

2009

I6dt Groep 1

Frank Aben (07072031)

Evert Pennings (07072104)

ADVENTURE WORKS CYCLES
[ELABORATION DOCUMENT]

Versie 1.0 – 12 april 2009

Afdeling:	Id6-dt groep 1
Document naam:	Elaboration document project I6
Document versie:	1.0
Document status:	Wacht op goedkeuring
Uitgiftedatum:	12-4-2009
Opgesteld door:	I-6 groep 1
Gewijzigd door:	I-6 groep 1
Sjabloonversie:	Dit document is geschreven op basis van lesmateriaal vak I-6
Distributie:	

Wijzigingshistorie

Versie	Datum	Auteur	Distributie	Opmerkingen
0.1	19-3-2009	F. aben	Groep 1	Initiele versie
0.2	24-3-2009	F.aben	Groep 1	Toevoeging diagrammen
0.3	27-3-2009	F.aben	Groep 1	Vaststellen MoSCoW lijst
0.4	30-3-2009	F.aben	Groep 1	Uitwerken chokepoints
0.5	2-4-2009	F.aben	Groep 1	Herziene opzet rapport. Uitwerking infrastructuur
0.6	7-4-2009	F.aben	Groep 1	Uitwerking programma structuur en beveiliging
0.7	12-4-2009	F.Aben	Groep 1	Toevoeging UML en Sequentie diagrammen
0.8	12-4-2009	F.Aben	Groep 1	Finishing touches
1.0	12-4-2009	F.Aben	Groep 1	Release versie

Inhoudsopgave

1.	Inleiding.....	5
2.	Requirements	5
2.1	requirements per stakeholder	5
	Opdrachtgever.....	5
	Afdeling Operations	5
	Afdeling Development	5
	World Wide Importers	6
2.2	MoSCoW lijst	6
	MUST.....	6
	SHOULD	6
	COULD	6
	WISH	6
3.	Voorgestelde architectuur	7
3.1	Lokale server.....	7
3.2	Synchronisatie databases.....	9
3.3	Updaten applicaties	10
3.4	Beveiliging.....	11
3.5	Uniforme 3-lagen structuur	12
3.6	Templates	16
4.	Sequence modellen	19
4.1	sequentie model #1: synchroniseren databases	19
4.2	Sequence model #2: updaten applicaties locale server(s).....	20

1. Inleiding

Het doel van dit document is een technische vertaling te geven van de bevindingen en afspraken zoals deze zijn opgenomen in het inception report. De bedoeling van dit rapport is een inzicht te geven van de opties die zijn overwogen en de keuzes die uiteindelijk zijn gemaakt om tot de architectuur te komen zoals deze beschreven staat in het AD.

2. Requirements

2.1 requirements per stakeholder

Opdrachtgever

- De nieuwe architectuur moet de huidige (stand-alone) systemen van iedere afdeling integreren. Hiervoor zullen bottom-up gemeenschappelijke componenten worden ontwikkeld (bijv. voor communicatie met de database, authenticatie, autorisatie). Topdown wordt er nog wel gebruik gemaakt van modules per afdeling, maar deze worden ontwikkeld volgens een uniforme architectuur.
- De nieuwe architectuur moet kostenbesparend zijn. Om dit te realiseren zal voornamelijk gebruik gemaakt worden van de bestaande infrastructuur, hardware en software-licenties die Adventure Works Cycles momenteel bezit.

Afdeling Operations

- Het onderhoud van de nieuwe architectuur moet minder tijd gaan kosten. Dit wordt als volgt gerealiseerd:
 - Er wordt een universeel update-systeem ontwikkeld, zodat het deployen van componenten eenvoudiger wordt.
 - De logging van applicaties en diensten binnen de architectuur wordt eenduidig (alle componenten loggen op dezelfde manier), zodat eventuele problemen makkelijker te detecteren zijn.

Afdeling Development

- Het ontwikkelen van nieuwe modules voor het systeem moet minder tijd kosten. Dit doel zal als volgt worden gerealiseerd:
 - Er wordt gebruik gemaakt van templates (bijv. Interfaces of Factory Classes), zodat nieuwe componenten snel kunnen worden ontwikkeld.
 - Er worden protocollen vastgelegd voor de communicatie tussen de onderlinge componenten, zodat de ontwikkelaar precies weet hoe bestaande componenten aangeroepen dienen te worden.
 - Componenten worden ontwikkeld in C#. De afdeling Development heeft hier ervaring mee, en zal dus sneller kunnen ontwikkelen in C# dan in talen die voor hen onbekend zijn.

World Wide Importers

- Het systeem moet 24-7 beschikbaar zijn. Om dit te bereiken worden diensten zover de huidige infrastructuur en hardware dit toelaat redundant uitgevoerd.
- Het systeem moet ondersteuning bieden voor meerdere talen en monetaire eenheden.
- Bestelinformatie mag niet te groot zijn (wegens bandbreedte). Om het dataverkeer tussen het hoofdkantoor en WWI zo laag mogelijk te houden zal WWI de beschikking krijgen over een lokale subset van de database.

2.2 MoSCoW lijst

Vanuit de requirements en de analyse van de huidige situatie is de volgende lijst met items gegenereerd. Deze items zijn gegroepeerd volgens het MoSCoW principe, wat staat voor Must, Should, Could, Wish. Een MoSCoW lijst geeft in basale lijnen de prioriteit binnen het project van de verschillende items weer. Voor dit project heeft dit de volgende verdeling tot gevolg:

MUST

- De bestaande infrastructuur moet zoveel mogelijk intact gelaten worden
- Binnen de architectuur moet voor de applicaties een 3-lagen structuur worden afgedwongen
- De architectuur moet voorzien in een ondersteuning van een interface op meerdere platformen.
- De data overdracht moet hetzelfde zijn ongeacht het platform dat de data benaderd
- De middleware moet uit te breiden zijn naar andere/nieuwe platformen

SHOULD

- Communicatie via het internet van (privacy) gevoelige data moet beveiligd zijn via HTTPS of SSL tunnel.
- De applicaties moeten voldoen aan de vanuit de architectuur voorgeschreven template structuur
- Het netwerk maakt gebruik van lokale databases met verschillende subsets. Dit met de bedoeling de beveiliging te vergroten, toegang tot de data te verbeteren en downtime te verkleinen.
- Voor communicatie tussen de componenten wordt SOAP gebruikt

