wat betekent dat de waarde automatisch wordt toegekend en verhoogt.

```
CREATE TABLE [SystemLog](
      [LogId] [int] IDENTITY(1,1) NOT NULL,
      [Timestamp] [datetime] NOT NULL,
```

```
      [Level] [varchar](100) NULL,
      [Message] [varchar](1000) NULL,
      [Exception] [varchar](max) NULL,
 CONSTRAINT [PK_ConversionLog] PRIMARY KEY CLUSTERED
(
      [LogId] ASC
))
```

2.4 Databasediagram



Figuur 2: Visio2010 diagram SystemLog

3. Programmatuur

In dit hoofdstuk wordt de volledige programmatuur van de Artena Data Validator uiteengezet. Hierbij wordt gebruik gemaakt van de eerdere ontwerpen van de subsystemen voor de Artena Data Validator. Deze ontwerpen zijn beschreven in de ontwerprapporten van iteratie 1 en iteratie 2.

3.1 Dynamic Linked Library

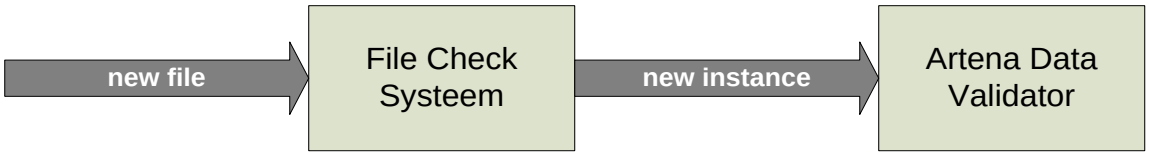
*Functionele eis **F9, F22, F29***

De Artena Data Validator geïmplementeerd moeten worden als softwarebibliotheek zodat het gemakkelijk implementeerbaar is door andere systemen. Een bijkomend voordeel is dat het systeem op deze manier gemakkelijk overdraagbaar is.

Een softwarebibliotheek zal worden geïmplementeerd als een Dynamic Linked Library (DLL) en zal dus dynamisch ingeladen kunnen worden een client (bijvoorbeeld een gebruikersinterface) die de Artena Data Validator zal gebruiken.

In de DLL zal er één toegangspunt zijn dat de gehele applicatie controleert, zodat de client van buitenaf maar enkele methoden hoeft te gebruiken en geen kennis hoeft te hebben van het onderliggende systeem en de processen die zich daar afspelen. Om dit te implementeren zal gebruik worden gemaakt van een Facade class[1]. De client zal een instantie aan moeten maken van deze class om de functionaliteiten van het systeem te gebruiken (zie §3.2).

Onderstaand figuur is afgeleid uit 'ontwerprapport iteratie 1, §4.1' en het gebruik van de Artena Data Validator door een *file check systeem* (de client). Door dit principe is het mogelijk om meerdere instanties aan te maken van de Artena Data Validator waardoor conversies en validaties parallel uitgevoerd kunnen worden (voor meerdere imports tegelijkertijd).



Figuur 3: Opstart Artena Data Validator (voorbeeld)

3.2 Entry point Artena Data Validator

De globale indeling van de Artena Data Validator inclusief zijn 'entry point':

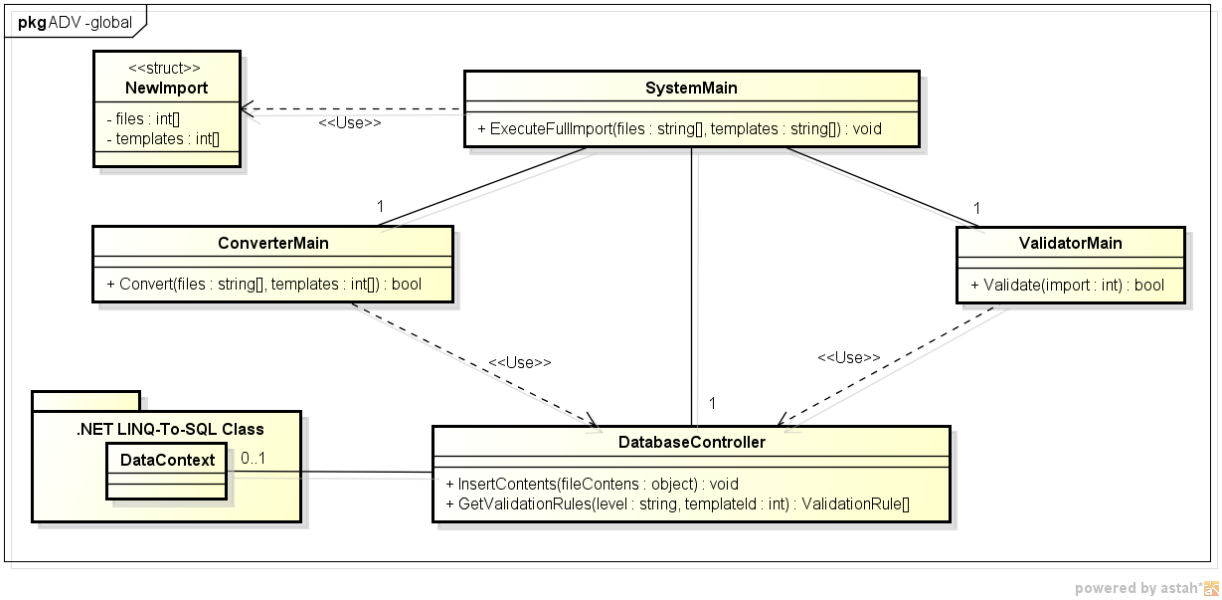
- Een toegangspunt voor de functionaliteiten van de Artena Data Validator. De 'facade class' van het gehele systeem, deze class is tevens het entry point van de Artena Data Validator.
- Een klasse die de meeste (en grote) query's voor de subsystemen uitvoert op de database van de Artena Data Validator.
- De 'core' van het conversiesysteem. Aan dit onderdeel zal de benodigde invoer meegeven worden om een conversie op te starten en dit deel bevat alle onderdelen om een volledige conversie uit te voeren en de juiste uitvoer weg te schrijven naar de database.

[1] http://sourcemaking.com/design_patterns/facade

- De 'core' van de validatie-engine. Aan dit onderdeel zal de benodigde input (de uitvoer van het conversiesysteem) meegegeven worden zodat het systeem de geconverteerde klantbestanden en kan valideren. Dit deel is ook verantwoordelijk voor het vastleggen van de resultaten van de validaties.
- Het log 'systeem', een registratiemiddel is om 'systeemfouten' vast te leggen. Dit systeem zal gebruikt moeten kunnen worden in de gehele Artena Data Validator.

Het toegangspunt van de Artena Data Validator, de facade class *SystemMain*, stuurt het gehele onderliggende systeem aan. Wanneer er iets veranderd aan de implementatie van de conversie –en validatieprocessen dan zal voor de client (die de facade class aanspreekt) weinig veranderen omdat de facade class deze veranderingen 'verbergt'.

Figuur 15 toont de structuur van het toegangspunt en de relaties met andere onderdelen binnen de Artena Data Validator. Na het figuur volgt een beschrijving van de onderdelen.



Figuur 4: Artena Data Validator - Global

SystemMain is de *Facade Class* van de Artena Data Validator en representeert het object die de toegang tot het systeem van buitenaf mogelijk maakt. Via dit object kan een client (zoals een user interface) nieuwe imports(conversie + validatie) initialiseren en is dus de *entry point* van de DLL.

NewImport is het object waarin de specifieke gegevens voor een nieuwe import worden bijgehouden: dit zijn identificaties van de templates (zoals gedefinieerd in de database) en de locaties van de aangeleverde klantbestanden.

ConverterMain heeft controle over het conversieproces van begin tot eind. Deze klasse zal de gewenste datalezers(s) kiezen, het conversieproces starten en de bestandsregels verkregen uit de bestandslezers moeten wegschrijven.

ValidatorMain heeft controle over het validatie proces van begin tot eind. Dit object zal bestandsregels die gevalideerd moeten worden verkrijgen uit de database via de aangereikte verwijzing naar een import. Vervolgens zal het alle geregistreerde validatieregels voor de

bestandsregels moeten ophalen. Met behulp van deze regels moeten de plug-ins verantwoordelijk voor de validaties worden ingeladen en moeten de validaties worden uitgevoerd.

DatabaseController is verantwoordelijk voor de connectie met de database en de uitvoer van alle benodigde query's voor het conversiesysteem en de validatie-engine. Voor de connectie met de database maakt deze klasse gebruik van een DataContext klasse.

DataContext is een LINQ-To-SQL klasse waarmee de databasecontroller en validatie plug-ins op een gemakkelijke manier kunnen communiceren met de database door direct databaseobjecten aan te spreken (i.p.v. query's te bouwen)

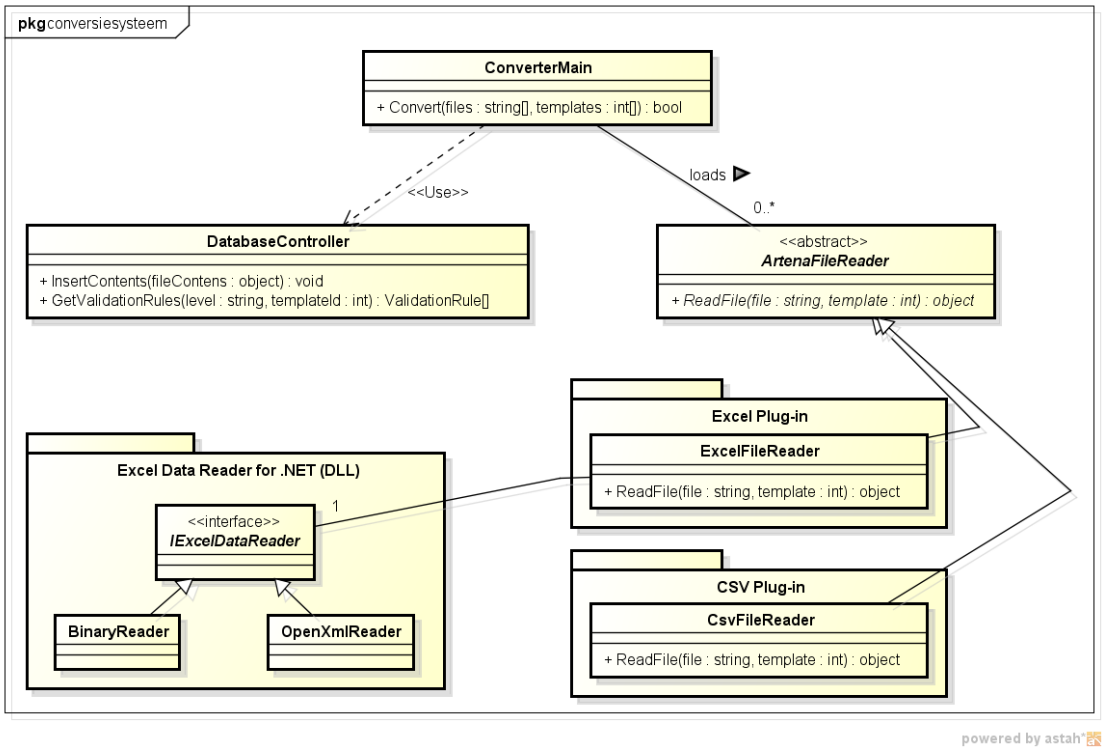
3.3 Subsysteem 1: Conversiesysteem

Het core gedeelte van het conversiesysteem bestaat globaal gezien uit twee onderdelen:

- Een onderdeel dat het gehele conversieproces overziet
- Het plug-in systeem waarmee bestandslezers worden ingeladen
- Een onderdeel dat communiceert met de database en de verantwoordelijkheid heeft om het conversiesysteem te voorzien van de gedefinieerde in –en uitvoer.

3.3.1 Klassendiagram

Het klassendiagram van het conversiesysteem is grotendeels afkomstig uit het ontwerprapport iteratie 1 en aangepast aan de nieuwe situatie. Zie figuur 5.



Figuur 5: Klassendiagram - Conversiesysteem

Om een conversieproces te starten zal de controller een *NewImport* object ontvangen van de facade class *SystemMain* waarin de kenmerken van de import beschreven zijn. De facade class heeft daarvoor de instructie tot conversie ontvangen van een willekeurige client.

Zoals te zien in figuur 16 wordt in dit systeem gebruik gemaakt van een plug-in systeem (zie ontwerprapport iteratie 1) om de conversie van verschillende bestandsformaten te ondersteunen:

- Een bestandsformaat wordt uitgelezen door een object waarvan zijn klasse overerft van een abstracte klasse (**ArtenaFileReader**). De subklasse moet één methode van de abstracte klasse verplicht implementeren zodat de controller (**ConverterMain**) altijd dezelfde methode kan aanroepen en het conversieproces minder aan veranderen onderhevig is.

3.3.2 Beschrijving klassen

ArtenaFileReader is de *abstracte klasse* waarin de een aantal noodzakelijke functionaliteiten van een datareader zijn vastgelegd, maar niet geïmplementeerd. Deze functionaliteiten hebben betrekking op het uitlezen en eventueel manipuleren van data uit een aangeleverd bestand van een bepaald bestandsformaat.

ExcelFileReader is een concrete klasse die overerft van ArtenaFileReader, deze klasse is verantwoordelijk voor het uitlezen van Excel 97-2003 en Excel2007-2010 klantbestanden en de data om te zetten naar het juiste formaat (CSV, zie requirements). Deze klasse is ook verantwoordelijk voor het manipuleren van Excel97-2003 bestanden, zodat de inhoud van deze data hetzelfde formaat (functionele eis **F8)** heeft als de inhoud van de Excel2007-2010 data. Deze klasse maakt gebruik van de Excel Data Reader for .NET bibliotheek.

CsvFileReader is de eigen implementatie voor het, regel voor regel, uitlezen van een Comma Separated Values klantbestand. Eventuele manipulaties aan de data kunnen hier ook worden uitgevoerd.

Excel Data Reader for .NET is de gebruikte (en gekozen) bibliotheek om de data van een Excel bestand uit te lezen en op te slaan in een *DataSet* object.

3.4 Subsysteem 2: Validatie-engine

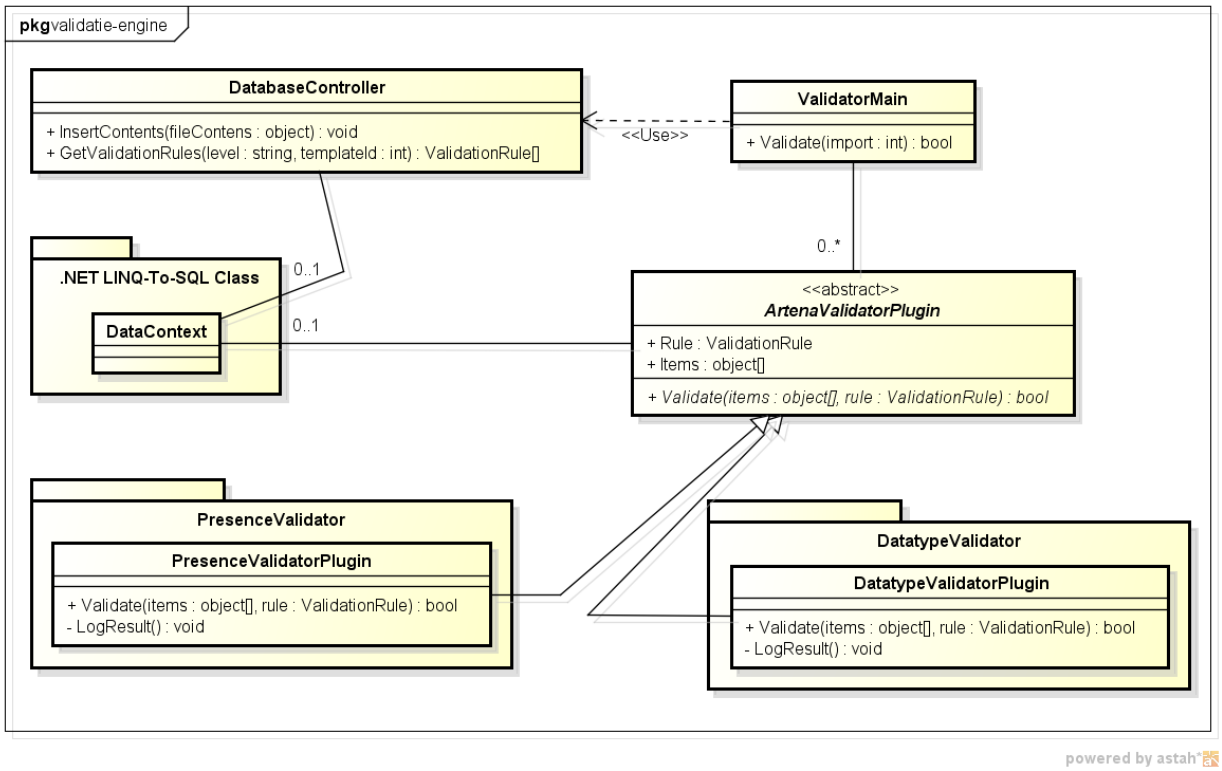
Het core gedeelte van de validatie-engine bestaat globaal gezien uit drie onderdelen:

1. Een controller die het gehele validatieproces (de validatieroutine) overziet
2. Het systeem dat de plug-ins van de validatieregels (op de juiste volgorde) inlaadt en de te controleren bestandsregels en condities meegeeft aan de plug-ins.
3. De validatie plug-in(s) die de bestandsregels op een juiste wijze valideren met de verkregen condities en een eventueel resultaat wegschrijven.

Naast deze onderdelen zal het object verantwoordelijk voor verbinding met de database (de DatabaseController) worden aangesproken om de in de requirements beschreven **invoer** te verkrijgen. De uitvoer (de validatieresultaten) wordt in de validatie plug-ins zelf afgehandeld door gebruik te maken van de DataContext.