COULD

<Geen items werden gevonden die binnen deze rubriek zouden passen>

WISH

- middleware moet zijn data flow kunnen up/downscalen al naar gelang de lijn

3. Voorgestelde architectuur

Bij het opstellen van de voorgestelde architectuur is zoveel mogelijk geprobeert alle requirements op een zo elegant mogelijke manier te combineren. Dit leidde ertoe dat wij al snel tot de conclusie kwamen dat aanpassingen in de applicaties alleen niet voldoende zouden zijn, maar dat aanpassingen in het serverlandschap nodig zouden zijn om zo veel mogelijk van de requirements te kunnen dekken. Echter werd vooropgesteld dat de huidige infrastructuur zoveel mogelijk intact gelaten zou moeten worden.

Als we naar de infrastructuur kijken willen we de volgende aanpassingen voorstellen:

- 1) Invoering van een lokale server in Mexico en lokale databases met hetzij subsets (vb. De webshop en resellers) hetzij de volledige data (vb. sales, thuiswerkers) aanwezig in de main database.
- 2) Synchronisatie en communicatie tussen de lokale databases en de main database via een synchronisatieserver waarbij de communicatie verloopt via een batchprotocol.
- 3) Update server die aanpassingen in de applicaties distribueert naar alle off-site lokaties.
- 4) Beveiliging van de data via de invoering van rollen en de ondersteuning van HTTPS/SSL aan de ene kant en het compartmentaliseren van de data aan de andere kant.

Waar het gaat om de applicaties zelf komen de volgende punten om de hoek kijken:

- 5) Invoering van een uniforme 3-lagen structuur waarbij de data overdracht tussen de client en de server een uniform karakter heeft. Hierbij zullen meerdere interfaces op meerdere platformen kunnen communiceren met dezelfde businesslogic.
- 6) Opbouw van de applicatie aan de hand van een template structuur die aanpassingen van functionaliteiten en business logic gemakkelijker moet maken.

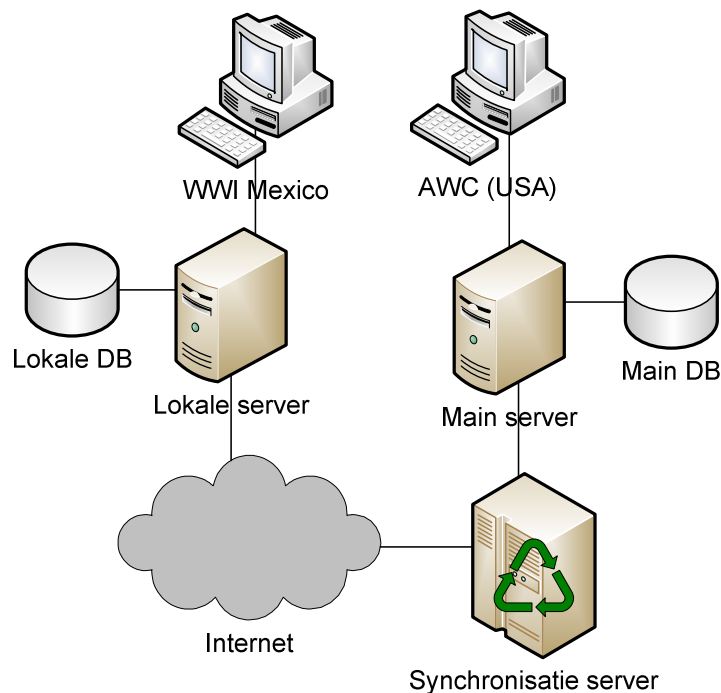
Deze punten zijn als volgt uitgewerkt.

3.1 Lokale server

Een van de grote struikelblokken bij opstellen van deze architectuur was de situatie in Mexico. Er werd aangegeven dat de bandbreedte niet genoeg was om een remote application oplossing mogelijk te maken. Zeker gezien het kostenaspect een rol speelt in dit project wilden we niet voorstellen de huidige bandbreedte op te waarderen, zeker gezien we onzeker zijn of deze stap uiteindelijk zichzelf zal terugverdienen.

Om het bandbreedte probleem op te lossen zijn we uitgegaan van een infrastructuur tussen de Verenigde Staten en Mexico die gebaseerd is op een overdracht van data in batches waardoor de bandbreedte slechts op gezette tijden belast wordt. Om dit systeem optimaal te laten werken introduceren we in Mexico een lokale server.

Deze lokale server is een server waarop een applicatielandschap draait dat spiegelt aan dat van AWC in de Verenigde Staten. Daar bovenop wordt een kopie van de gehele AWC database of een subset van deze database eveneens lokaal op deze server geplaatst. Al naar gelang de behoeftes in Mexico wordt het mogelijk om het applicatielandschap en/of de subsets van de database uit te breiden.



Voorbeeld lokale server architectuur

Het voordeel van deze opzet is dat zelfs als het internet langdurig plat komt te liggen de batchfile via een koeriersdienst op en neer gestuurd kan worden (de zgn. FedEx noodmaatregel) of als dit niet mogelijk blijkt te zijn dat elk systeem onafhankelijk van elkaar kan blijven opereren.

De nadelen die aan dit systeem kleven zijn dat er op gezette tijden down-time ingecalculeerd dient te worden om de synchronisaties en updates te laten lopen. Dit kan voor het grootste gedeelte worden afgevangen door de batchmomenten te laten vallen buiten kantooruren en op momenten dat er normaal andere batchverwerkingen en/of backup routines ingeplend zijn (een synchronisatie routine gaat in deze goed hand in hand met een backup routine).

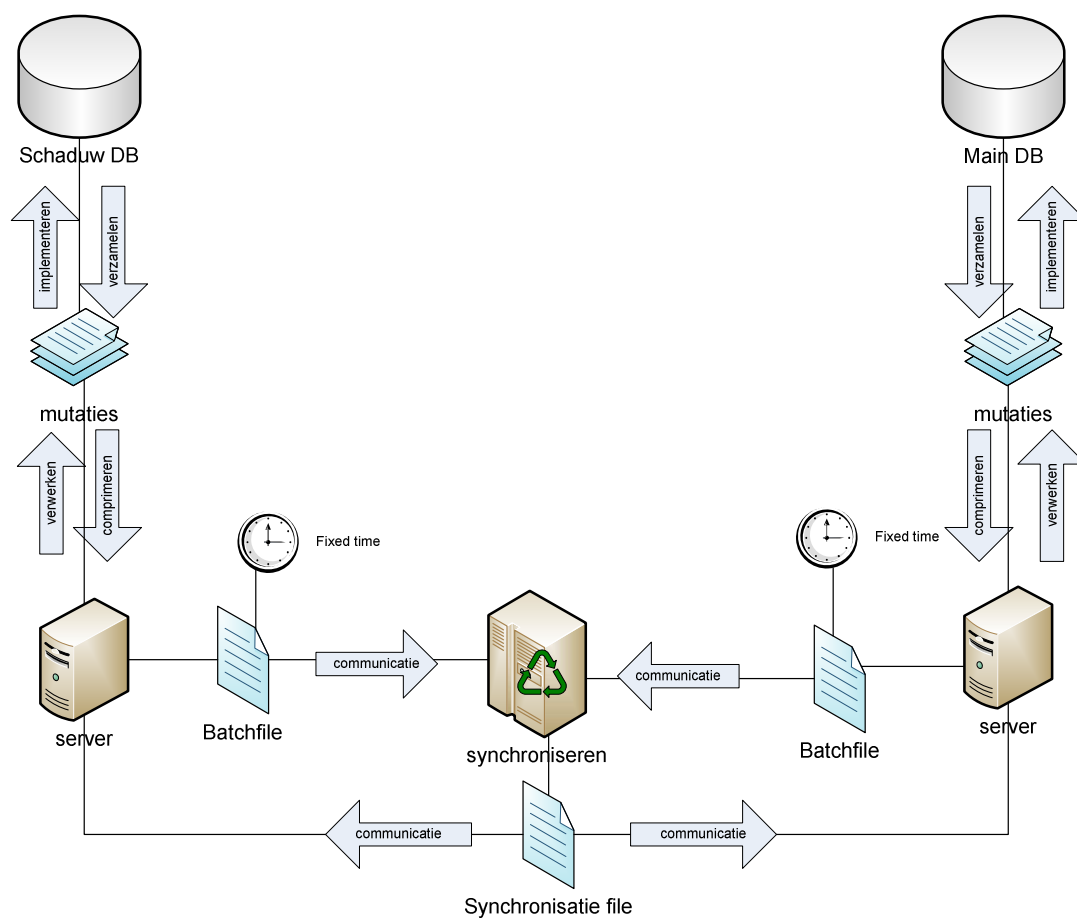
Het andere nadeel is dat dit systeem nooit real-time alle data kan laten zien die in het systeem aanwezig is, maar dat er altijd een vertraging in de data zit die even groot is als de synchronisatie interval die is ingesteld. Deze interval kan verkleind worden waardoor data snelle overall toegankelijk is, maar dit betekent ook een toename van het aantal update momenten die het systeem kunnen vertragen.

3.2 Synchronisatie databases

Echter alleen het invoeren van een lokale server zou niet genoeg zijn om dit probleem op te lossen. Het enige wat het doet is twee systemen creëren die (bijna) gelijk zijn, maar die niet met elkaar communiceren.

Om dit probleem op te lossen willen we gebruik maken van een systeem waarbij data uitwisseling wordt bewaard en opgespaard om vervolgens in een burst-communicatie wordt overgezet.

Om dit principe goed te laten werken zie je 2 concepten naar boven drijven die binnen de ICT veelvuldig gebruikt worden, nml. Compressie en batchverwerking.



Voorbeeld database synchronisatie

Om te beginnen wordt er een interval afgesproken waarop data verzameld en verzonden wordt. Dit kan naar gelang de behoefte voor actuele data verhoogd of verlaagd worden. Binnen deze tijd worden alle mutaties op de database gelogd.

Om te voorkomen dat door tijdsverschillen de synchronisatie verkeerd kan lopen (vb. Verschillen per land in overschakelen van zomer naar winter tijd) zal het synchronisatie routine worden opgestart door de synchronisatie server.

Op het moment dat de synchronisatie verwerking wordt opgestart worden alle mutaties verzameld en gecomprimeerd in een batchfile. Deze batchfile wordt vervolgens opgestuurd naar een synchronisatie server die alle batchfiles binnenkrijgt.

Na het verzamelen van alle batchfiles zal deze synchronisatie server aan de hand van vooraf gedefinieerde regels een totaalijst maken van alle mutaties en waar nodig verschillende mutaties met elkaar verrekenen. Uit deze synchronisatiestap rolt een synchronisatiefile die vervolgens aan alle servers wordt aangeboden.

Zodra deze synchronisatiefile binnen is zal de server deze file vergelijken met de batchfile die is opgestuurd en waar nodig de aanpassingen verwerken in de database.

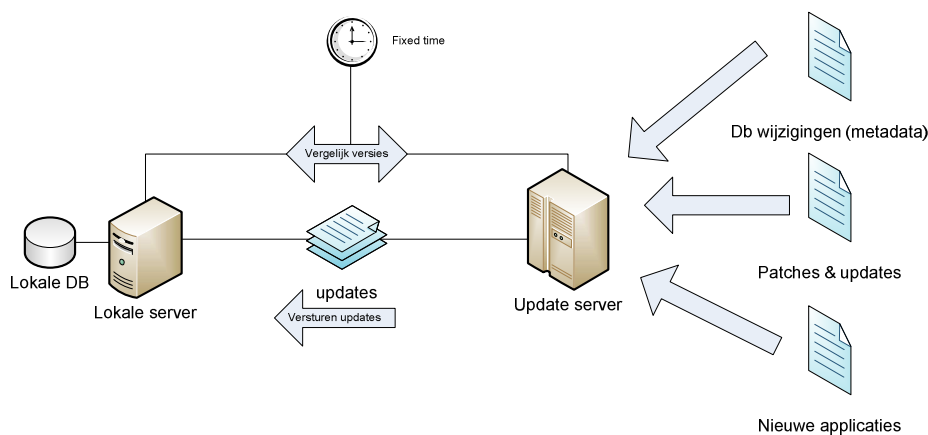
3.3 Updaten applicaties

Om ervoor te zorgen dat iedereen met dezelfde applicaties blijft werken kan een probleem opleveren als het systeem verschillende remote servers bevat. Om dit probleem op te lossen willen we een methode gebruiken die op dit moment veel gebruikt wordt bij oa. Microsoft en veel games: de update server.