3.4.1 Klassendiagram

Het klassendiagram van de validatie-engine is grotendeels afkomstig uit het ontwerprapport van de tweede iteratie en aangepast aan de nieuwe situatie. Zie figuur 6.



Figuur 6: Artena Data Validator - Validatie-engine

Om een validatieproces te starten zal de controller van de validatie-engine een verwijzing naar de net zojuist geïmporteerde bestandsregels ontvangen. Het initiëren van een validatieproces wordt gedaan vanuit de *SystemMain* klasse.

Met behulp van de kenmerken van de ontvangen import moet de engine de bijbehorende validatieregels inladen en vervolgens de regels op de juiste volgorde uitvoeren door één of meerdere validatie plug-ins in te laden .

Wanneer een validatieresultaat niet succesvol is voor een bepaalde regel (of kolom of veld of …) dan zal een validatieresultaat door de plug-in weggeschreven worden naar de database door gebruik te maken van het datacontext object.

3.4.2 Beschrijving klassen

ArtenaValidatorPlugin is de *abstracte klasse* waarin de een aantal noodzakelijke functionaliteiten van een validatieregel zijn vastgelegd, maar niet geïmplementeerd. Deze klasse is de basis waarvan elke plug-in (klasse) gebruik maakt. De functie **Validate()** is zoals eerder beschreven een verplichte implementatie voor een concrete klasse.

PresenceValidatorPlugin is de concrete implementatie van een plug-in waarmee een verplichte invulling van een veld gecontroleerd kan worden. Deze plug-in erft over van de ArtenaValidatorPlugin en wordt geïmplementeerd als een aparte Dynamic Linked Library (package).

DatatypeValidatorPlugin is de concrete implementatie van een plug-in waarmee de waarde in een veld vergeleken kan worden met een datatype. Deze plug-in erft over van de ArtenaValidatorPlugin en wordt geïmplementeerd als een aparte Dynamic Linked Library (package).

3.5 Log systeem

*Functionele eis **F27, F28, NF6***

Het log systeem van de Artena Data Validator is bedoeld als registratiemiddel voor 'systeemfouten'. Het log systeem moet omgaan met een aantal variabelen om een correct resultaat/log te plaatsen in de database:

- Twee niveaus van logging: SYSTEM_ERROR, SYSTEM_WARNING
- Het log bericht
- Een eventuele beschrijving van voorgekomen exceptie

3.5.1 Beschikbaarheid logger

Omdat de logger systeemfouten vastlegt moet de gehele applicatie toegang kunnen hebben tot de logger (en zijn functies). Systeemfouten kunnen vrijwel overal in de applicatie voorkomen en moeten altijd gelogd kunnen worden, zodat de oorzaak van dat soort fouten achteraf te achterhalen is.

3.5.2 Soorten loggers

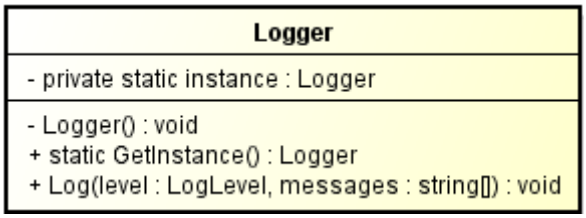
Zoals beschreven zullen alle gegevens die gelogd moeten worden, worden weggeschreven naar een tabel in de database, *SystemLog.* Het is echter mogelijk dat in de toekomst andere soorten loggers ondersteund moeten worden, zoals bijvoorbeeld het wegschrijven van log gegevens naar een bestand.

Het gebruikte log systeem binnen de Artena Data Validator moet dergelijke uitbreidingen (zoals bovenstaand beschreven) kunnen ondersteunen, zodat ook dit 'subsysteem' een mate van flexibiliteit bevat.

3.5.3 Implementatie (Singleton)

Voor de implementatie van een log object zijn er meerdere mogelijkheden:

- Een normale klasse, waarbij elk object dat wil loggen een nieuwe instantie van de klasse moet aanmaken (of verkrijgen). Dit houdt in dat elk object dat gebruikt wordt binnen de Artena Data Validator een dergelijke logger hebben.
- Een statische klasse met daarin statische methoden, waardoor de methoden gebruikt voor het loggen overal beschikbaar zijn (voor elk object binnen de applicatie)
- Een Singleton[2], waardoor de er (net als de statische klasse) maar één log object kan zijn in de gehele applicatie en waarbij de het object overal in de applicatie beschikbaar is.



Figuur 7: Logger (Singleton)

Voor de Artena Data Validator is gekozen voor de laatste oplossing omdat de Singleton de meest object georiënteerde oplossing is. Bij het gebruik van een Singleton kan ook *polymorphism* worden

[2] http://sourcemaking.com/design_patterns/singleton

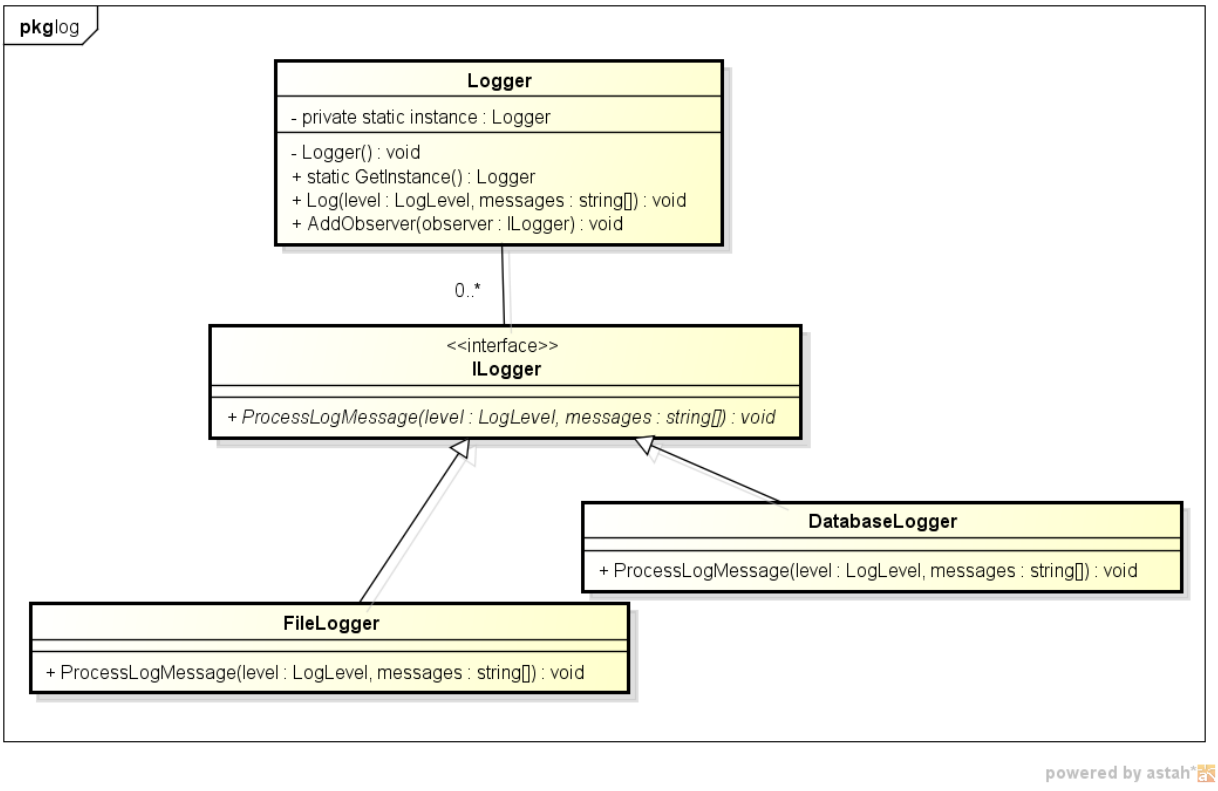
toegepast, heeft de gehele applicatie toegang tot een log object (i.p.v. de log methoden van de statische oplossing) en kan het Singleton object wordt pas geïnitialiseerd wanneer het nodig is. Zie figuur 8.

3.5.4 Observer pattern

Zoals in §3.5.2 beschreven is, moet de mogelijkheid bestaan om meerdere soorten loggers te gebruiken in de Artena Data Validator. Als bijvoorbeeld de connectie met de database faalt dan kan er geen log worden weggeschreven naar die database maar zal er bijvoorbeeld gelogd kunnen worden naar een bestand.

Om deze structuur mogelijk te maken wordt een Observer[3] pattern geïmplementeerd in de Artena Data Validator. Ieder type logger dat een de Artena Data Validator wil implementeren moet als observer worden toegevoegd aan de in §3.5.3beschreven Logger klasse.

Wanneer er een log gemaakt moet worden dan wordt het bericht doorgegeven aan alle observers. De observers handelen het log bericht af (en bepalen dus ook of zij dat bericht moeten loggen). Zie het klassendiagram in figuur 8.



Figuur 8: Volledige Implementatie Logger

De Singleton *Logger* klasse heeft een lijst met nul of meerdere concrete implementaties van de *interface ILogger*. De *ILogger* implementaties zijn dus de observers die worden toegevoegd aan de Logger klasse via de functie **AddObserver()**. In figuur 9 zijn twee observers beschreven, een logger die logt naar een database en een logger die logt naar een bestand. Elke logger bepaalt zelf (op basis van

[3] http://sourcemaking.com/design_patterns/observer

de vorm van het log bericht) of hij wel of niet logt in de implementatie van de **ProcessLogMessage()** methode.

3.6 Configuratie CSV File Reader

*Functionele eis **F26***

Een nieuwe eis die is ontstaan in een late fase van de ontwikkeling is het ondersteunen van meer dan enkele CSV formaten. Deze CSV formaten verschillen op twee punten van elkaar:

- Delimeters/Separatoren: komma(,) of puntkomma(;) of tab(\t)
- Indicatoren: quote(“), apostrof(‘)

Om de verschillende combinaties van delimeters en indicatoren te ondersteunen zal gebruik worden gemaakt van een configuratiebestand die van toepassing is op de gehele applicatie. In dit bestand staat bijvoorbeeld ook de *ConnectionString* waarin de specificatie van de database beschreven is.

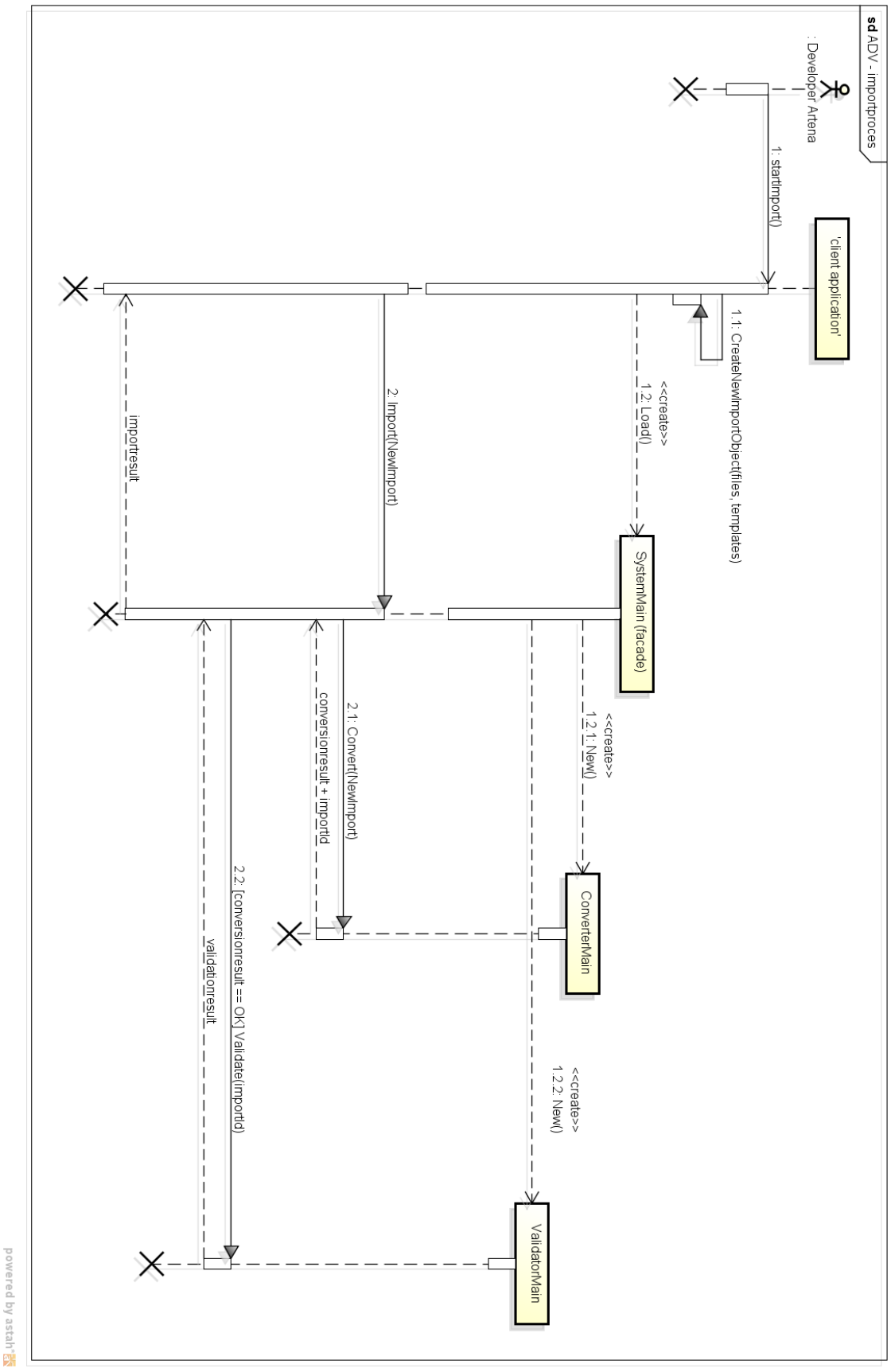
In het configuratiebestand zullen twee variabelen worden toegevoegd waarin de delimeters en indicatoren verwerkt zijn. Bij het inladen van de CSV file reader plug-in (tijdens een conversieproces) zal deze plug-in waarden van de twee variabelen uit het configuratiebestand gebruiken bij verwerking van een CSV bestand.

3.7 Het gehele importproces

Door de samenvoeging van het conversiesysteem en de validatie-engine zal een nieuwe proces ontstaan waarin een conversie en validatie van aangeleverde klantbestanden gecombineerd zal zijn. Figuur 9 beschrijft dit nieuwe proces globaal door middel van een UML sequence diagram.

In het diagram wordt de samenhang tussen de klassen *SystemMain*, *ConverterMain* en *ValidatorMain* beschreven wanneer de gebruiker een import start via een gebruikersinterface of andere client. Het is duidelijk te zien dat een validatie afhankelijk is van een conversie en dat het resultaat van een import afhankelijk is van beide processen.

Het diagram toont niet te verdere verwerking binnen een conversie –of validatieproces aangezien dit al behandelt is in de ontwerprapporten van de eerste en de tweede iteratie.



Figuur 9: UML Sequence diagram - Importproces

4. Begrippenlijst

- Abstract class
  - *Een klasse waarin minimaal één methode abstract is. Dit houdt in dat is aangeven hoe de methode eruit ziet, maar niet wat hij doet (de implementatie). Elke klasse die overerft van de abstracte klasse moet de abstracte methode altijd implementeren*
- ConnectionString
  - *Een string met daarin gegevens over een bepaalde databron (zoals een database) en de wijze waarop verbonden moet worden met de databron. In een dergelijke string staan bijvoorbeeld ook authenticatiemethode of gebruikersnaam/wachtwoord.*
- DataContext
  - *De representatie van een LINQ-to-SQL class. Van dit object kunnen meerdere instanties aangemaakt worden zodat de applicatie met de betrokken database kan communiceren.*
- DataSet
  - *Een .NET object die als een container voor grote hoeveelheden data kan dienen. Een dataset bestaat uit nul of meerdere datatables die elk weer uit nul of meerdere datacolumns en datarows bestaan.*
- File check systeem
  - *Een systeem dat één of meerdere locaties continu scant of pollt en bij upload van een nieuw bestand op één van deze locaties een actie uitvoert*
- Instantie
  - *Een nieuwe kopie van een object dat gebruikt kan worden door een systeem, object of methode*
- Interface
  - *Een 'klasse' die een bepaalde functionaliteit definieert. Alle methoden die in de interface zijn vastgelegd moeten worden geïmplementeerd door een overervende klasse. Lijkt heel erg op abstract class, maar er is geen ruimte voor een default implementatie in een interface definitie.*
- Polymorphisme
  - *Het feit dat een klasse van eenzelfde 'uiterlijk' of 'type' is als een andere klasse maar dat de onderliggende implementatie (en dus functionaliteit) kan verschillen.*

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage N

Constructierapport Iteratie 3: Artena Data Validator

Versie 1.0

Opleiding HBO Informatica – De Haagse Hogeschool

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 09-05-2012

Documentbeheer

Versie	Datum	Status	Omschrijving
1.0	09-05-2012	Eerste versie	Eerste versie constructie met daarin documentatie opgesteld in alle drie de iteraties. + klassendiagram (applicatieniveau) + activiteitsdiagrammen plug-ins + beschrijving user interface + beschrijving gebruikte 'beheer' tool

Inhoudsopgave

1. Inleiding 4
2. Applicatiestructuur 5
2.1 Artena Data Validator – Core 5
2.2 Artena Data Validator – Log systeem 6
2.2.1 Singleton pattern 6
2.2.2 Observer pattern 7
2.3 Artena Data Validator – DataContext 8
2.4 Entiteiten (templates + rules) 9
3. Plug-in systeem 10
3.1 Inladen plug-in & caching 10
3.2 Uitlezen Excel bestand 10
3.3 Datatype validatie 12
4. User interfaces 13
4.1 Client applicatie 13
4.2 Beheer 13
5. Begrippenlijst 15

1. Inleiding

Dit rapport beschrijft een deel van de resultaten gemaakt bij de implementaties in de drie uitgevoerde iteraties voor de Artena Data Validator. Dit rapport kan worden gezien als een samenstelling van de constructies van die drie iteraties.