Als we kijken naar de voorgestelde architectuur zien we dat delen van het totale systeem alleen via het internet te bereiken zullen zijn. Bij het uitrollen van applicaties kan het dus zijn dat niet alle onderdelen van het gehele netwerk aanwezig zijn tijdens de uitrol. Om ervoor te zorgen dat iedereen met dezelfde applicaties aan de slag gaat worden aanpassingen en updates klaargezet op een update server waarmee systemen op vaste tijden of punten (dit kan een bepaald tijdstip zijn, of tijdens het opstarten/afsluiten.) toegang hebben tot deze updates.

In tegenstelling tot de databases is synchronisatie hier minder een issue en zal het verzoek tot update vanuit de lokale server gerunt worden.

In zo'n geval zal de lokale server contact zoeken met de update server en daar het versielog vergelijken met het versielog dat lokaal aanwezig is. Mocht hier een verschil inzitten, dan zal de lokale server een update procedure in gang zetten, waarbij de laatste updates van de update server worden gehaald. Na het overhalen van deze updates zullen deze op de lokale server worden geïnstalleerd, waarna de verbinding met de lokale server wordt verbroken.

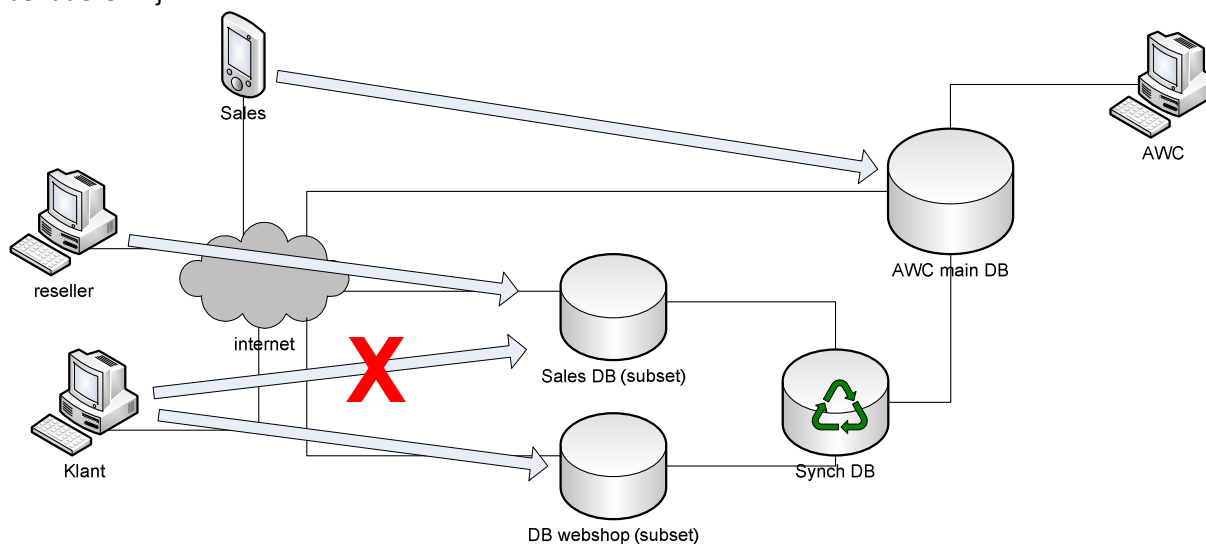


Voorbeeld update server mechanisme

3.4 Beveiliging

Qua beveiliging willen we in de architectuur gebruik maken van 2 methodes, nml. Data-compartmentalisatie en Secure connections.

Zoals iedereen die in ICT werkt weet, er is geen systeem zo veilig dat niemand het kan hacken. Gezien meerdere stake holders hebben aangegeven dat bepaalde gegevens absoluut niet in verkeerde handen mogen vallen leek het ons de beste optie om bepaalde data te compartmentaliseren, dwz. De data te verdelen in aparte, redundante subsets die apart van de main database in de Verenigde Staten te benaderen zijn.



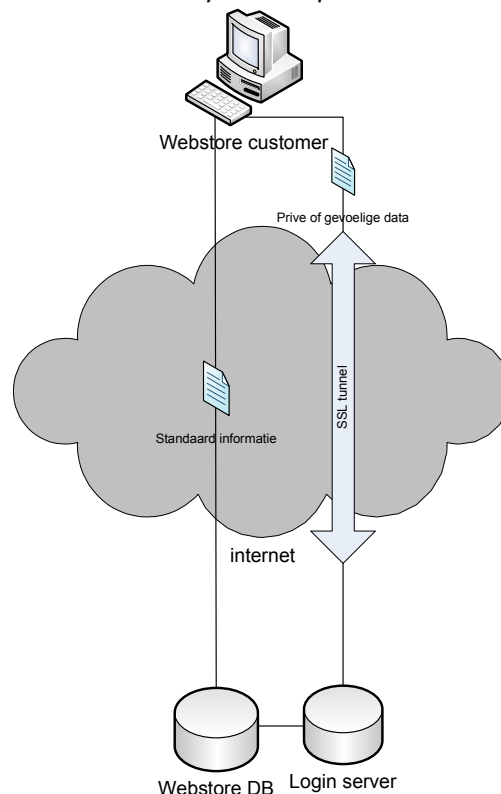
Voorbeeld data compartmentalisatie

Doordat elke gebruiker slechts die subset kan benaderen die ook daadwerkelijk voor hem van belang is kan voorkomen worden dat iemand via conventionele wegen aan informatie kan komen die afgeschermd hoort te zijn.

Om ervoor te zorgen dat alle subsets overeenkomende data hebben maken we wederom gebruik van de synchronisatieserver waar alle mutaties met elkaar worden verrekend en worden uitgeleverd aan de verschillende subsets.