Dit rapport bevat een klassendiagram van de gehele Artena Data Validator (op applicatieniveau) waarin de structuur van het systeem tot in detail duidelijk wordt (**hoofdstuk 2**).

Daarnaast (**hoofdstuk 3**) bevat dit rapport een beschrijving van het plug-in beheer. Dit hoofdstuk bevat daarnaast ook een beschrijving van de implementatie van een bestandslezer plug-in en een validatie-plug-in in de vorm van UML activiteitsdiagrammen.

Tot slot (**hoofdstuk 4**) bevat dit document de beschrijvingen van de gebruikte interfaces (GUI's). Één GUI wordt gebruikt voor de initialisatie van het importproces en één GUI wordt gebruikt om beheer uit te voeren op de configuraties van de subsystemen van de Artena Data Validator.

2. Applicatiestructuur

In dit hoofdstuk worden meerdere klassendiagrammen op applicatieniveau beschreven. De diagrammen zijn gegenereerd vanuit Visual Studio 2010 en wijken daarom af van de UML notatie. Hieronder de legenda voor een klassendiagram in Visual Studio 2010.

1 op 0..1 relatie

1 op 0..n relatie

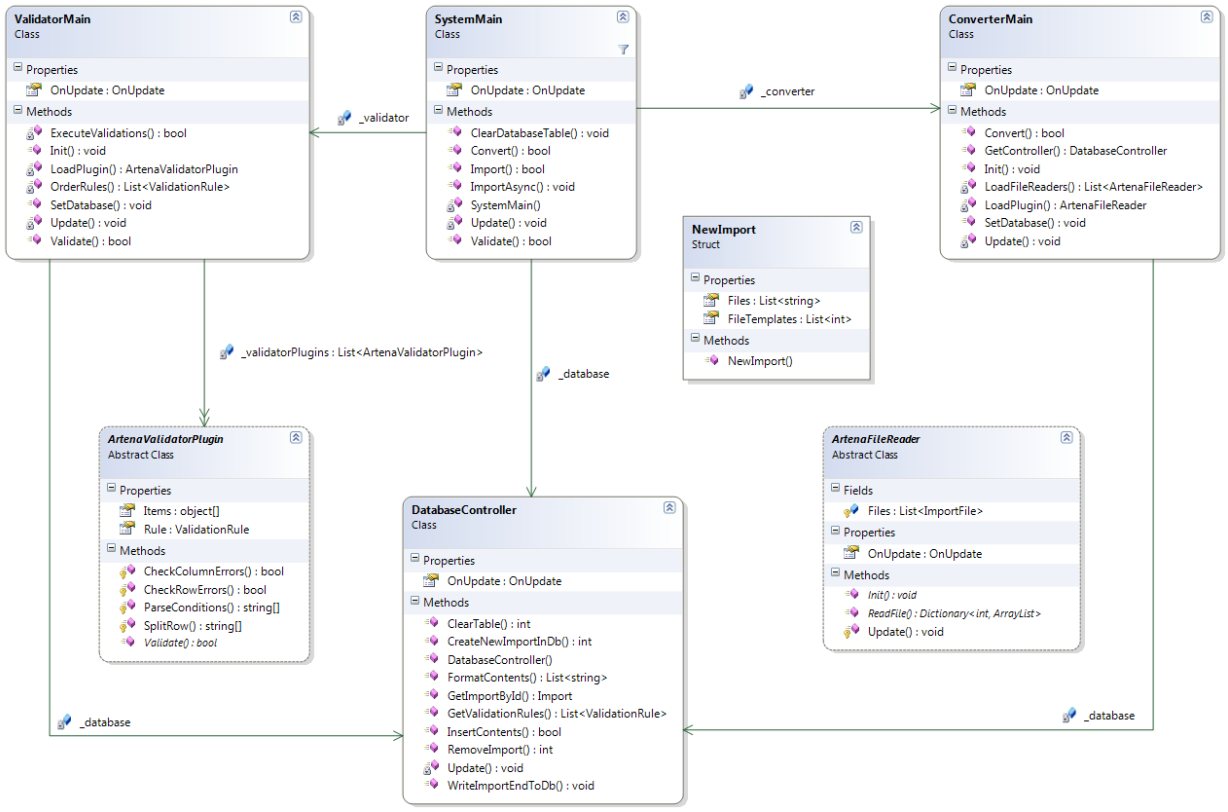
Public methode

Private methode

Private variabele

Bij de klassendiagrammen zijn ook de parameters niet meegenomen omdat ze anders onleesbaar worden.

2.1 Artena Data Validator – Core



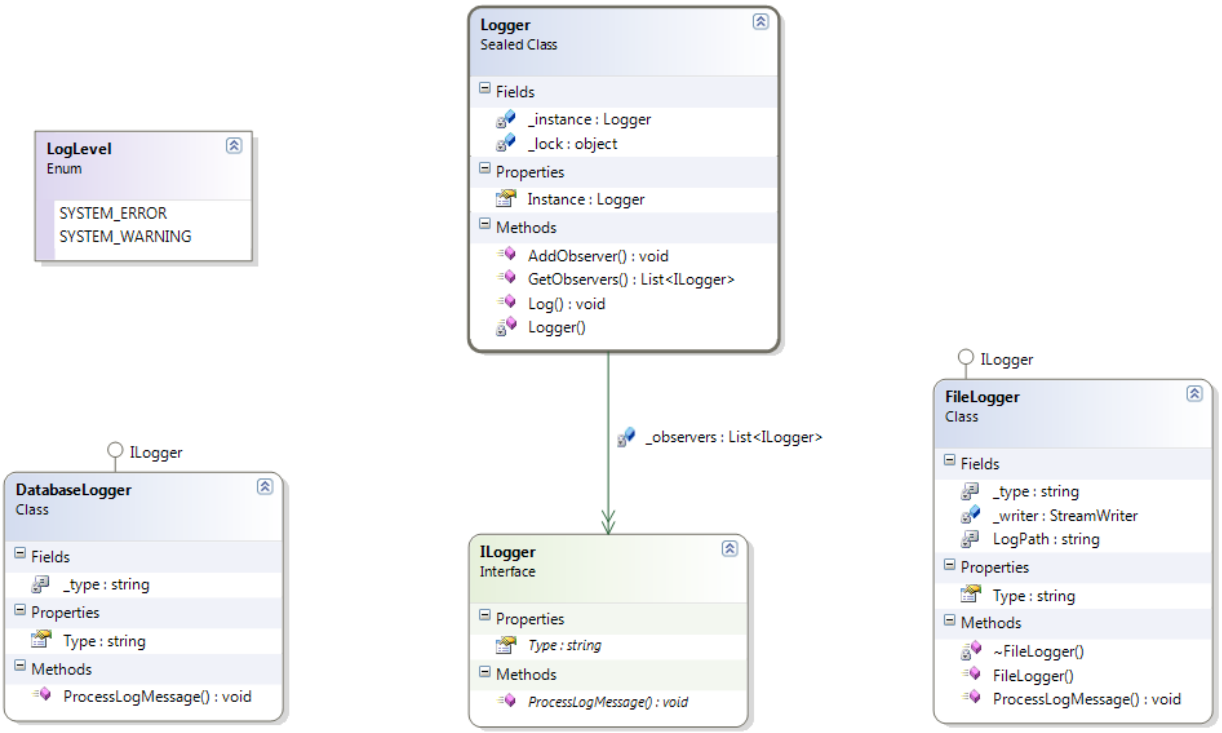
Figuur 1: Core ADV

De structuur in dit diagram komt overeen met het klassendiagram uit het ontwerprapport. Het is duidelijk te zien dat de *SystemMain* class alle andere onderdelen aanstuurt. Met behulp van de methode **Import()** kan de client (zoals een GUI) een conversie en validatie starten door een aantal bestanden (locaties) en templates mee te geven.

De *SystemMain, ValidatorMain en ConverterMain* maken gebruik van de *DatabaseController* voor communicatie met de database. Vanuit de SystemMain wordt één instantie van een DatabaseController meegegeven aan de andere *main* klassen.

Het laatste opvallende kenmerk van deze structuur is dat vrijwel alle klassen een *property* **OnUpdate** bevatten. Deze property omvat een *delegate* waarmee vanuit een *child class(bijvoorbeeld ConverterMain)* gegevens uitgewisseld kunnen worden met zijn *parent class(bijvoorbeeld SystemMain).*

2.2 Artena Data Validator – Log systeem



Figuur 2: Structuur logger

Figuur 2 toont het klassendiagram van het log systeem. De twee(drie) gebruikte design patterns zijn duidelijk terug te vinden in dit diagram. Daarnaast zijn de twee soorten loggers (de *DatabaseLogger* en *FileLogger*) aanwezig.

2.2.1 Singleton pattern

1. **_instance**: private member variabele waarin de instantie van de Logger is opgeslagen als deze is aangemaakt.
2. **_lock**: private member variable gebruikt voor het *double checked locking pattern*
3. **Instance:** property waarmee de aanvraag voor de Logger instantie gedaan kan worden door de Artena Data Validator
4. **Logger()**: *private constructor* zodat de aanvraag voor een logger instantie alleen via 1.3 gedaan kan worden i.p.v. via de (default) constructor

De code hieronder toont de implementatie van de Singleton in de **Instance** property:

```
        public static Logger Instance
        {
```

```
            get
            {
                // If this is the first time we're referring to the
                // singleton object, the private variable will be null.
                if (_instance == null)
                {
                    // for thread safety, lock an object when
                    // instantiating the new Logger object. This prevents
                    // other threads from performing the same block at the
                    // same time.
                    lock (_lock)
                    {
                        // Double-checked locking pattern
                        // Two or more threads might have found a null
                        // _instance and are therefore trying to create a
                        // new one. One thread will get to lock first, and
                        // the other one will wait until _lock is released.
                        // Once the second thread can get through, _instance
                        // will have already been instantiated by the first
                        // thread so test the variable again.
                        if (_instance == null)
                        {
                            _instance = new Logger();
                        }
                    }
                }
                return _instance;
            }
        }
```

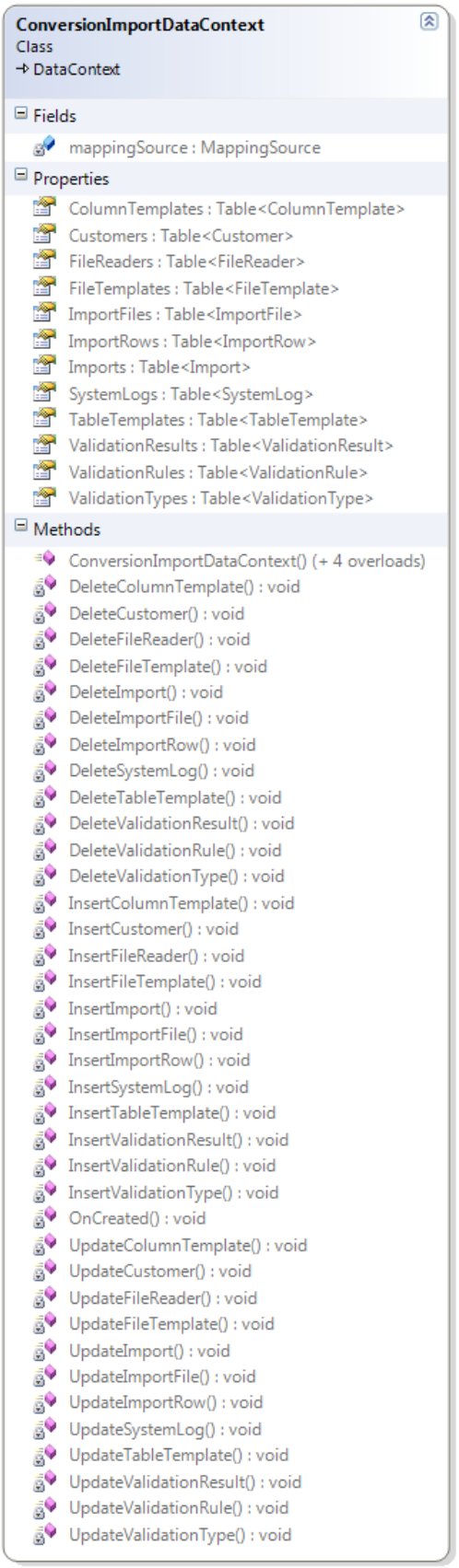
2.2.2 Observer pattern

1. **Interface *ILogger*:** Definieert de structuur van een observer
2. **List<ILogger>**: Geeft de mogelijkheid tot vastleggen van meerdere observers in de *Logger* klasse
3. **AddObserver()**: Methode waarmee van buitenaf ILogger implementatie toegevoegd kan worden aan de *Logger* klasse

De code hieronder toont het verwerken van een logbericht voor 0 of meerdere loggers:

```
        public void Log(LogLevel logLevel, params string[] messages)
        {
            foreach (ILogger observer in _observers)
            {
                observer.ProcessLogMessage(logLevel, messages);
            }
        }
```

2.3 Artena Data Validator – DataContext



Figuur 3: DataContext class

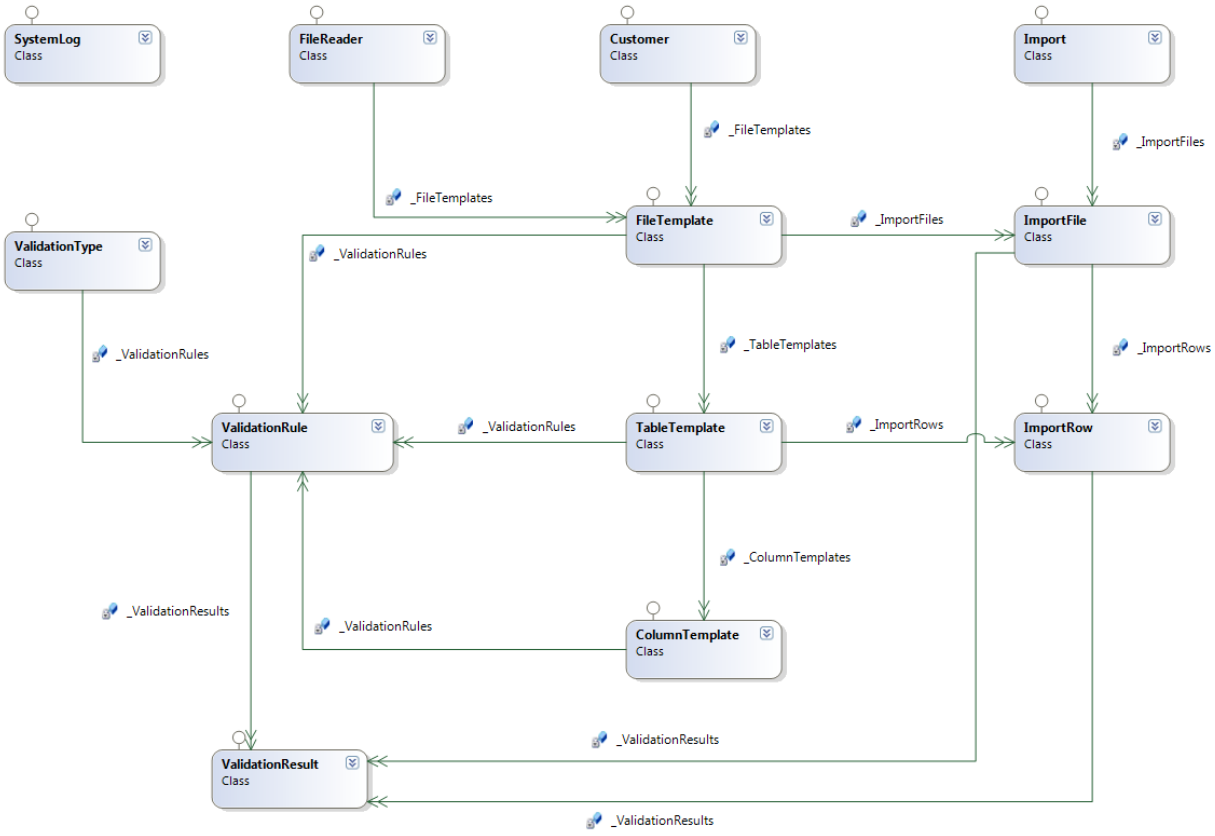
Figuur 2 maakt duidelijk dat de DataContext alle gegevens bevat om *CRUD* operaties uit te voeren op database. Voor elk entiteittype uit de EER diagrammen uit de ontwerprapporten is een property en een verzameling methoden aanwezig in de context.

Bijvoorbeeld de property *Customers* die verwijst naar een lijst van *Customer* entiteiten aanwezig in de database.

Bij deze entiteit horen drie methoden:

1. DeleteCustomer()
2. InsertCustomer()
3. UpdateCustomer()

2.4 Entiteiten (templates + rules)



Figuur 4: Entiteiten genereert door LINQ-to-SQL

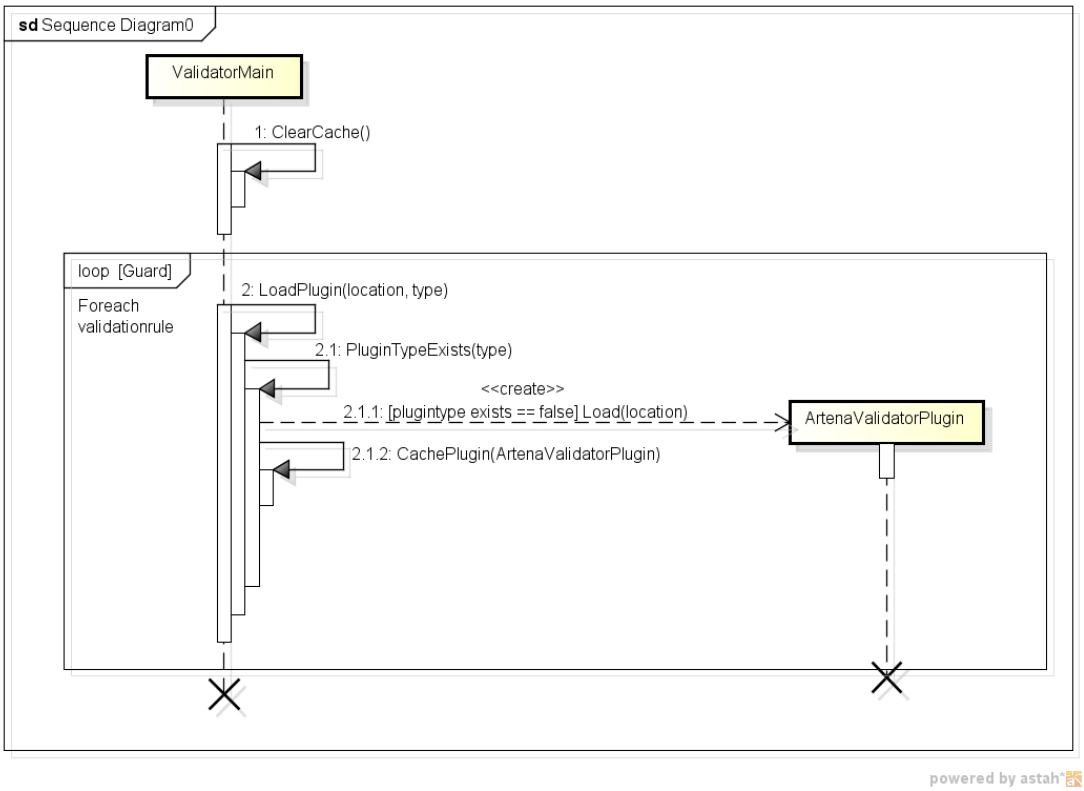
Figuur 3 toont de klassen die gegenereerd zijn vanuit de database door de LINQ-to-SQL mapper. Voor elke tabel/entiteit uit de database is een klasse gecreëerd voor gebruik in de programmatuur. De inhoud (niet weergeven) en de relaties tussen deze klassen komen overeen met de inhoud en relaties aanwezig in de database.

3. Plug-in systeem

Het plug-in systeem dat gebruikt wordt in het conversiesysteem en de validatie-engine is het belangrijkste onderdeel dat zorgt voor de flexibiliteit en uitbreidbaarheid van de Artena Data Validator. In dit hoofdstuk wordt inzicht gegeven in het inladen van een plug-in, het proces dat doorlopen wordt bij het uitlezen van een Excel bestand en het proces dat doorlopen wordt bij het uitvoeren vaneen datatype validatie op een veld.

3.1 Inladen plug-in & caching

Zoals beschreven in de ontwerprapporten wordt een plug-in eerst ingeladen alvorens gebruik gemaakt kan worden van de functionaliteiten die de plug-in biedt. In het geval van de validatie plug-ins worden deze plug-ins tevens gecached gedurende de doorloop van het validatieproces. Dit is belangrijk omdat het inladen van een plug-in (een DLL) veel *overhead* oplevert. Zie figuur 5.



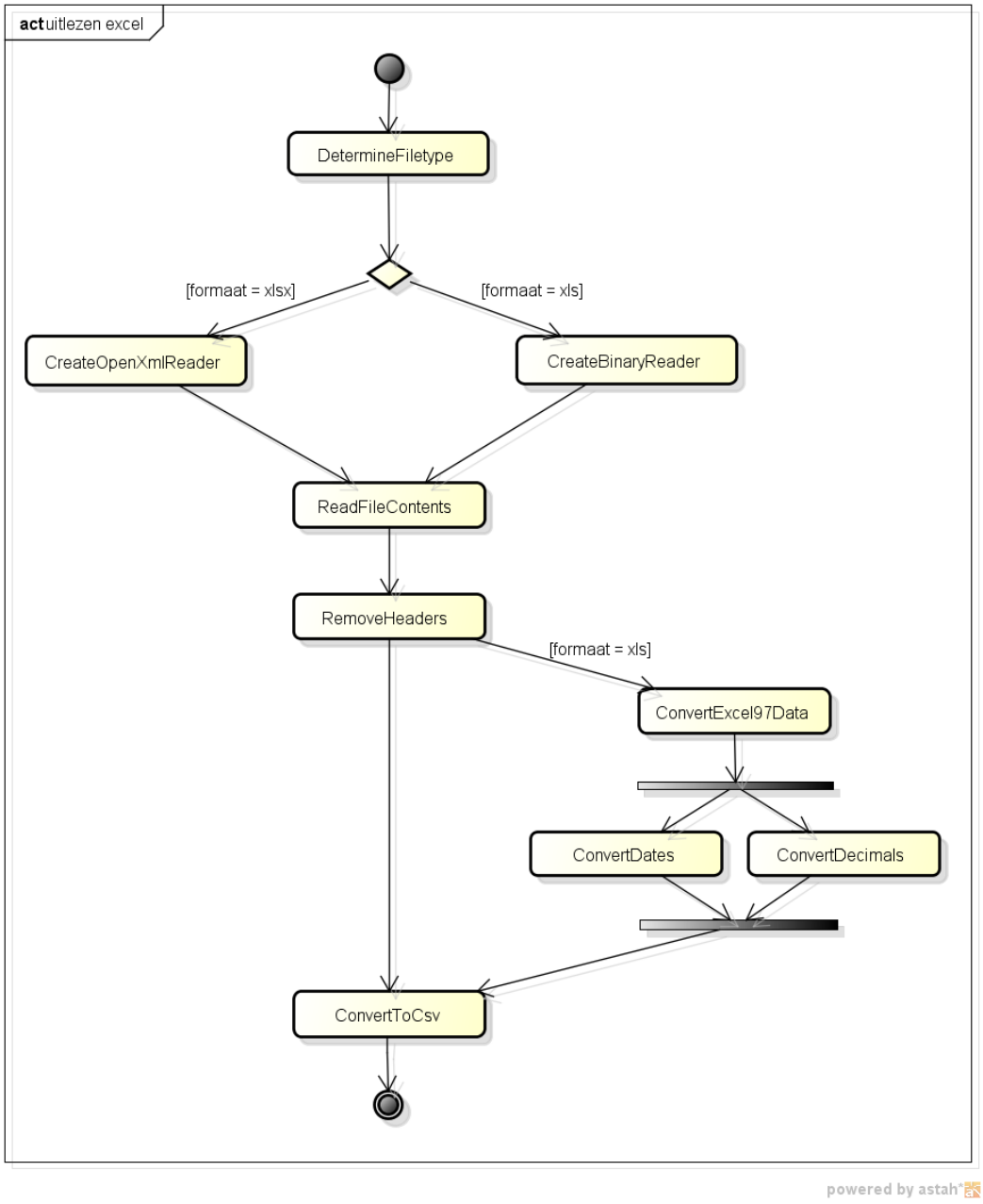
Figuur 5: plug-in loading & caching

Deze vorm van caching is gebaseerd op het *Type* attribuut van de plug-in (dit is ook waar Reflection grotendeels gebruik van maakt). Voor dat een plug-in van een bepaald type ingeladen moet worden wordt eerst gekeken naar een lijst met al eerder ingeladen plug-ins. Indien een plug-in van het gewenste type daar niet in aanwezig is, zal het systeem alsnog een plug-in moeten inladen. De code voor het inladen van een plug-in in zijn simpelste vorm:

```
Assembly assembly = Assembly.LoadFrom(location);
object o = assembly.CreateInstance(type);
ArtenaValidatorPlugin plugin = o as ArtenaValidatorPlugin
```

3.2 Uitlezen Excel bestand

In deze paragraaf wordt het proces waarmee een Excel bestand wordt uitgelezen beschreven. Dit om in detail aan te geven wat er zich kan afspelen in een plug-in. In dit geval wordt de ExcelFileReader plug-in beschreven. Het activiteitsdiagram van figuur 6 geeft het proces weer wat zich afspeelt in de functie *Convert()*.



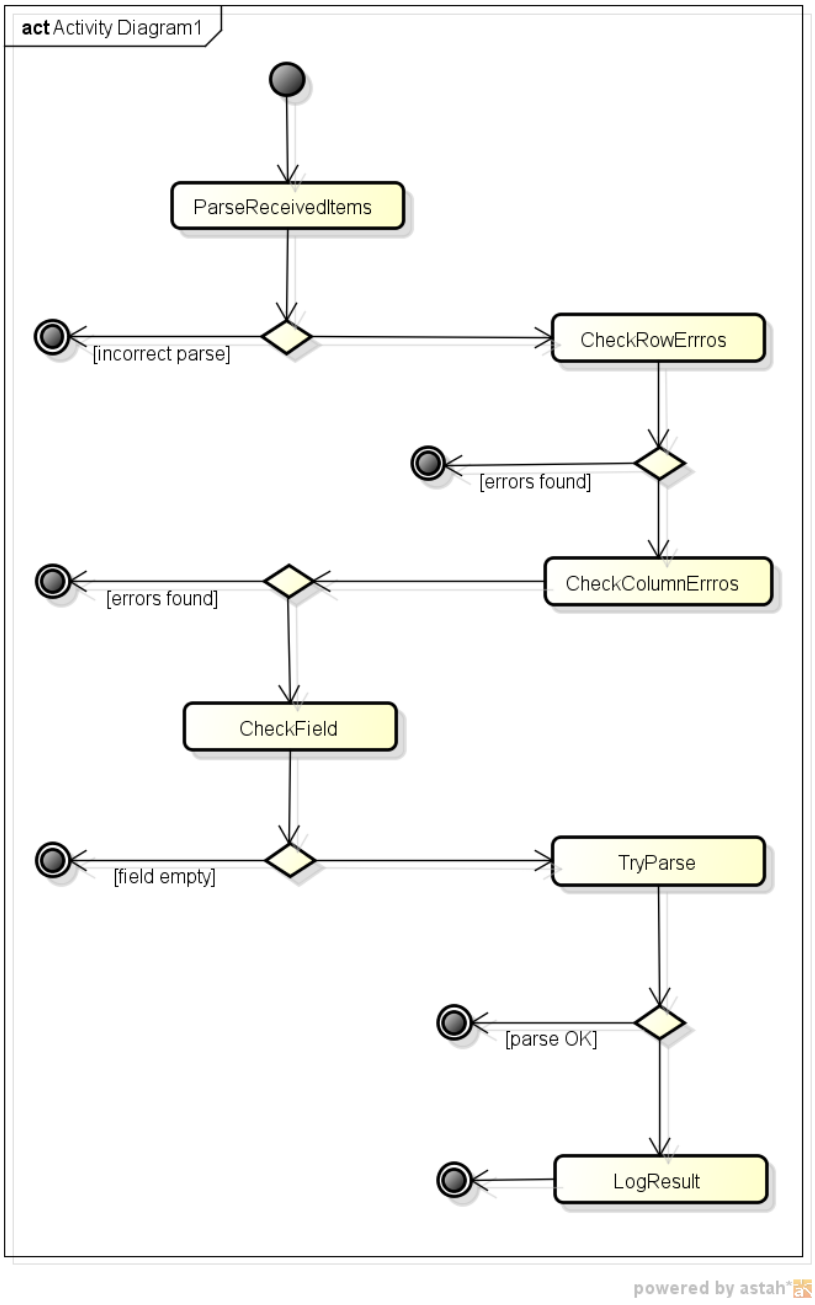
Figuur 6: Activiteitsdiagram - Uitlezen Excel bestand

Het startpunt (bovenin) is de aanroep van de functie *Convert()* vanuit het conversieproces. De functie handelt Excel2007 bestanden anders af dan Excel97-2003 bestanden. Na het verwijderen van de kopregels worden de datums en decimale getallen gemanipuleerd zodat het formaat van deze datatypen overeenkomt met het Excel2007 formaat.

Uiteindelijk worden de uitgelezen bestandsregels omgezet naar CSV formaat (default CSV: "…","…") en geretourneerd naar het bovenliggende (conversie)proces.

3.3 Datatype validatie

In deze paragraaf wordt het proces beschreven dat zich afspeelt wanneer een datatype validatie wordt uitgevoerd door de daarvoor gemaakte plug-in. In figuur 7 staat een activiteitsdiagram waarin het proces van de methode *Validate()* is uiteengezet.



Figuur 7: Activiteitsdiagram - Datatype validatie

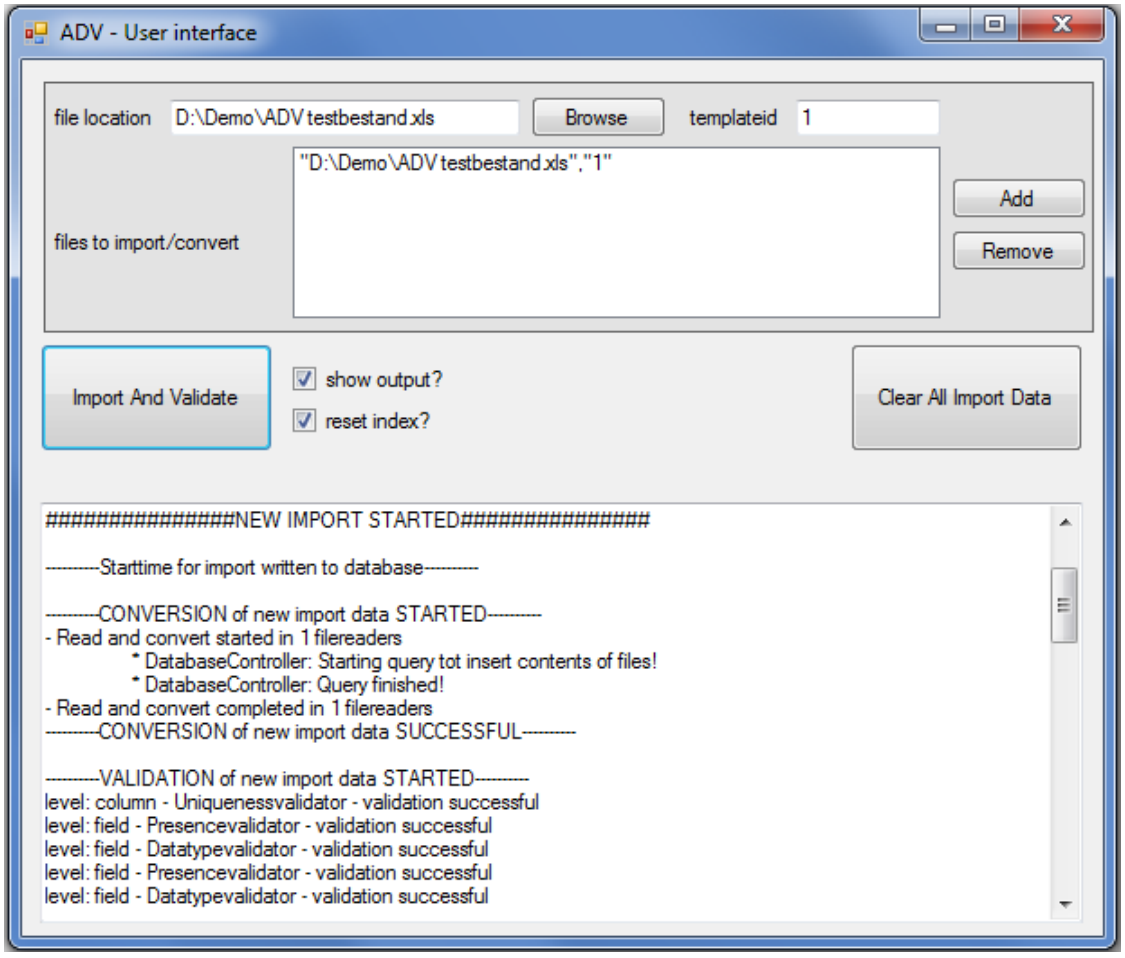
In deze methode worden een aantal controles uitgevoerd op de ontvangen gegevens (vanuit het bovenliggende proces) alvorens een check op datatype wordt gedaan. Het is duidelijk te zien dat er niet gevalideerd wordt als één van deze controles faalt. Wanneer de validatie (de *TryParse*) faalt dan zal een validatieresultaat weggeschreven worden naar de database (de *LogResultI*).