Het scheiden van verschillende datasets zal niet alle beveiligings issues afvangen . Daarom zal daarnaast het principe van beveiligde verbindingen via SSL worden toegevoegd. Dit zal in eerste instantie alleen gelden voor privegegevens, betalings transacties en gevoelige data, maar kan worden uitgebreid naar meerdere vormen van data mocht daarom gevraagd worden.

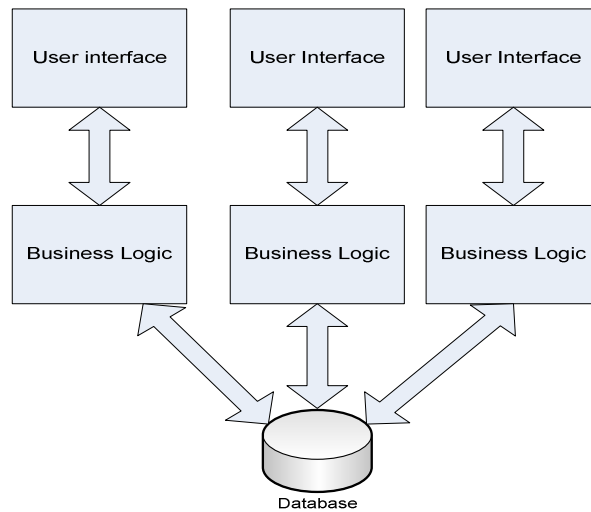
Het principe van een SSL tunnel gaat gepaard met een authenticatie server waarop login gegevens worden bijgehouden en gelogged. Pas nadat iemand zich heeft geïdentificeerd via deze authenticatie server krijgt de persoon toegang tot de additionele waar die gebruiker recht op heeft (bv. Zijn persoonsgegevens, zijn betalings overzicht etc.) Reguliere data aanvragen zoals product informatie en afbeeldingen van producten zullen niet via dit systeem lopen.



Voorbeeld SSL tunnel

3.5 Uniforme 3-lagen structuur

De structuur van de applicaties zoals deze op dit moment bij AWC aanwezig zijn, hebben zoals al in het inception rapport is aangegeven de neiging om op zich te staan waarbij niet is gekeken naar hoe deze applicaties onderling met elkaar zouden moeten/kunnen communiceren. De huidige situatie ziet er ongeveer zo uit:

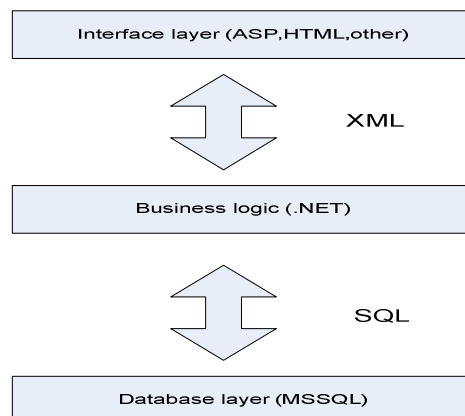


Voorstelling huidige situatie AWC

Om ervoor te zorgen dat het gehele systeem compacter en gemakkelijker te onderhouden wordt stellen we de volgende dingen voor.

Het basis principe van de nieuwe architectuur wordt dat zoveel mogelijk componenten van de verschillende applicaties herbruikt moeten worden en de communicatie tussen de user interface en de business logic in alle gevallen hetzelfde protocol moet volgen. Hetzelfde geldt voor de communicatie tussen de business logic en de database.

Om dit te bereiken stellen we de invoering van een uniforme drie lagen structuur voor die er ongeveer zo uit komt te zien:



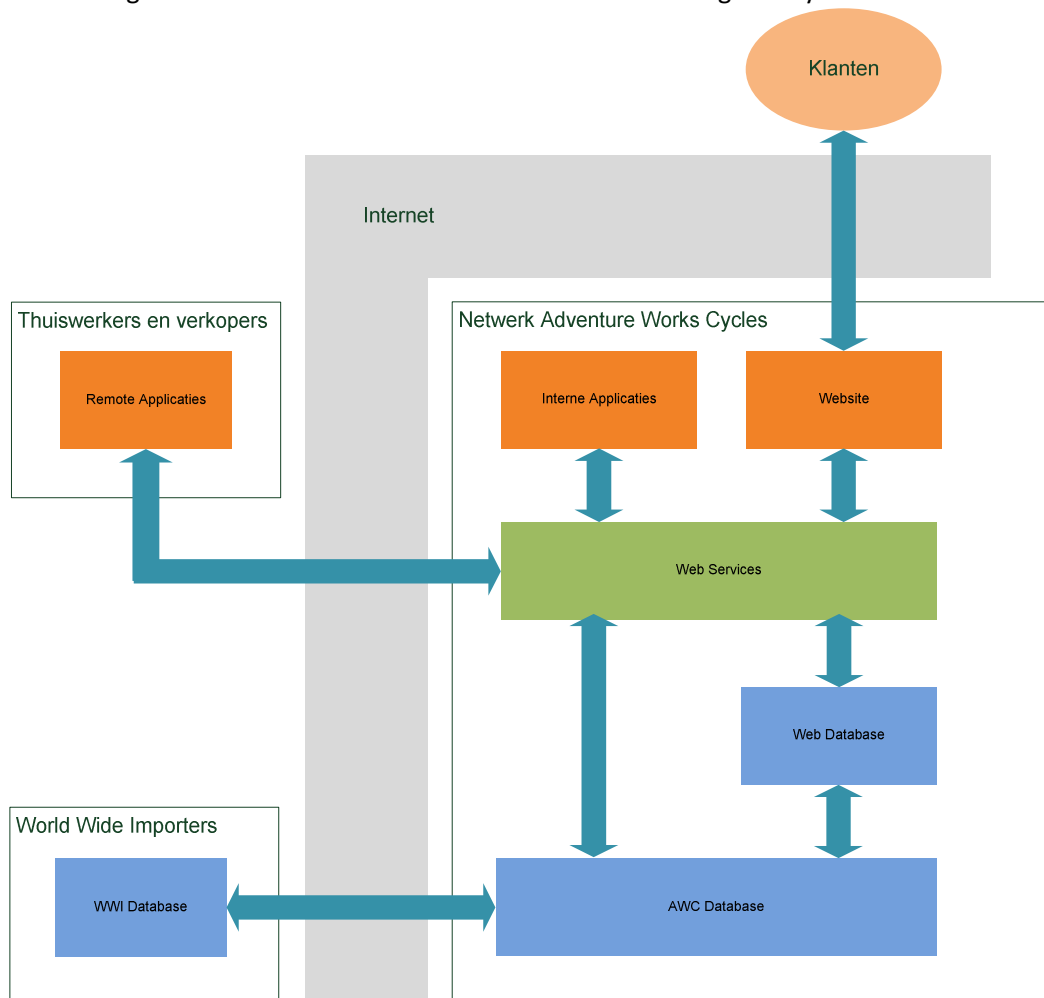
Schema voorgestelde 3 lagen structuur

Het voordeel van de voorgestelde architectuur zal zijn dat ongeacht de user interface de benadering, verwerking en opslag van data ten alle tijden hetzelfde zal verlopen via vast gestelde uniforme protocollen. Hierdoor zal bijvoorbeeld een PDA applicatie dezelfde input leveren als een windows applicatie met dezelfde functionaliteit, waardoor dingen zoals diacrieten, wat soms een probleem kan opleveren als men in meerdere talen werkt altijd op dezelfde manier zullen worden omschreven en opgeslagen.

Na alle opties bekeken te hebben is er besloten deze drie lagen structuur uit te voeren in een web services architectuur.

De Web service architectuur kan omschreven worden als een architectuur gebaseerd op een interface die is opgebouwd uit componenten die toegankelijk is via standaard webprotocollen en waarbij wordt gecommuniceerd via o.a. SOAP. Deze architectuur maakt het mogelijk om op afstand een dienst op te vragen.

In de voorgestelde architectuur zullen de web services als volgt het systeem aan elkaar knopen:



Globale representatie nieuwe architectuur

Netwerk Adventure Works Cycles

Het deel van het systeem dat binnen het bedrijfsnetwerk van Adventure Works Cycles in de VS. opereert.

AWC Database

De centrale database(s) van Adventure Works Cycles. Hierin staat informatie over producten, voorraden, verkopen, inkopen, management informatie, enz. In de database wordt tevens bepaald wie er toegang heeft tot bepaalde data. De database wordt tevens gebruikt voor *authenticatie* en *authorisatie* van gebruiker die via een Web Service een taak willen uitvoeren.

Web Database

Een database die enkel data bevat die van belang is voor bezoekers van de website. Deze database wordt regelmatig gesynchroniseerd met de AWC Database. De database wordt ook gebruikt voor de *authenticatie* en *authorisatie* van bezoeker van de website.

Website

De website bevat een webshop en geeft ondersteuning voor producten. De website kan data opvragen en muteren in de Web Database.

Web Services

Een Web Service is een component dat een verzoek van andere webservices of applicaties kan uitvoeren via het interne netwerk of via Internet. Een webservice kan indien nodig informatie opvragen en muteren in de AWC Database. Zodra een Web Service een verzoek ontvangt, voert het een taak uit en stuurt het resultaat terug naar de afzender, mits deze gemachtigd is de taak uit te voeren.

Interne Applicaties

Applicaties ter ondersteuning van de bedrijfsprocessen binnen Adventure Works Cycles. Dit zijn de applicaties die door de interne medewerkers gebruikt worden. De applicatie kan voor het uitvoeren van bepaalde taken een verzoek naar een Web Service sturen, en het antwoord op een verzoek verwerken. Een applicatie kan met meerdere Web Services communiceren.

World Wide Importers

Het deel van het systeem dat bij World Wide Importers in Mexico opereert.

WWI Database

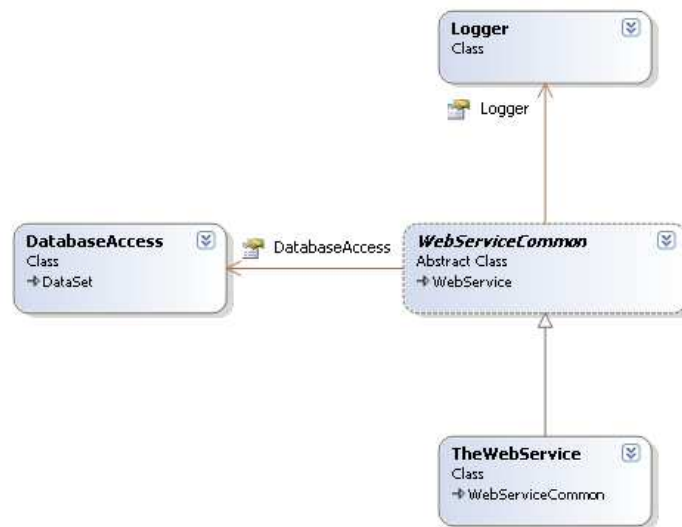
Een database die enkel data bevat over het bestellen en afleveren van onderdelen voor Adventure Works Cycles. Deze database wordt via internet gesynchroniseerd met de AWC Database.

Thuiswerkers en verkopers

Het deel van het systeem dat toegankelijk is voor Thuiswerkers en verkopers.

Remote Applicaties

Remote Applicaties zijn gelijk aan de Interne Applicaties, met als verschil dat ze met de Web Services kunnen communiceren via het Internet.