4. User interfaces

4.1 Client applicatie

De client applicatie is de user interface gebruikt tijdens de ontwikkeling van de Artena Data Validator. Via deze interface is het mogelijk om een instantie van de Artena Data Validator in te laden (door de DLL in te laden) en de functionaliteiten van het systeem te gebruiken. De laatste versie van de user interface (figuur 8) ondersteunt onderstaande functionaliteiten:

- Het selecteren van bestanden
- Invoeren van bestand/template combinaties (één of meer)
- Volledige import uitvoeren inclusief conversie en validatie
- Verwijderen van alle imports en importbestanden en importregels



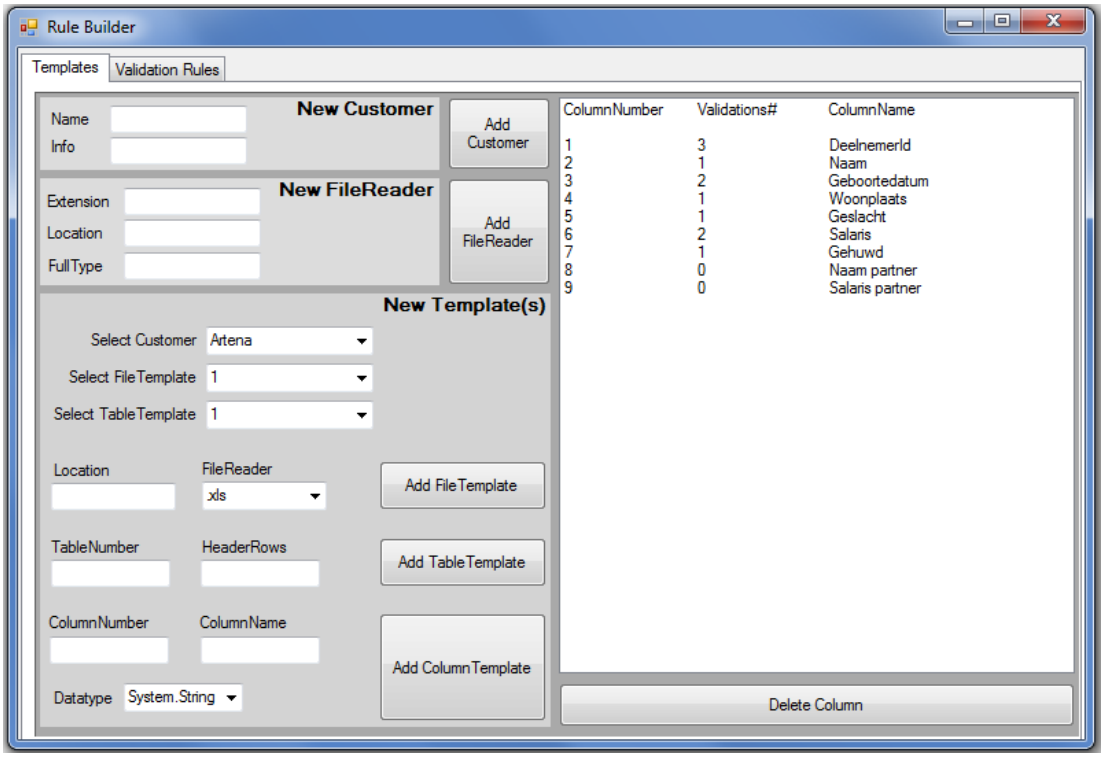
Figuur 8: laatste versie user interface

Deze interface toont tevens een overzicht van succesvolle of niet-succesvolle conversie –en validatieprocessen. Met behulp van deze interface is het mogelijk om snel een nieuwe import door te voeren of alle importdata te verwijderen.

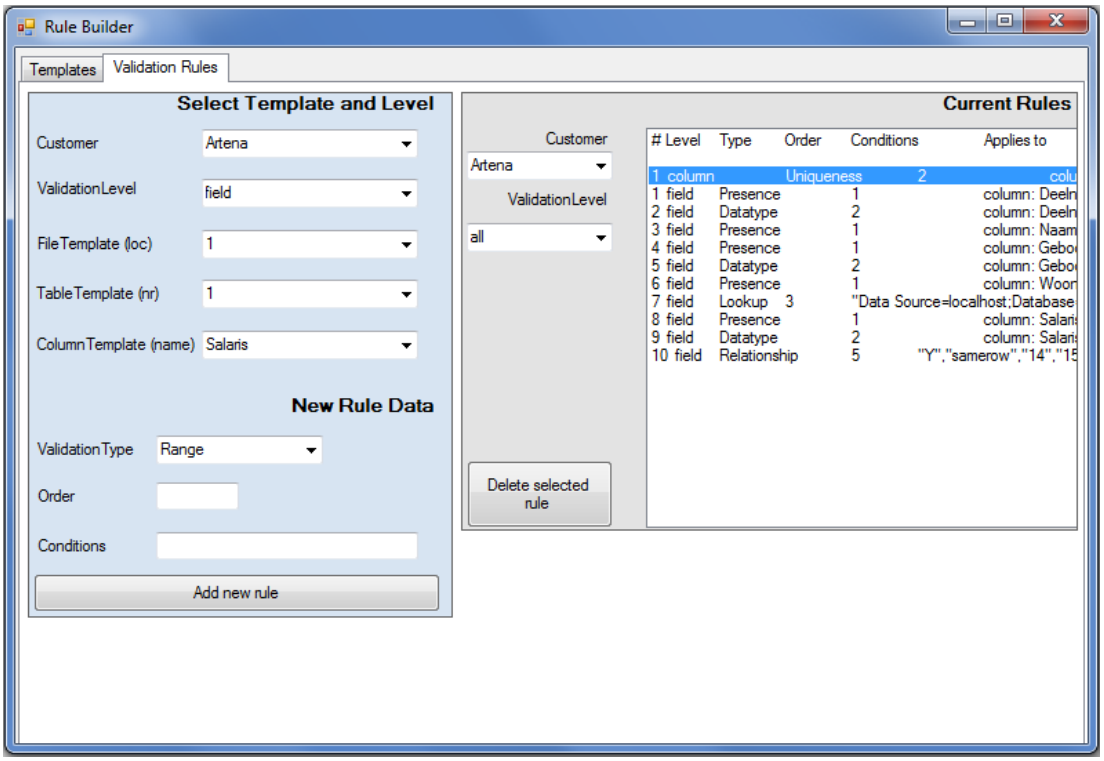
4.2 Beheer

Naast de user interface is ook een beheer interface gemaakt om ‘eenvoudig’ klanten, templates en validatieregels toe te voegen en te verwijderen. Deze interface is vooral een handige tool geweest tijdens de ontwikkeling en zal in deze staat niet worden gebruikt tijdens productie. De applicatie is

symbolisch de ‘rule builder’ genoemd en bestaat uit twee delen. Figuur 9 (deel 1) toont het beheer van templates en figuur 10(deel 2) toont het beheer van validatie regels.



Figuur 9: Rule builer - templates



Figuur 10: Rule builder - validatie regels

5. Begrippenlijst

- Property
    - *Een member van een klasse die wordt gebruikt om een waarde van een andere member variabele uit te lezen of te beschrijven. Een property is een accessor en wordt gebruikt om get en set uit te voeren op variabelen. Een property kan teven publiekelijk toegankelijk gemaakt worden.*
- Delegate
    - *Een type die een verwijzing naar een methode kan bevatten. Wanneer een methode is toegewezen aan een delegate dan zal deze delegate zich op precies dezelfde manier als die methode gedragen. Het is dus mogelijk om een vanuit een willekeurige klasse en methode uit een andere klasse aan te roepen zonder dat de eerste klasse iets afweet van de tweede klasse, als de delegate maar een methode bevat (en dus ook de verwijzing naar de klasse waarin de methode zich bevindt).*
- Child class
    - *Het kind van een andere klasse. Deze benoeming wordt gebruikt als een klasse een andere klasse instantieert en bewaart. De instantie van de klasse is dan de child.*
- Parent class
    - *De klasse die een instantie van een andere klasse aanmaakt en bewaart (gebruikt).*
- Private constructor
    - *Standaard zijn constructors public zodat een instantie van een klasse van buitenaf aangemaakt kan worden. In sommige gevallen loont het om een private constructor te hebben waarbij de klasse alleen 'zichzelf' kan instantieren (Singleton).*
- Double checked locking pattern
    - *Een uitbreiding op het Singleton pattern die vrijwel standaard wordt toegepast bij een implementatie van een Singleton pattern. Dit pattern zorgt ervoor dat de Singleton ook werkt in een multi-threaded omgeving. In een dergelijke omgeving bestaat de kans dat twee threads tegelijkertijd een instantie van de Singleton willen en dat de Singleton twee keer bestaat (terwijl dat niet mag)*
- Type
    - *Elk object is van een bepaald type wat wordt gerepresenteerd door een Type property.*

Afstudeeropdracht: Verbetering van gegevenstransfer bij Artena

Bijlage O

Systeemtest

Testplan – Testscript – Testrapport

Testplan

Versie 1.0

Opleiding HBO Informatica

Afstudeerder: Axel Polet
Studentnummer: 08046816

Datum: 03-05-2012

Inhoudsopgave

1. Testomschrijving 3
1.1 Algemene informatie 3
1.2 Opdrachtformulering 3
2. Testspecificatietechniek 3
3. Testbasis (globaal) 4
4. Testomgeving 4
5. Testproducten 4
6. Testeenheden 5
7. Testbasis (specifiek) 5
7.1 Excel97-2003 bestand 5
7.2 Excel2007 bestand 6
7.3 CSV bestand 6
7.4 Importbestand Ahold 7
7.5 Importbestand De Eendragt 7

1. Testomschrijving

1.1 Algemene informatie

Deze systeemtest zal worden uitgevoerd op de Artena Data Validator, ontwikkeld tijdens een afstudeerproject bij Artena in Den Haag. De systeemtest zal worden uitgevoerd door de ontwikkelaar van de Artena Data Validator (tevens afstudeerder), Axel Polet.

De systeemtest zal worden uitgevoerd wanneer de implementatie (de constructiefases) is afgerond en het opgestelde ontwerp is geïmplementeerd. De test zal worden uitgevoerd met behulp van software zoals deze aanwezig is in de ontwikkelomgeving (waarin die is ontwikkeld).

1.2 Opdrachtformulering

De opdrachtgever van deze systeemtest is tevens de opdrachtgever van het afstudeerproject, Paul Bruggeman (stakeholder 1) en Nico Hoekstra (stakeholder 2).

De scope waarin deze systeemtest valt is dezelfde scope die van toepassing is op het afstudeerproject:

- De systeemtest wordt uitgevoerd op een tool getest in de ontwikkelomgeving
- De gemaakte functionaliteiten zullen worden getest met behulp van zelf ontwikkelde bestanden en oude importbestanden gebruikt door Artena (zie testbasis)
- De tests op de Artena Data Validator zullen worden gestart met behulp van de (tijdens het afstudeerproject) ontwikkelde gebruikersinterface.

Het doel van de systeemtest is het vaststellen of het gemaakte systeem, de Artena Data Validator, voldoet aan de requirements (functioneel en niet-functioneel) die zijn opgesteld tijdens de definitiefases.

2. Testspecificatietechniek

De testspecificatietechniek is de wijze waarop testgevallen worden gespecificeerd voor een bepaalde testsoort. Omdat deze systeemtest tot doel heeft om de aanwezigheid van bepaalde functionaliteiten in de Artena Data Validator aan te tonen moet een dergelijke specificatietechniek daar ook op ingericht zijn.

De testmethode TMap beschrijft een aantal testspecificatietechnieken die toegepast kunnen worden bij de uitvoer van een systeemtest. Van al deze technieken komt de Dataflowtest (DFT) het dichtste in de buurt met het doel van deze systeemtest.

De DFT is een informele testspecificatietechniek waarmee functionaliteiten van een systeem getest kunnen worden. De DFT test concentreert zich vooral op gegevensstromen: worden gegevens voldoende gemanipuleerd (conversie), worden berekeningen goed uitgevoerd (validatie) en sluiten de functies intern goed op elkaar aan (aansluiting tussen conversieproces en validatieproces).

Er wordt in TMap informatie gegeven over de manier waarop de testgevallen samengesteld kunnen worden bij een DFT. Men kan (afhankelijk van het project) hier ook van afwijken. Het zal erop neerkomen dat de testgevallen voor deze systeemtest op basis van **de perceptie van de tester** (dus de ontwikkelaar) zullen worden opgezet.

3. Testbasis (globaal)

De functionaliteit van de Artena Data Validator samengevat:

Het importeren van bestanden (formaat Excel of CSV). Een import bestaat uit een conversie van de inhoud van de bestanden en een validatie op die geconverteerde inhoud.

De testbasis bestaat globaal gezien uit een aantal onderdelen:

- De requirements (uit het analyserapport van de Artena Data Validator) waaruit de verschillende testgevallen af te leiden zijn
- Zelf aangemaakte bestanden
- Oude importbestanden
- De database van de Artena Data Validator met daarin de juiste templates en overige gegevens om een import uit te voeren.

Voor deze systeemtest zullen in totaal vijf bestanden gebruikt worden:

1. Excel97-2003 (xls) bestand (zelf aangemaakt)
2. Excel2007 (xlsx) bestand (zelf aangemaakt)
3. CSV bestand (zelf aangemaakt)
4. Importbestand Ahold (xls)
5. Importbestand Eendracht (csv)

In hoofdstuk 6 staat een detaillering van deze bestanden met daarin een specificatie van de:

- Bestanden
- Tabellen
- Kolommen

4. Testomgeving

De testomgeving is dezelfde als de ontwikkelomgeving, namelijk:

- Operating System: Windows 7
- Ontwikkelomgeving: Visual Studio 2010
- DBMS: MS SQL Server 2008 (management studio)

5. Testproducten

De testproducten die worden opgeleverd zijn:

1. Het testplan (dit document)
2. De bestanden gebruikt bij de testbasis (behalve de klantbestanden)
3. Het testscript (inclusief testgevallen)
4. Het testrapport met daarin de testresultaten, testbevindingen en conclusies

6. Testeenheden

Testeenheden worden gebruikt om de verschillende subsystemen in het te testen systeem te onderscheiden. Deze subsystemen worden ook onderscheid in het **Analyserapport van de Artena Data Validator.**

Onderstaande testeenheden zijn van toepassing bij deze systeemtest:

Subsysteem/Testeenheid
Conversiesysteem
Validatie-engine – Valideren
Validatie-engine – Resultaten
Log systeem

De testcases (die weer bestaan uit testgevallen) worden verdeeld over deze vier testeenheden. Hiermee wordt getracht een beheersbaar testscript op te kunnen stellen.

7. Testbasis (specifiek)

In dit hoofdstuk beschrijf ik de testbasis waarmee getest gaat worden. Vanuit de testbasis zullen één of meerdere testgevallen worden gemaakt zodat de vastgelegde requirements getest kunnen worden. Dit hoofdstuk is ingedeeld in een aantal paragrafen.

In elke paragraaf wordt de structuur van één van de vijf (zie H2) bestanden uiteengezet zoals deze in de vorm van templates in de database van de Artena Data Validator gezet zal worden.

De eerste drie bestanden zijn ook bijgevoegd als bijlage aan dit document (bijlage A-C). In deze bijlagen is ook de inhoud van de bestanden in te zien. Deze inhoud zal veranderd kunnen worden (in kopieën van deze bestanden) om bepaalde testgevallen op te stellen.

7.1 Excel97-2003 bestand

Voor het eerste bestand worden onderstaande gegevens in de database:

Description	FileReader
Excel97 test	ExcelFileReader

TableNumber	HeaderRows
1	1

ColumnNumber	ColumnName	Datatype
1	Klantnummer	System.Int32
2	Naam	System.String
3	Geboortedatum	System.Datetime
4	Geslacht	System.String
5	Salaris	System.Double

6	Burgerlijke staat	System.String
7	Naam partner	System.String
8	Salaris partner	System.Double

7.2 Excel2007 bestand

Het tweede bestand is in .XLSX formaat en bevat niet één maar twee tabellen:

Description	FileReader
Excel2007 test	ExcelFileReader

Tabel 1:

TableNumber	HeaderRows
1	1

ColumnNumber	ColumnName	Datatype
1	Klantnummer	System.Int32
2	Naam	System.String
3	Geboortedatum	System.Datetime
4	Geslacht	System.String
5	Salaris	System.Double
6	Burgerlijke staat	System.String

Tabel 2:

TableNumber	HeaderRows
2	1

ColumnNumber	ColumnName	Datatype
1	Klantnummer	System.Int32
2	Naam partner	System.String
3	Salaris partner	System.Double

7.3 CSV bestand

Het csv bestand bevat uiteraard maar één ‘tabel’ met daarin dezelfde kolommen als het Excel2007 bestand:

Description	FileReader
CSV test	CsvFileReader

TableNumber	HeaderRows
1	3

ColumnNumber	ColumnName	Datatype
1	Klantnummer	System.Int32
2	Naam	System.String
3	Geboortedatum	System.Datetime
4	Salaris	System.Double
5	Burgerlijke staat	System.String

7.4 Importbestand Ahold

Het importbestand van Ahold dat gebruikt gaat worden voor het testen van de Artena Data Validator is behoorlijk groot:

- 25087 regels
- 76 kolommen

Van deze kolommen zullen de eerste 10 worden toegevoegd als template in de database van de Artena Data Validator:

Description	FileReader
Ahold Excel test	ExcelFileReader

TableNumber	HeaderRows
1	1

ColumnNumber	ColumnName	Datatype
1	Rekendatum	System.Datetime
2	BSN	System.Int32
3	Deelnemer_Naam	System.String
4	Deelnemer_tussenvoegsel	System.String
5	Deelnemer_voorletters	System.String
6	Deelnemer_geslacht	System.String
7	Straatnaam	System.String
8	Huisnummer	System.String
9	Huisnummer_toevoeging	System.String
10	Postcode	System.String

7.5 Importbestand De Eendragt

Voor het importbestand van de Eendragt geldt ook dat het bestand behoorlijk groot is:

- 20569 regels
- 155 kolommen

Voor dit bestand zullen wederom weer de eerste 10 kolommen worden toegevoegd als template:

Description	FileReader

Eendragt CSV test	CsvFileReader

TableNumber	HeaderRows
1	1

ColumnNumber	ColumnName	Datatype
1	Stand_Per	System.Datetime
2	Polisnummer	System.String
3	DeelnemerAanhef	System.String
4	DeelnemerNaam	System.String
5	GeboorteDatum	System.Datetime
6	DeelnemerBurgerServiceNummber	System.Int32
7	DeelnemerGeslacht	System.String
8	PartnerAanhef	System.String
9	PartnerNaam	System.String
10	PartnerGeslacht	System.String

Testrapport

Versie 1.0

Opleiding HBO Informatica

Afstudeerder: Axel Polet
Studentnummer: 08046816
Datum: 08-05-2012

Contents

1. Inleiding 3
2. Resultaten testeenheid 1: Conversiesysteem 4
2.1 Testcase 1.1 4
2.2 Testcase 1.2 4
2.3 Testcase 1.3 4
3. Testresultaten testeenheid 2: Validatie-engine (valideren) 5
3.1 Testcase 2.1: Table-level validatie 5
3.2 Testcase 2.2: Row-level validatie 5
3.3 Testcase 2.3: Column-level validaties 5
3.4 Testcase 2.4: Field-level validaties 5
3.5 Testcase 2.5: Overige eisen 6
3.5.1 Functionele eis F19 6
3.5.2 Functionele eis F20 6
3.5.3 Functionele eis F22 6
3.6 ‘ Performance’ test 7
4. Testresultaten testeenheid 3: Validatie-engine – validatieresultaten 8
5. Testresultaten testeenheid 4: Log systeem 9
6. Conclusie 10

1. Inleiding

Dit document bevat de resultaten van de uitgevoerde systeemtest op de Artena Data Validator. De systeemtest is uitgevoerd aan de hand van het testscript.

De resultaten zullen ook worden ingedeeld per testeenheid, testcase en testactie om deze zo overzichtelijk mogelijk te maken. De beschrijvingen van de resultaten zijn zo summier mogelijk. Er zal worden beschreven of de resultaten overeen komen met de verwachte resultaten (controles).

Wanneer het resultaat van de test niet overeenkomt met het verwachte resultaat (= de controlewaarde) dan is niet voldaan aan alle (functionele) requirements. Met behulp van de beschrijving van die resultaten zal de tester/ontwikkelaar de aanleiding tot de problemen op moeten kunnen lossen. Indien dat is gedaan dan kunnen die testcases/testacties opnieuw uitgevoerd worden ter controle.

2. Resultaten testeenheid 1: Conversiesysteem

In dit hoofdstuk zijn de resultaten van de tests uitgevoerd binnen testeenheid 1 vastgelegd. Per testcase is een tabel genoteerd waarin de testacties, controles en resultaten zijn uiteengezet. Wanneer een aan een controle (en dus het verwachte resultaat) is voldaan dan wordt in de kolom '**OK?**' ok genoteerd en wanneer er niet aan is voldaan dan wordt er nok genoteerd.

Voor eventuele bijzonderheden en opmerkingen wordt de kolom '**Opmerking'** gebruikt.

2.1 Testcase 1.1

# Testactie	# Controle(s)	OK?	Opmerking
1	C01-A/C01-C	ok	
2	C02-A/C02-C	ok	
3	C03-A/C03-C	**nok**	CSV bestand niet correct uitgelezen, bestandslezer plug-in stopt met lezen na een lege regel.
4	C04-A/C04-C	**nok**	CSV bestand niet correct uitgelezen, bestandslezer plug-in stopt met lezen na een lege regel.
5	C05-A/C05-B	ok	Gebruikersinterface loopt vast, fout wel geregistreerd in systeem
6	C06-A/C06-C	ok	Alle regels – kopregel zijn uitgelezen

2.2 Testcase 1.2

# Testactie	# Controle(s)	OK?	Opmerking
1	C01-A/C01-D	ok	
2	C02-A/C02-D	ok	
3	C03-A/C03-D	ok	
4	C04-A/C04-D	ok	
5	C05-A	ok	
5	C05-B	**nok**	CSV bestand niet correct uitgelezen, bestandslezer plug-in stopt met lezen na een lege regel. De regels uit het bestand die na de lege regel staan worden niet mee genomen.

2.3 Testcase 1.3

# Testactie	# Controle(s)	OK?	Opmerking
1	C01-A	ok	Beide conversies succesvol uitgevoerd
1	C01-B	ok	Aantal rijen in tabel = 46075

3. Testresultaten testeenheid 2: Validatie-engine (valideren)

In dit hoofdstuk zijn de resultaten van de tests uitgevoerd binnen testeenheid 2 vastgelegd. Per testcase is een tabel genoteerd waarin de testacties, controles en resultaten zijn uiteengezet. Wanneer een aan een controle (en dus het verwachte resultaat) is voldaan dan wordt in de kolom '**OK?**' ok genoteerd en wanneer er niet aan is voldaan dan wordt er nok genoteerd.

Voor eventuele bijzonderheden en opmerkingen wordt de kolom '**Opmerking'** gebruikt.

3.1 Testcase 2.1: Table-level validatie

De table-level validatie *RowsRange* werkt naar behoren.

# Testactie	# Controle(s)	OK?	Opmerking
1	C01	ok	
2	C02	ok	
3	C03	ok	
4	C04	ok	

3.2 Testcase 2.2: Row-level validatie

De row-level validatie *FieldCount* werkt naar behoren.

# Testactie	# Controle(s)	OK?	Opmerking
1	C01	ok	
2	C02	ok	

3.3 Testcase 2.3: Column-level validaties

Beide column-level validaties werken naar behoren: *Uniqueness en FieldsRange*

# Testactie	# Controle(s)	OK?	Opmerking
1 - uniqueness	C01	ok	Geen fouten gevonden
2 - uniqueness	C02	ok	Een fout gevonden en geregistreerd in database
3 - fieldsrange	C03	ok	Geen fouten gevonden
4 - fieldsrange	C04	ok	Een fout gevonden en geregistreerd in database

3.4 Testcase 2.4: Field-level validaties

Alle field-level validaties werken naar behoren:

- Presence
- Range
- Lookup
- Datatype
- Format
- Relationship

# Testactie	# Controle(s)	OK?	Opmerking
1 – Presence, Datatype	C01	ok	Alle gegevens in het bestand voldoen aan de regels, dus geen fouten geregistreerd
2 – Presence, Datatype	C02-A/C02-D	ok	In totaal vier fouten geregistreerd, wat klopt met het aantal fouten aanwezig in het bestand
3 – Range, Format, Lookup	C03	ok	Alle gegevens in het bestand voldoen aan de regels, dus geen fouten geregistreerd
4 – Range, Format, Lookup	C04-A/C04-D	ok	In totaal vier fouten geregistreerd, wat klopt met het aantal fouten aanwezig in het bestand
5 – Relationship	C05	ok	Relatie tussen twee tabellen is aanwezig doordat alle benodigde velden zijn ingevuld, geen fout dus
6 – Relationship	C06	ok	Fout geregistreerd omdat een veld in tabel 2 niet is ingevuld

3.5 Testcase 2.5: Overige eisen

3.5.1 Functionele eis F19

# Testactie	# Controle(s)	OK?	Opmerking
1 – Validatie van elk niveau	Alleen een table-level fout geregistreerd	ok	Overige validaties zijn niet uitgevoerd omdat table-level validatie een fout aangaf.

3.5.2 Functionele eis F20

# Testactie	# Controle(s)	OK?	Opmerking
1 – Column-level + Field-level	Alleen een column-level fout geregistreerd	ok	Field-level validatie (datatype) niet uitgevoerd omdat de gehele kolom een fout bevat

3.5.3 Functionele eis F22

# Testactie	# Controle(s)	OK?	Opmerking
1 – Presence, Datatype	Aanwezige fouten in beide bestanden vastgelegd in db	ok	Voor beide gevalideerde bestanden zijn fouten vastgelegd in de database

3.6 ‘ Performance’ test

Hoewel er geen eisen zijn gesteld aan de performance van de Artena Data Validator is toch een test uitgevoerd waarbij op 10 kolommen van twee grote bestanden validaties uitgevoerd zijn. Het resultaat is eenduidig: de validaties duren langer dan 5 minuten. De afhandeling van verschillende soorten regels kan echter nog geoptimaliseerd worden, voor dit project valt dat buiten de scope.

# Testactie	# Controle(s)	OK?	Opmerking
1 – Meerdere field-level validaties	Doorlopen tijd minder dan 5 minuten	**nok**	Voor het Excel bestand geldt dat met twee validaties op field-niveau voor de eerste 10 kolommen de validatie +/- 15 minuten in beslag nam.
2 – Meerdere field-level validaties	Doorlopen tijd minder dan 5 minuten	**nok**	Voor het CSV bestand geldt dat met twee validaties op field-niveau voor de eerste 10 kolommen de validatie +/- 20 minuten in beslag nam.

4. Testresultaten testeenheid 3: Validatie-engine – validatieresultaten

Na uitvoer van de tests kan geconcludeerd worden dat de resultaten vastgelegd door de validatie-engine voldoen aan de eisen.

# Testactie	# Controle(s)	OK?	Opmerking
1 – Table-level	Resultaat bevat verwijzing naar validatieregel en importbestand	ok	Het resultaat van de table-level validatie bevat een verwijzing naar de validatieregel (en dus ook naar de tabeltemplate) en naar het importbestand waarop het resultaat van toepassing is
2 – Row-level	Resultaat bevat verwijzing naar validatieregel en importregel	ok	Het resultaat van de row-level validatie bevat een verwijzing naar de validatieregel (en dus ook naar de tabeltemplate) en naar de importregel waarop het resultaat van toepassing
3 – Column-level	Resultaat bevat verwijzing naar validatieregel en importregel	ok	Het resultaat van de column-level validatie bevat een verwijzing naar de validatieregel (en dus ook naar de kolomtemplate) en naar de importregel waarop het resultaat van toepassing is
4 – Field-level	Resultaat bevat verwijzing naar validatieregel en importregel	ok	Het resultaat van de field-level validatie bevat een verwijzing naar de validatieregel (en dus ook naar de kolomtemplate) en naar de importregel waarop het resultaat van toepassing is

5. Testresultaten testeenheid 4: Log systeem

De geregistreerde systeemfout voldoet aan de eisen gesteld in het analyserapport en komt overeen met het verwachte resultaat.

# Testactie	# Controle(s)	OK?	Opmerking
1 –	Kolom succes bevat waarde 0	ok	
2 – Row-level	Een foutbeschrijving voldoet aan de gestelde eisen	ok	De fout bevat: • Het deel van het systeem waarin de fout ontstond • Het proces waarin de fout ontstond • De kenmerken (details) van de fout

6. Conclusie

Uit de, in dit document beschreven, resultaten is een conclusie te trekken met betrekking tot de voldoening aan de gestelde functionele eisen. Omdat de tests ook zijn opgedeeld in een aantal testeenheden zijn er vier deelconclusies te trekken uit de resultaten van de systeemtest:

Testeenheid 1: Conversiesysteem

Het converteren van Excel bestanden levert geen problemen op en voldoet aan de eisen. Het converteren van CSV bestanden gaat echter *fout* wanneer een hele regel ergens midden in het bestand leeg is gelaten. De regels die eventueel achter deze lege regel in het bestand aanwezig zijn worden niet meegenomen in de conversie. Volgens de systeemtest voldoet het systeem dus niet aan de eisen, echter is een dergelijke eis (dat alle regels geconverteerd moeten zijn) nergens opgenomen vanwege overbodigheid. Er zal met de stakeholders overlegd moeten worden of deze fout opgelost moet worden

Testeenheid 2: Validatie-engine - valideren

Alle validaties die het systeem volgens de functionele eisen moest ondersteunen zijn met succes getest. Voor elk type validatie is een resultaat vastgelegd in de database omdat ingelezen bestand een fout bevatte. De performance test heeft echter gefaald, maar omdat hier geen specifieke eisen voor zijn vastgelegd kan niet worden gezegd dat het systeem niet aan de eisen voldoet. Er zal echter wel met de stakeholders overlegd moeten worden om na te gaan of de huidige performance acceptabel is.

Testeenheid 3: Validatie-engine - validatieresultaten

De resultaten die zijn vastgelegd door de validatie-engine op de verschillende validatieniveaus bevatten de juiste gegevens om te zeggen dat de validatieresultaten voldoen aan de eisen. Alle gestelde gegevens en relaties om een resultaat te gebruiken zijn aanwezig.

Testeenheid 4: Log systeem

Net als het resultaat van testeenheid 3 voldoen ook systeemfouten geregistreerd in de database aan de vastgelegde functionele eisen. Alle gegevens geregistreerd bij een systeemfout voldoen aan de eisen.

Met behulp van de deelconclusies kan gezegd worden dat het gehele systeem niet voldoet aan de vastgelegde eisen. Echter zijn de voorgekomen fouten niet direct te koppelen aan functionele eis en daardoor niet toepasbaar voor deze test. De fouten zullen uiteraard besproken worden met de stakeholders en eventuele nieuwe eisen die daaruit voortvloeien zullen getest moeten worden.

De conclusie is dus dat de Artena Data Validator voldoet aan functionele eisen.

Testscript

Versie 1.0

Opleiding HBO Informatica

Afstudeerder: Axel Polet
Studentnummer: 08046816

Datum: 03-05-2012

Inhoudsopgave

1. Inleiding 3
2. Testeenheid 1: Conversiesysteem 4
2.1 Testcase 1.1 4
2.1.1 Testacties 4
2.1.2 Controles (verwacht resultaat) 5
2.2 Testcase 1.2 5
2.2.1 Testacties 5
2.2.2 Controles (verwacht resultaat) 6
2.3 Testcase 1.3 6
2.3.1 Testacties 6
2.3.2 Controles (verwacht resultaat) 7
3. Testeenheid 2: Validatie-engine - Valideren 8
3.1 Testcase 2.1: Table-level 8
3.1.1 Testacties 8
3.1.2 Controles (verwacht resultaat) 9
3.2 Testcase 2.2: Row-level 9
3.2.1 Testacties 9
3.2.2 Controles (verwacht resultaat) 10
3.3 Testcase 2.3: Column-level 10
3.3.1 Testacties 10
3.3.2 Controles (verwacht resultaat) 11
3.4 Testcase 2.4: Field-level 11
3.4.1 Testacties 12
3.4.2 Controles (verwacht resultaat) 12
3.5 Testcase 2.5: Overige eisen validatie-engine 13
3.5.1 Functionele eis F19 13
3.5.2 Functionele eis F19 13
3.5.1 Functionele eis F21 14
3.5.2 Functionele eis F22 14
3.6 'Performance' test 14
4. Testeenheid 3: Validatie-engine – Validatieresultaten 16
4.1 Testcase 3.1: Validatieresultaten 16
5. Testeenheid 4: Log systeem 17
5.1 Testcase 4.1: Systeemfouten 17

1. Inleiding

Dit document bevat het testscript zoals deze uitgevoerd moet worden voor de systeemtest op de Artena Data Validator. In dit document worden de testeenheden (zie testplan) getest op hun vereiste functionaliteiten (zie analyserapport) met behulp van meerdere testgevallen.

Om testgevallen vast te leggen is gekeken naar de testbasis (inclusief de functionele eisen). Per testeenheid worden de functionele eisen voor dat subsysteem gegroepeerd worden in verschillende testcases om zo de test beter beheersbaar te maken. Het is dan bijvoorbeeld mogelijk om één testcase opnieuw uit te voeren in plaats van het gehele testscript opnieuw uit te voeren.

Per testcase zijn één of meerdere testgevallen worden gedefinieerd. Op basis van de testgevallen zijn één of meerdere testacties beschreven die samen tot een (deel)testscript leiden. Tot slot is voor elke testactie een verwacht resultaat vastgelegd.

2. Testeenheid 1: Conversiesysteem

Bij deze testeenheid worden de functionaliteiten van het conversiesysteem getest. De testgevallen, uit te voeren acties en controles hebben vooral betrekking op de invoer en uitvoer van het conversiesysteem. Om deze functionaliteiten te testen zullen meerdere imports (=conversies) uitgevoerd worden door op verschillende soorten bestanden (testbasis, zie testplan). Ook zal de inhoud van de bestanden voor sommige tests gewijzigd worden.

2.1 Testcase 1.1

Functionele eisen: **F1 – F5 + F11 + F26**

Onderstaande testgevallen hebben betrekking op de **invoer** die het conversiesysteem verwacht en de afhandeling van deze invoer.

Test geval	Aantal bestanden	Bestands formaten	Ontbrekende velden/regels in bestanden?	Correcte template	Delimiter Indicator
1	1	.xls	Y	Y	Nvt
2	1	.xlsx	N	Y	Nvt
3	1	.csv	Y	Y	CSV default
4	2	.xls & .csv	Y	Y	CSV default
5	1	.xls	N	N	nvt
6	1	.csv	N	Y	; (eendracht)

2.1.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface

1. Importeer het zelfgemaakte Excel 97-2003 bestand
    a. File template = 1
    b. Enkele velden leegmaken

2. Importeer het zelfgemaakte Excel 2007 bestand
    a. File template = 2

3. Importeer het zelfgemaakte CSV bestand
    a. File template = 3
    b. Configuratiebestand – delimiter: , (komma)
    c. Configuratiebestand – indicator: “ (quote)
    d. Enkele velden leeg
    e. Een regel tussendoor leeg
    f. Ontbrekend veld in een regel

4. Importeer het CSV bestand van testactie 3 tezamen met het Excel97-2003 bestand van testactie 1
    a. File templates = 1, 3
    b. Configuratiebestand – delimiter: , (komma)
    c. Configuratiebestand – indicator: “ (quote)

5. **Importeer het Excel97-2003 bestand van testactie 1**
    a. File template = 99

6. Importeer het CSV importbestand van de Eendragt
    a. File template = 5
    b. Configuratiebestand – delimiter: ; (puntkomma)
    c. Configuratiebestand – indicator: (leeg)

2.1.2 Controles (verwacht resultaat)

Onderstaande controles moeten worden uitgevoerd op alle testacties behalve testactie 5:

C01/C06-A: Geen systeemfouten ontstaan tijdens conversieproces (zie ook de tabel *SystemLog)*

C01/C06-B: Alle regels uit de bestanden minus de HeaderRows (zie tabel *TableTemplate*) zijn weggeschreven naar de juiste tabel in de database

C01/C06-C: In kolom *Success* in de tabellen *Import* en *ImportFile* staat de waarde 1

Onderstaande controles moeten worden uitgevoerd op testactie 5:

C05-A: Systeemfout (kan template niet herkennen) vastgelegd in tabel *SystemLog*

2.2 Testcase 1.2

Functionele eisen: **F4 – F8**

Onderstaande testgevallen hebben betrekking op de **uitvoer** die het conversiesysteem genereert.