UML webservice

3.6 Templates

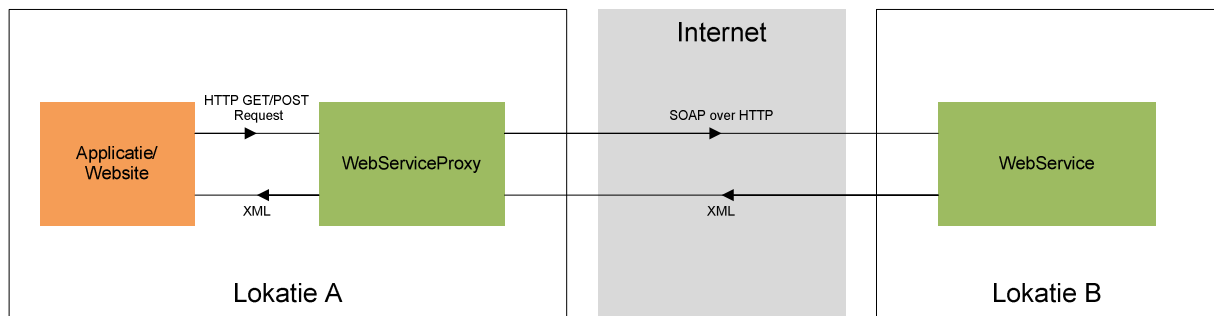
Zoals al in de vorige paragraaf is aangegeven is de web services architectuur gestoelt op een opbouw uit componenten. Gezien we een standaard in deze componenten nastreven is het de bedoeling dat deze componenten gegenereerd gaan worden aan de hand van vooraf gedefinieerde templates.

Deze templates zijn onder te verdelen in 2 formaten, nml. Functionality templates, waarin standaard acties afgevangen worden zoals CRUD functionaliteit, maar ook views e.d. en Business rule templates waarin de regelset waaraan bepaalde functionaliteiten zich moeten houden zijn gedefinieerd.

Om het te vertalen in een manier die iedereen begrijpt, wordt het systeem opgedeeld in stukjes die als legosteentjes bij elkaar gebracht kunnen worden tot een bouwwerk wat tot tand komt aan de hand van de requests van de gebruiker.

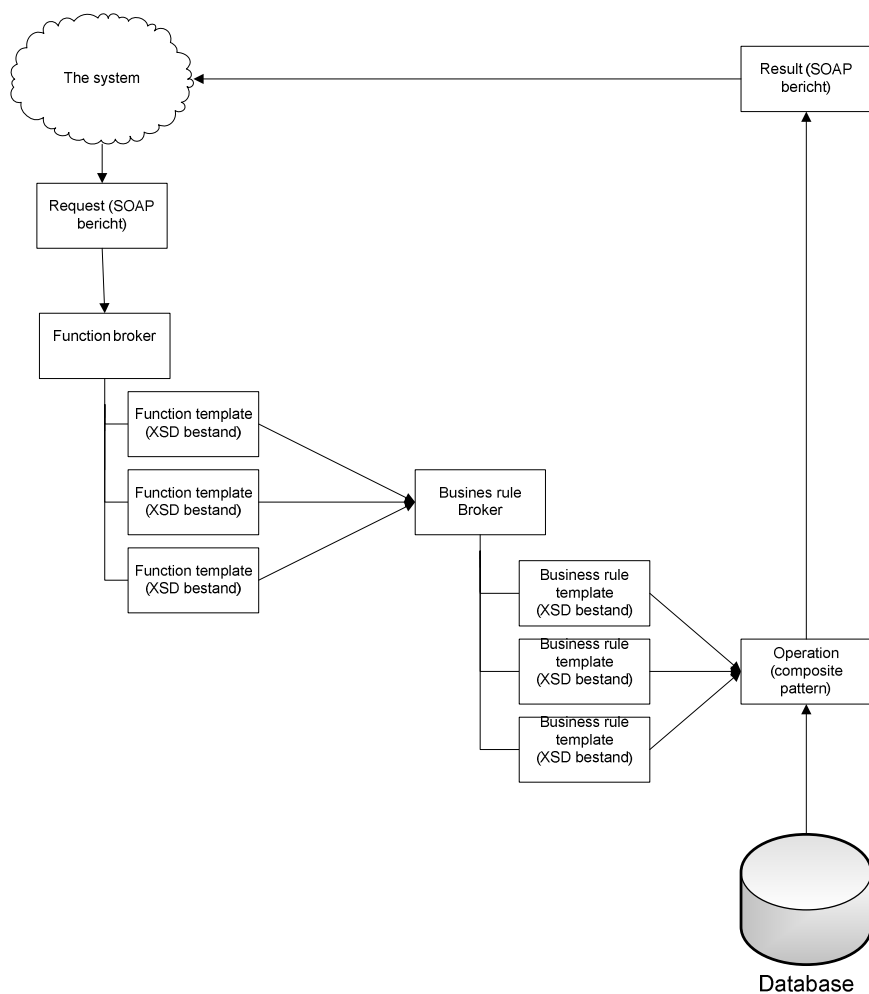
Dit gaat als volgt te werk:

Een gebruiker benaderd via zijn user interface een bepaalde functionaliteit die volgens het systeem mogelijk is. In het systeem zal de function broker dit verzoek interpreteren door de juiste function templates aan te roepen.



Communicatie via de webservice

Zodra deze templates worden aangeroepen zullen deze een verzoek indienen bij de Business rule broker voor de benodigde business rules die met deze functionaliteiten samenhangen. De businessrule broker zal deze templates vervolgens activeren voor gebruik met de functionaliteiten waarna de gebruiker de functionaliteit via zijn interface kan benaderen.



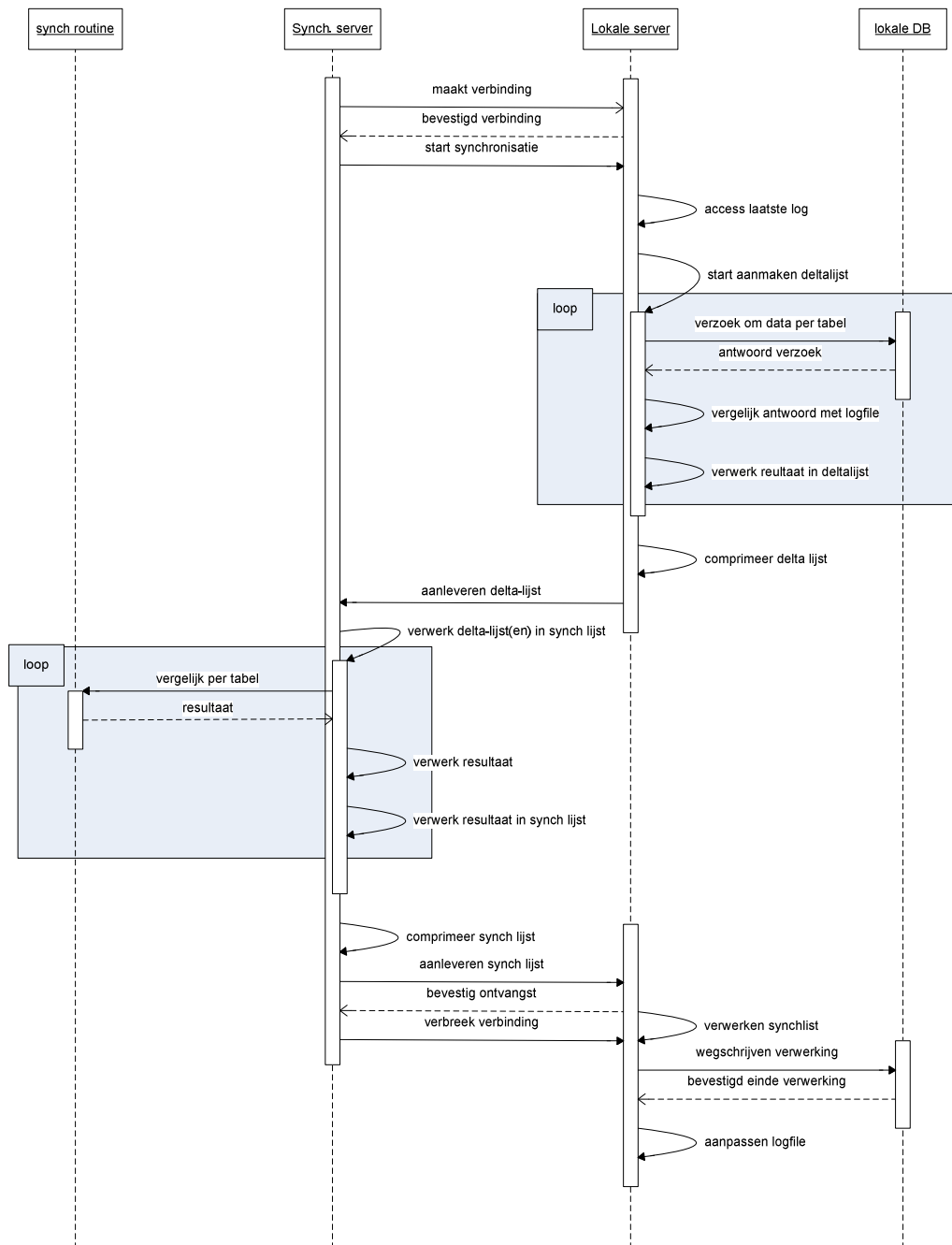
Verloop opbouw componenten via web services

Het voordeel van dit systeem is dat functionaliteiten zowel als business rules meerdere malen door verschillende onderdelen van het systeem gebruikt kunnen worden zonder dat meerdere programma's geschreven hoeven te worden die dezelfde functionaliteiten hebben.

Dit is ook in onderhoudstechnisch opzicht een groot voordeel gezien aanpassingen aan functionaliteiten en business rules slechts een keer aangepast hoeven te worden.

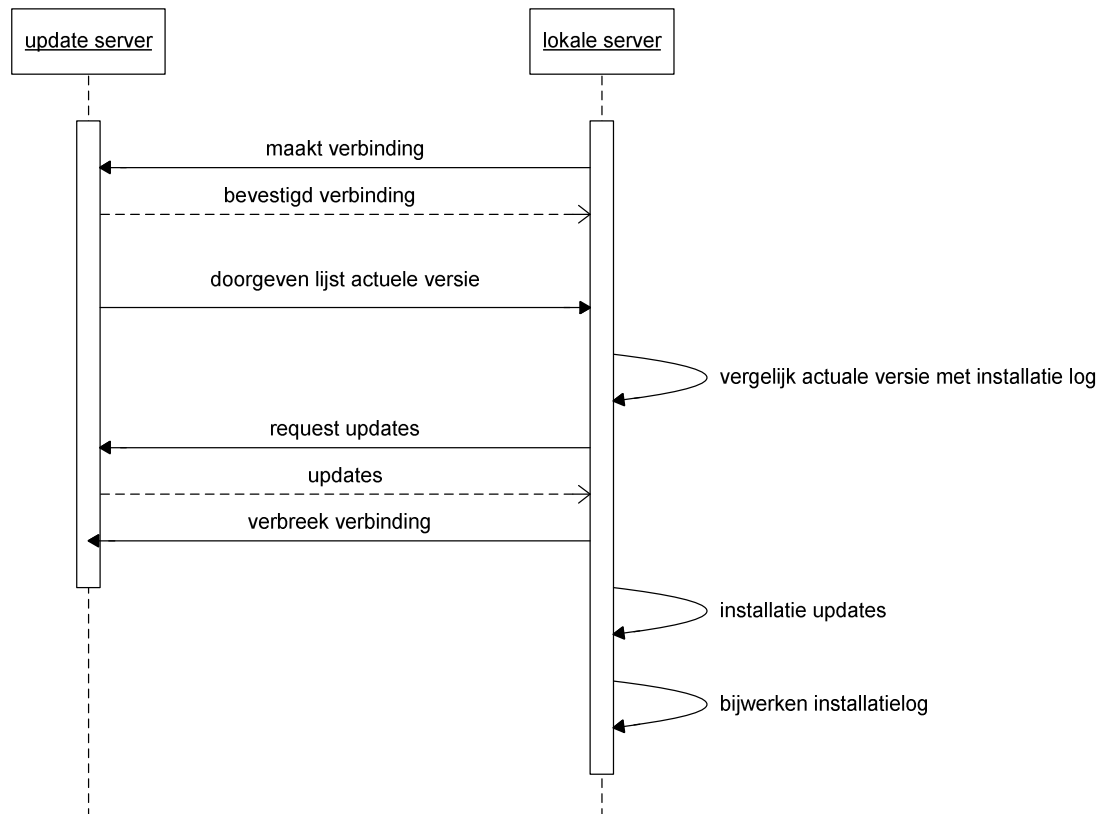
4. Sequence modellen

4.1 sequentie model #1: synchroniseren databases



Sequentie diagram data synchronisatie

4.2 Sequence model #2: updaten applicaties locale server(s)



Sequentie diagram applicatie update