Test geval	Bestands formaat	Ontbrekende velden/regels in bestanden?	Duizendtal separatoren
1	.xls	Y	N
2	.xlsx	Y	N
3	.xls	N	Y
4	.xlsx	N	Y
5	.csv	Y	N

2.2.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface)

1. Importeer het zelfgemaakte Excel 97-2003 bestand (enkele wijzigingen)
    a. File template = 1
    b. Enkele velden leegmaken
    c. Getallen zonder separatoren
    d. Datumvelden zijn ingevuld

2. Importeer het zelfgemaakte Excel 2007 bestand (enkele wijzigingen)
    a. File template = 2
    b. Enkele velden leegmaken
    c. Getallen zonder separatoren

d. Datumvelden zijn ingevuld

3. Importeer het zelfgemaakte Excel 97-2003 bestand (enkele wijzigingen)
    a. File template = 1
    b. Geen lege velden
    c. Duizendtal separator (punt)

4. Importeer het zelfgemaakte Excel 2007 bestand (enkele wijzigingen)
    a. File template = 2
    b. Geen lege velden
    c. Duizendtal separator (punt)

5. Importeer het zelfgemaakte CSV bestand
    a. File template = 3
    b. Enkele velden leeg
    c. Een regel tussendoor leeg
    d. Ontbrekend veld in een regel

2.2.2 Controles (verwacht resultaat)

Onderstaande controles moeten worden uitgevoerd op testacties 1-4

C01/C04-A: Alle velden (leeg of niet) zijn gescheiden in de vorm van default CSV formaat:
“veld n”,”veld n+1”

C01/C04-B: Datumnotaties in uitvoer van Excel97-2003 en Excel 2007 is van hetzelfde formaat:
dd-mm-yyyy hh:mm:ss

C01/C04-C: Notaties van decimale getallen in uitvoer van Excel97-2003 en Excel 2007 is gescheiden door dezelfde separator: ‘.’ (punt)

C01/C04-D: Getallen in de uitvoer van alle soorten Excel bestanden bevatten geen duizendtalseparatoren.

Onderstaande controles moeten worden uitgevoerd op testactie 5:
C05-A: Alle velden (leeg of niet) zijn gescheiden in de vorm van default CSV formaat:
“veld n”,”veld n+1”
C05-B: Lege regels (0 velden) zijn niet aanwezig in de uitvoer.

2.3 Testcase 1.3

Functionele eisen: **F9**

Voor deze functionele eisen is het voldoende om twee testacties (testgevallen) uit te voeren, omdat deze eisen een zeer duidelijke functionaliteit beschrijven waarop weinig variabelen van invloed zijn.

2.3.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface

1. Start twee instanties van de gebruikersinterface waarmee de Artena Data Validator wordt gecontroleerd:
    a. Instantie 1: voer een import uit voor het Excel importbestand van Ahold
        i. File template = 4
    b. Instantie 2: voer een import uit voor het CSV importbestand van de Eendragt
        i. File template = 5

2.3.2 Controles (verwacht resultaat)

Onderstaande controles moet worden uitgevoerd op testactie 1

C01-A: De kolom *Success* in de tabellen *Import* en *ImportFile* bevatten de waarde '1' in de rijen voor de uitgevoerde conversie

C01-B: De som van het aantal rijen in de tabel *ImportRow* is gelijk aan som van het aantal rijen in de geconverteerde bestanden minus de kopregels.

3. Testeenheid 2: Validatie-engine - Valideren

Voor deze testeenheid zullen de functionaliteiten met betrekking tot het valideren van de 'geconverteerde gegevens' getest worden. Daartoe zullen meerdere validatieprocessen uitgevoerd gaan worden waarmee de inhoud van de bestanden uit de testbasis gevalideerd gaan worden.

Om de uit te voeren tests goed beheersbaar te maken zullen deze worden ingedeeld naar validatieniveau. De vier validatieniveaus (zoals beschreven in de requirements) zijn:

1. Table-level (tabel niveau)
2. Row-level (rij niveau)
3. Column-level (kolom niveau)
4. Field-level (veld niveau)

Per niveau komen verschillende soorten validaties aan bod waarbij per validatie opzettelijk een fout in het bestand aanwezig zal zijn zodat de validatie ook daadwerkelijk een fout moet genereren (volgens de eisen).

De verschillende validaties zullen niet op alle mogelijke bestanden worden uitgevoerd, omdat de invoer van de validatie-engine ongeacht het bestandsformaat hetzelfde zal zijn (dit moet dan aangetoond zijn door de resultaten van de testcases van testeenheid 1: conversiesysteem).

3.1 Testcase 2.1: Table-level

Functionele eisen: **F13, F14**
Te valideren bestanden: **Excel97 (Ahold), CSV (De Eendragt)**

Onderstaande testgevallen hebben betrekking op het valideren op tabel niveau

Test geval	Bestands formaat	Validatietype	Parameters bestandsinhoud
1	.csv	RowsRange	Aantal regels binnen range
2	.xls	RowsRange	Aantal regels binnen range
3	.csv	RowsRange	Aantal regels buiten range
4	.xls	RowsRange	Aantal regels buiten range

Alvorens de testacties uitgevoerd worden moet eerst een set aan validatieregels worden vastgelegd in de database. **Validatieset:**

Validatietype	Testbasis bestand #	Condities
RowsRange	4: Excel97 (Ahold)	Aantal regels tussen 10000 en 20000
RowsRange	5: CSV (Eendragt)	Aantal regels groter dan 15000

3.1.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface)

1. Valideer het Excel 97-2003 importbestand van Ahold
    a. File template = 4

2. Valideer het CSV importbestand van de Eendragt
    a. File template = 5

3. Valideer het Excel 97-2003 importbestand van Ahold
    a. File template = 4
    b. Beperk het aantal regels tot 15000

4. Valideer het CSV importbestand van de Eendragt
    a. File template = 5
    b. Beperk het aantal regels tot 10000

3.1.2 Controles (verwacht resultaat)

C01: Wel validatieresultaat

C02: Geen validatieresultaat

C03: Geen validatieresultaat

C04: Wel validatieresultaat

3.2 Testcase 2.2: Row-level

Functionele eisen: **F13, F15**
Te valideren bestand: **CSV (zelfgemaakt)**

Deze testcase kan alleen worden uitgevoerd met een CSV bestand omdat de enige validatie op dit niveau controleert op het aantal aanwezige velden in een regels. Dit aantal kan alleen in CSV bestanden willekeurig worden aangepast omdat Excel bestanden altijd een juiste hoeveelheid regels produceren in het conversiesysteem.

Onderstaande testgevallen hebben betrekking op het valideren op rij niveau

Test geval	Validatietype	Parameters bestandsinhoud
1	FieldCount	Aantal velden in regel voldoende
2	FieldCount	Aantel velden in één regel te weinig

Alvorens de testacties uitgevoerd worden moet eerst een set aan validatieregels worden vastgelegd in de database. **Validatieset:**

Validatietype	Testbasis bestand #	Condities
RowCount	3: CSV bestand (zelf)	Nvt

3.2.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface)

1. Valideer het zelfgemaakte CSV bestand
    a. File template = 3

2. Valideer het zelfgemaakte CSV bestand
    a. File template = 3
    b. Verwijder een veld op een willekeurige regel in het bestand

3.2.2 Controles (verwacht resultaat)

C01: Geen validatieresultaat

C02: Wel validatieresultaat

3.3 Testcase 2.3: Column-level

Functionele eisen: **F13, F16**
Te valideren bestand: **Excel97 (zelfgemaakt)**

Onderstaande testgevallen hebben betrekking op het valideren op kolom niveau van het zelfgemaakt Excel97 bestand.

Test geval	Validatietype	Parameters bestandsinhoud
1	Uniqueness	Alle velden in de eerste kolom zijn uniek
2	Uniqueness	Één veld in de eerste kolom is niet uniek
3	FieldsRange	Aantal ingevulde velden valt binnen range
4	FieldsRange	Aantal ingevulde velden valt buiten range

Alvorens de testacties uitgevoerd worden moet eerst een set aan validatieregels worden vastgelegd in de database. **Validatieset:**

Validatietype	Testbasis bestand #	Condities
Uniqueness	1: Excel97 bestand (zelf)	Nvt
RowCount	2: Excel97 bestand (zelf)	Aantal ingevulde velden is groter dan 4

3.3.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface)

1. Valideer het zelfgemaakte Excel97-2003 bestand
    a. File template = 1

2. Valideer het zelfgemaakte Excel97-2003 bestand
    a. File template = 1
    b. Zorg voor een dubbele waarde in de eerste kolom van het bestand

3. Valideer het zelfgemaakte Excel97-2003 bestand
    a. File template = 1

4. Valideer het zelfgemaakte Excel97-2003 bestand
    a. File template = 1
    b. Zorg voor minder dan 4 ingevulde velden

3.3.2 Controles (verwacht resultaat)

C01: Geen validatieresultaat

C02: Wel validatieresultaat

C03: Geen validatieresultaat

C04: Wel validatieresultaat

3.4 Testcase 2.4: Field-level

Functionele eisen: **F13, F17**
Te valideren bestand: **Excel2007 (zelfgemaakt)**

Er zijn in totaal zes verschillende validaties die uitgevoerd kunnen worden op veld-niveau. Om de functionaliteit van deze validaties te testen zal op het Excel2007 bestand elke validatie worden uitgevoerd. Dit bestand bevat namelijk alle benodigde kenmerken (zoals de aanwezigheid van twee tabellen) om alle field-level validaties uit te voeren.

Onderstaande testgevallen hebben betrekking op het valideren op veld niveau

Test geval	Validatietype	Kolomnaam	Parameters bestandsinhoud
1	Presence	Naam	Alle velden de kolom zijn ingevuld
2	Presence	Naam	Één van de velden in de kolom is leeg
3	Datatype (int)	Klantnummer	Aantal velden in de kolom zijn getallen
4	Datatype (int)	Klantnummer	Één van de velden in de kolom is tekst
5	Datatype (date)	Geboortedatum	Aantal velden in de kolom zijn datums
6	Datatype (date)	Geboortedatum	Één van de velden in de kolom is tekst
7	Datatype (double)	Salaris	Alle velden in de kolom zijn decimalen
8	Datatype (double)	Salaris	Één van de velden in de kolom is tekst
9	Range (double)	Salaris	Alle velden in de kolom vallen binnen de range
10	Range (double)	Salaris	Één of meerdere velden in de kolom vallen buiten range
11	Range (date)	Geboortedatum	Alle velden in de kolom vallen binnen de range
12	Range (date)	Geboortedatum	Één of meerdere velden in de kolom vallen buiten range
13	Format	Burgerlijke staat	De kolom voldoet aan het formaat
14	Format	Burgerlijke staat	De kolom voldoet niet aan het formaat
15	Lookup	Geslacht	De kolom voldoet aan de lookup waarde (in een lookuptabel) voor een geslacht
16	Lookup	Geslacht	De kolom voldoet niet aan de lookup waarde (in een lookuptabel) voor een geslacht
17	Relationship	Burgerlijke staat	De kolommen gerelateerd aan de gespecificeerde kolom zijn ingevuld
18	Relationship	Burgerlijke staat	De kolommen gerelateerd aan de gespecificeerde kolom zijn niet ingevuld

Alvorens de testacties uitgevoerd worden moet eerst een set aan validatieregels worden vastgelegd in de database. **Validatieset:**

Validatietype	Testbasis bestand #	Condities
Presence	2: Excel07 bestand (zelf)	Nvt
Datatype	2: Excel07 bestand (zelf)	Nvt
Datatype	2: Excel07 bestand (zelf)	Nvt
Datatype	2: Excel07 bestand (zelf)	Nvt
Range	2: Excel07 bestand (zelf)	Range op salaris: tussen de 2000 en 3000
Range	2: Excel07 bestand (zelf)	Range op geboortedatum: groter dan 1-1-1980
Format	2: Excel07 bestand (zelf)	Formaat voor burgerlijke staat: GEH of ALS of SAM
Lookup	2: Excel07 bestand (zelf)	Verwijzing naar de lookuptabel in andere database
Relationship	2: Excel07 bestand (zelf)	Als de waarde van de burgerlijke staat GEH is dan moeten de andere velden o.a. *partnernaam* niet leeg zijn

3.4.1 Testacties

Testacties die uitgevoerd moeten worden bij de testgevallen (m.b.v. user interface)

1. Valideer het zelfgemaakte Excel2007 bestand (testgeval 1-8)
    a. File template = 2

2. Valideer het zelfgemaakte Excel2007 bestand (testgeval 1-8)
    a. File template = 2
    b. Zorg voor de afwijkingen zoals beschreven in de testgevallen 2, 4, 6, 8

3. Valideer het zelfgemaakte Excel2007 bestand (testgeval 9-16)
    a. File template = 2

4. Valideer het zelfgemaakte Excel2007 bestand (testgeval 9-16)
    a. File template = 2
    b. Zorg voor de afwijkingen zoals beschreven in de testgevallen 10, 12, 14, 16

5. Valideer het zelfgemaakte Excel2007 bestand (testgeval 17-18)
    a. File template = 2

6. Valideer het zelfgemaakte Excel2007 bestand (testgeval 17-18)
    a. File template = 2
    b. Zorg voor de afwijkingen zoals beschreven in testgeval 18

3.4.2 Controles (verwacht resultaat)

Onderstaande controles moet worden uitgevoerd op testactie 1

C01: Geen validatieresultaten voor de vier validatieregels uit testgeval 1-8

Onderstaande controles moet worden uitgevoerd op testactie 2

C02-A: Validatieresultaat m.b.t. presence validatie (kolom naam)
C02-B: Validatieresultaat m.b.t. datatype validatie (kolom klantnummer)
C02-C: Validatieresultaat m.b.t. datatype validatie (kolom geboortedatum)
C02-D: Validatieresultaat m.b.t. datatype validatie (kolom salaris)

Onderstaande controles moet worden uitgevoerd op testactie 3
C03: Geen validatieresultaten voor de vier validatieregels uit testgeval 9-16

Onderstaande controles moet worden uitgevoerd op testactie 4
C04-A: Validatieresultaat m.b.t. range validatie (kolom salaris)
C04-B: Validatieresultaat m.b.t. range validatie (kolom geboortedatum)
C04-C: Validatieresultaat m.b.t. format validatie (kolom burgerlijke staat)
C04-D: Validatieresultaat m.b.t. lookup validatie (kolom geslacht)

Onderstaande controles moet worden uitgevoerd op testactie 5
C05: Geen validatieresultaat voor de validatieregel uit testgeval 17(18)

Onderstaande controles moet worden uitgevoerd op testactie 6
C06: Validatieresultaat m.b.t. de relationship validatie (kolom burgerlijke staat)

3.5 Testcase 2.5: Overige eisen validatie-engine

3.5.1 Functionele eis F19

Te valideren bestand: **CSV (zelfgemaakt)**

Voor deze functionele eis zorgen we ervoor dat het CSV bestand faalt op de table-level validatie omdat er te weinig rijen aanwezig zijn het bestand. Naast de table-level validatie zijn er nog drie andere validaties ingesteld voor het bestand. Deze validaties (row, column –en field-level) mogen niet uitgevoerd worden in dat geval.

Validatieset:

1. RowsRange validatie: minimaal 4 rijen aanwezig in het bestand
2. RowCount validatie
3. Uniqueness validatie op kolom 1
4. Presence validatie op kolom 2

Testactie:

Valideren van Excel97 terwijl te weinig rijen aanwezig zijn (minder dan 4), één rij te weinig velden bevat, de eerste kolom niet uniek is en niet alle velden ingevuld in de tweede kolom

Verwacht resultaat:

In de database is alleen een validatieresultaat aanwezig voor de table-level validatie

3.5.2 Functionele eis F19

Te valideren bestand: **CSV (zelfgemaakt)**

Voor deze functionele eis zorgen we ervoor dat het CSV bestand faalt op een column-level validatie omdat er te weinig ingevulde velden aanwezig zijn in een kolom. De field-validaties aanwezig in die kolom mogen dan niet meer worden uitgevoerd.

Validatieset:

1. FieldsRange validatie: minimaal 4 ingevuld velden aanwezig in kolom 3
2. Datatype validatie op kolom 3

Testactie:

Valideren van Excel97 terwijl te weinig ingevulde velden aanwezig zijn (minder dan 4) en een aantal waarden in kolom 3 niet van het datatype *Datetime* is.

Verwacht resultaat:

In de database is alleen een validatieresultaat aanwezig voor de column-level validatie

3.5.1 Functionele eis F21

Functionele eis is bewezen doordat tijdens de ontwikkeling meerdere plug-ins zijn toegevoegd om nieuwe soorten validaties uit te voeren. De wijze waarop dit gebeurt is momenteel nog behoorlijk technisch en valt daarmee buiten de scope van deze systeemtest.

3.5.2 Functionele eis F22

Te valideren bestanden: **Excel97 (zelfgemaakt) en CSV (zelfgemaakt)**

Voor deze functionele eis moet een willekeurige set aan validaties voor een Excel97 bestand en een CSV bestand worden toegevoegd aan het systeem. Vervolgens moet onderstaande testactie uitgevoerd worden:

Testactie:

Zorg dat beide bestanden fouten bevatten zodat validatieresultaten worden vastgelegd.

Start twee instanties van de gebruikersinterface waarmee de Artena Data Validator wordt gecontroleerd. Start in beide instanties een validatie op (instantie 1: Excel97, instantie 2: CSV).

Verwacht resultaat:

In de database zijn validatieresultaten aanwezig voor beide bestanden.

3.6 'Performance' test

Te valideren bestanden: **Excel97(Ahold), CSV (De Eendragt)**

Tot slot zal een test worden uitgevoerd waarbij de performance van de Artena Data Validator getest zal worden. Het doel is de waarover een validatieproces doet bij een groot bestand vast te leggen.

Om dit te bereiken zal onderstaande validatieset gebruikt moeten worden:

1. Excel97: Op elke kolom van de 10 beschreven kolommen (zie testbasis in testplan) een presence en datatype validatie
2. Excel97: Op een aantal kolommen één extra validatie (afhankelijk van het soort kolom)
3. CSV: Op elke kolom van de 10 beschreven kolommen (zie testbasis in testplan) een presence en datatype validatie

4. Excel97: Op een aantal kolommen één extra validatie (afhankelijk van het soort kolom)

Testacties:

1. Importeer het Excel 97 bestand (+/- 25000 regels & 76 kolommen)
    a. Template = 4

2. Importeer het CSV bestand (+/- 20000 regels & 155 kolommen)
    a. Template = 5

Verwacht resultaat (voor zowel testcase 1 als testcase 2):

In het geval dat zich geen fouten voordoen:

- Importtijd is minder dan 5 minuten
- Geen validatieresultaten in de database

In het geval van gevonden fouten:

- Importtijd is minder dan 5 minuten
- Validatieresultaten in de database

4. Testeenheid 3: Validatie-engine – Validatieresultaten

Om vast te kunnen stellen of de vastgelegde validatieresultaten aan de eisen voldoen is het mogelijk om een aantal testacties/testgevallen uit de vorige testcases opnieuw te gebruiken.

4.1 Testcase 3.1: Validatieresultaten

Functionele eis: **F23-F25**

Testacties:

1. Testactie 3 van testcase 2.1 (zie hoofdstuk 3.1) – Table-level
2. Testactie 2 van testcase 2.2 (zie hoofdstuk 3.2) – Row-level
3. Testactie 2 van testcase 2.3 (zie hoofdstuk 3.3) – Column-level
4. Testactie 2 van testcase 2.4 (zie hoofdstuk 3.4) – Field-level

Verwacht resultaat:

Voor elk van de validatieresultaten vastgelegd in testacties 1-4 geldt dat deze resultaten van een bepaald niveau zijn (Error | Warning) en dat deze resultaten voldoen aan de eigenschappen vastgelegd in functionele eis F24:

- De validatieregel (type en niveau) en condities waar niet aan is voldaan
- De reden waarom er niet is voldaan aan de condities
- Het tabelnummer waarop de validatie betrekking had
    - Table-level
    - Row-level
    - Column-level
    - Field-level
- Het rijnummer waarop de validatie betrekking had
    - Row-level
    - Field-level
- Het kolomnummer waarop de validatie betrekking had
    - Column-level
    - Field-level

5. Testeenheid 4: Log systeem

Het testen van de functionaliteit van het logsysteem is vrij gemakkelijk, omdat de eisen voor dit systeem zich beperken tot de eisen aan de data die in de database wordt vastgelegd van een systeemfout. Er hoeft dus alleen gekeken te worden naar de tabel *SystemLog.* Er kan ook voor deze test een al eerder uitgevoerde testactie gebruikt worden.

5.1 Testcase 4.1: Systeemfouten

Functionele eis: **F27-F28**

Testactie:

1. Start een conversie voor een Excel97-2003 bestand:
    a. Templateid = 1
    b. Zorg ervoor dat de bestandslezer voor Excel bestanden niet aanwezig is op de in de database vastgelegde locatie.

Verwacht resultaat:

C01-A: De kolom *Success* in de tabellen *Import* en *ImportFile* bevatten de waarde ‘0’ in de rijen voor de uitgevoerde conversie
C01-B: In de tabel *SystemLog* is een registratie gemaakt van de fout waarin de volgende gegevens aanwezig zijn:

- Het deel van het systeem waarin de fout ontstond
- Het proces waarin de fout ontstond
- De kenmerken (details) van de fout

Bijlage P – Systeemdocumentatie

Artena Data Validator

Features – v1.0

This standard is used by:

Artena Business Consulting

All rights reserved. Nothing in this document may be copied or multiplied and/or made public, in any possible manner, without prior written confirmation of Artena Business Consulting.

Artena Business Consulting

Koninginnegracht 29

2514 AB THE HAGUE

The Netherlands

phone +31 (070) 363 77 33

Contents

Contents

1. Introduction _____ 3

2. Validation process _____ 4

3. Supported file types _____ 6

4. Supported validation types _____ 7

Versioning

Version	Date	Status	Author	Comments
1.0	16-05-2012		A.O. Polet	

1. Introduction

1.1 Introduction

This document describes the main features of the Artena Data Validator. This application is developed in order to validate data of files (or datasets) that have to be imported into different systems used by Artena.

This document includes a description of:

- the steps which are undertaken to validate an import (**ch2**)
- the different file formats which the Artena Data Validator supports (**ch3**)
- the different types of validations the Artena Data Validator can perform (**ch4**)

1.2 Audience

This document is meant for the developers of Artena who have a need to validate newly delivered files from clients or who want to change the configuration of different validations by changing the conditions of various validation rules.

2. Validation process

2.1 Template structure

The Artena Data Validator bases itself on a template structure which is defined in a database. The template structure consists of three template types:

- File template: describes file including file type
- Table template: describes table including table number and header rows to ignore
- Column template: describes column including column number, column name and data type.

A file template contains 1 or more table templates and a table template contains 0 or more column templates. Each file that needs to be validated has to be described as a template in the database used by the Artena Data Validator. See figure 1

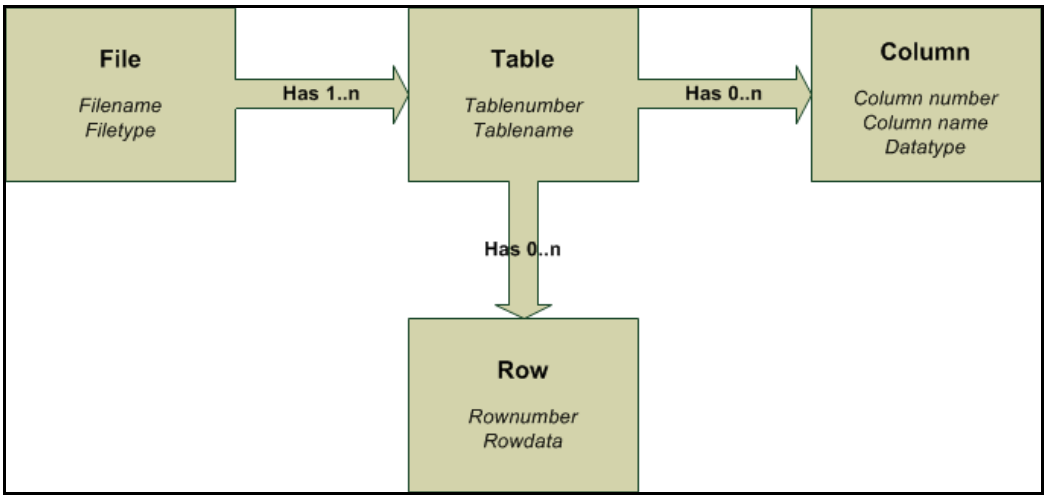


FIGURE 1: FILE STRUCTURE

It is possible to define a set of validation rules on each template so that the validation of the file can be executed on multiple levels (see chapter 4)

2.2 Total validation process

The validation process is dissected in a two separate processes, the conversion/staging process and the validation process. The conversion process (figure 2):

1. *Reading the delivered file(s) according to the defined template (conversion)*
2. *Writing the data of the delivered file(s) to the database in which the data is related to the defined templates (conversion)*

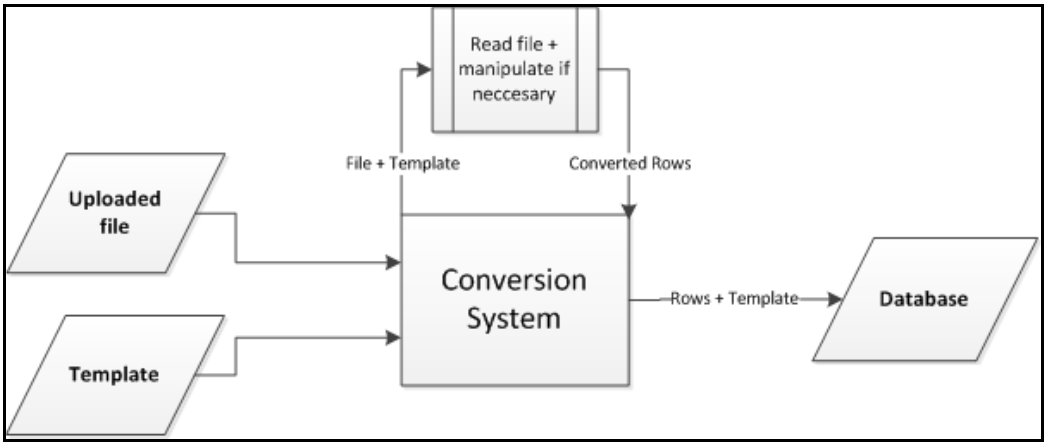


FIGURE 2: CONVERSION PROCESS (STEPS 1-2)

After an conversion/staging has been successfully executed the validation process is started in which the converted files (rows of those files) are validated(figure 3):

3. *Loading a set of validation rules for each level (see chapter 4) from the database in which the templates are also set*
4. *Executing the validation rules for each level on different parts of the data and capturing 'wrong' values by inserting a validation result in the same database*

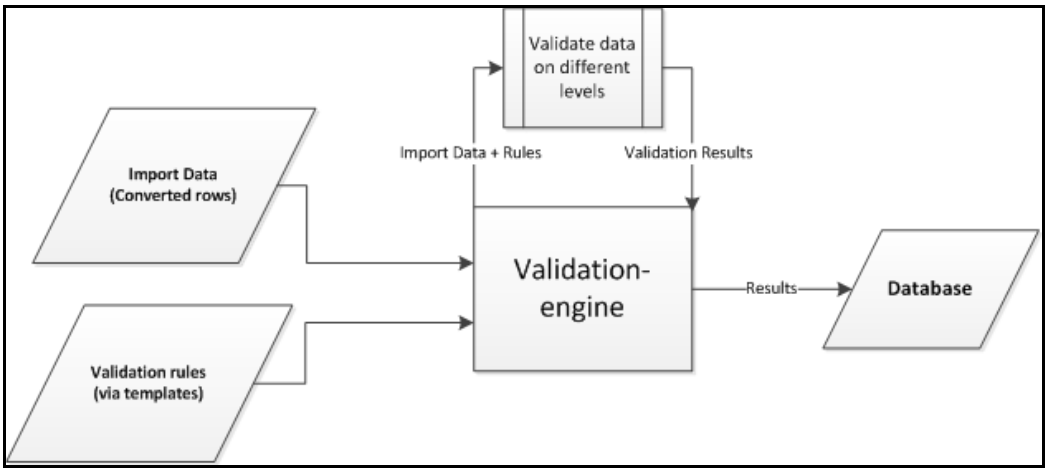


FIGURE 3: VALIDATION PROCESS (STEPS 3-4)

2.3 Plug-in system

The Artena Data Validator uses a plug-in system to implement great flexibility when it comes to adding or changing supported file types and validation types.

The plug-in system is implemented in the same way for both the file types and the validation types:

- Each file type or validation type is handled by a separate plug-in which is implemented as a DLL in C#
- For each plug-in (DLL) the database used by the Artena Data Validator holds two records: the location and type of the plug-in (see ch3 & ch4).

3. Supported file types

3.1 File types

The Artena Data Validator currently supports these types:

- Excel97-2003, extension: **.xls**
- Excel2007-2010, extension: **.xlsx**
- Comma Seperated Values (CSV), extension: **.csv**

For the Excel97-2003 and the Excel2007-2010 files the same filereader plug-in is used to read the contents of the file: ExcelFileReader.dll

The CSV files are read by another filereader plug-in: CsvFileReader.dll

3.2 Plug-in data

The data for the filereader is related to a FileTemplate in the database used by the Artena Data Validator. This is done by a relationship with the *FileReader* table which contains two types of data for loading a plug-in:

- Location, for instance “C:\Software\ExcelFilereader.dll
- Type, for instance “Artena.DataValidator.ExcelFileReader”

By using these two strings the conversion routine can automatically load a filereader plug-in and pass a reference to the file which needs to be read. If the strings carry incorrect data such as a not existing location the conversion will be aborted.

4. Supported validation types

4.1 Description

The Artena Data Validator supports multiple validation types which can be executed if configured. The most important feature is the ability to apply validations on multiple levels: file, table, row, column and field. Each validation level applies to a different section of the file that needs to be validated, see figure 2.

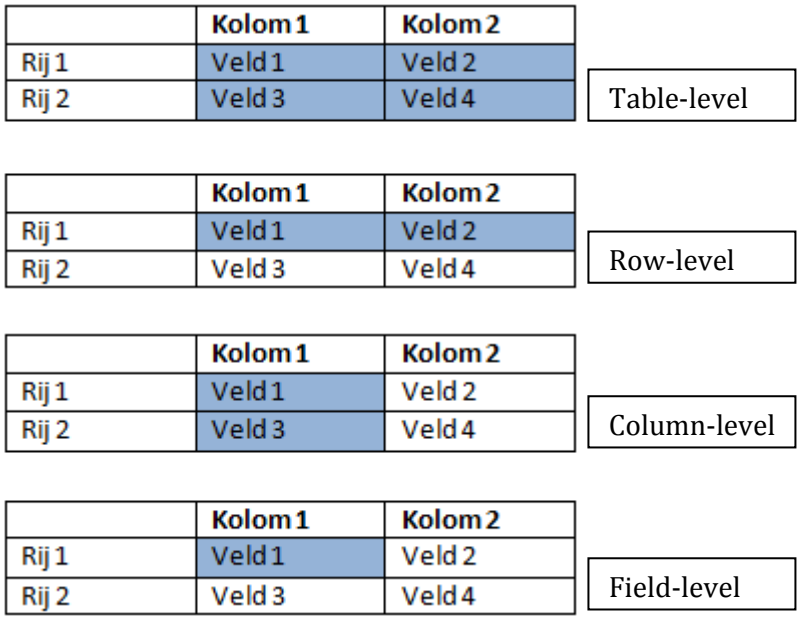


FIGURE 4: VALIDATION LEVELS

File-level validations are currently not implemented and therefore not shown in the image above.

Validations are linked to a different template(chapter 2) based on the level on which they are executed:

- File-level validations: linked to a file template
- Table-level and row-level validations: linked to a table template
- Colum-level and field-level validations: linked to a column template

In paragraphs 4.2-4.5 the different validation types are described per validation level. Different conditions (the specification of the rule) can be applied per validation type. The conditions are saved in CSV format in the database used by the Artena Data Validator: “aaa”,”bbb”

4.2 Table-level validations

1. RowsRange

This validation focusses on checking the number of rows present in table. The number of rows in the table is compared to a given range.

Possible Condition	Description	Notation "operator","limit1","limit2"
1	The number of rows should be greater than 2000	">","2000"
2	The number of rows should be greater than or equal to 2000	">=","2000"
3	The number of rows should be smaller than 2000	"<","2000"
4	The number of rows should be smaller than or equal to 2000	"<=","2000"
5	The number of rows should be between 1000 and 2000	"<>","1000","2000"

4.3 Row-level validations

1. FieldCount

This validation checks if the number of field (or columns) in a single row are equal the desired amount. The desired amount can be set as a condition or is checked automatically based on the number of column templates which apply on the row.

Possible Condition	Description	Notation (optional) "number_of_fields"
1	The number of fields in the given row is equal to 10	"10"
2	The number of fields in the given row is equal to the column templates who apply to the row	

4.4 Column-level validations

1. FieldsRange

This validation checks the number of filled fields in a single column. Each field holding data (of whatever kind) is included in a sum. If the sum does not match the given range the validation will generate a result.

Possible Condition	Description	Notation "operator","limit1","limit2"
1	The number of filled fields should be greater than 2000	">","2000"
2	The number of filled fields should be greater than or equal to 2000	">=","2000"

3	The number of filled fields should be smaller than 2000	“<”,”2000”
4	The number of filled fields should be smaller than or equal to 2000	“<=”,”2000”
5	The number of filled fields should be between 1000 and 2000	“<>”,”1000”,”2000”

2. Uniqueness
This validation will check if every field in a column is unique. Every value is allowed once with this validation type. This validation type requires no condition to be set.

4.5 Field-level validations

1. Presence
This validation checks if a single field is filled (= not empty) because its presence is required. This validation also requires no condition to be set.

2. Datatype
This validation checks if the value of the checked field matches the datatype which is set in the corresponding column template. This validation also requires no condition to be set.

3. Format
This validation checks if the value of the checked field matches the given parameters. The check can be executed by giving a character_array or a regular expression. Thus the format validator supports multiple format methods.

Possible Condition	Description	Notation “format_method”,”parameters”
1	The value of the field has to be ‘M’ or ‘V’	“character_array”,”M,V”
2	A date in the field can only has to be of the format: dd/mm/yyyy	“regular_expression”, ”(0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])[- /.](19|20)”

4. Range
This validation type checks if the value in the field matches the given range. The possible conditions for this validation type are the same as the *RowsRange* and *FieldsRange*.

Possible Condition	Description	Notation “operator”,”limit1”,”limit2”

1	The value of the field should be greater than 2000	“>”,”2000”
2	The value of the field should be greater than or equal to 2000	“>=”,”2000”
3	The value of the field should be smaller than 2000	“<”,”2000”
4	The value of the field should be smaller than or equal to 2000	“<=”,”2000”
5	The value of the field should be between 1000 and 2000	“<>”,”1000”,”2000”

5. Lookup

This validation checks if the value of the checked field matches formats/values defined in a specific lookup table. These lookup tables are used by other systems of Artena and define the required format or value for certain fields. In order to execute these validations the condition needs to hold the info needed to retrieve these lookup values.

Possible Condition	**Description**	**Notation** **“connectionstring”,”scheme”, ”table”,”column”**
1	The value of the checked field needs to be equal to the values defined in the lookup table. The lookup table is located in the database defined by the connection string	“*ConnectionString*”,” *database scheme*”,”*lookup table*”,”*lookup column”*

6. Relationship

This validation checks a relationship between two or more fields. The relationship validation currently checks one of the following:

- ‘If field 1 has a certain value then field 2 (or more) need to be present.’
- ‘Value of field 1 has to be present in column 2 (in table #)’

Possible Condition	**Description**	**Notation** **”relationship_type”, ”variable1”,”variable2”**
1	If the value of the checked field equals ‘married’ then the fields ‘partner_name’ en ‘partner_gender’ have to be filled.	”rel_samerow”, married”, ”column N”,”column N+1 ”