Integrering av kassasystem och nätbutik

DdD Retail - Magento 2

Mathias Andtbacka

Examensarbete
Informations- och medieteknik
22 december 2016

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	5837
Författare:	Mathias Andtbacka
Arbetets namn:	Integrering av kassasystem och nätbutik
Handledare (Arcada):	Magnus Westerlund
Uppdragsgivare:	HÄT Systems OY / Kari Lehtonen

Sammandrag:
Syftet med arbetet är att utreda vilka möjligheter Magento 2 ramverket ger för att implementera kommunikation med ett externt kassasystem. Arbetet går igenom vilken data som finns i de olika systemen samt vilken data som skickas mellan systemen. Sedan beskrivs uppbyggnaden av tilläggsmoduler för Magento. Slutligen beskrivs implementationen av en tilläggsmodul samt de olika delarnas ansvarsområden.

Nyckelord:	DdD Retail Magento Modul Integrering ERP SOAP
Sidantal:	29
Språk:	Svenska
Datum för godkännande:	22 december 2016

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Informaatio- ja mediateknologia
Tunnistenumero:	5837
Tekijä:	Mathias Andtbacka
Työn nimi:	Kassajärjestelmän ja nettikaupan integraatio
Työn ohjaaja (Arcada):	Magnus Westerlund
Toimeksiantaja:	HÄT Systems OY / Kari Lehtonen

Tiivistelmä:

Työn tarkoitus on selvitellä millä tavalla Magento 2 ohjelmisto voi liittää ulkoiseen kassajärjestelmään. Työ käy ensin läpi minkälainen data järjestelmistä löytyy ja millä tavalla tämä data voi kytkeä yhteen. Sen jälkeen selvitetään millä tavalla Magenton lisäosat ovat rakennettuja. Lopuksi käydään läpi lisäosa kokonaisuudessaan ja miten järjestelmien tarjoama data on liitetty yhteen.

Avainsanat:	DdD Retail Magento Moduuli Integraatio ERP SOAP
Sivumäärä:	29
Kieli:	Ruotsi
Hyväksymispäivämäärä:	22. joulukuuta 2016

DEGREE THESIS	
Arcada	
Degree Programme:	Information- and mediatechnology
Identification number:	5837
Author:	Mathias Andtbacka
Title:	Integrating a cloud based Point-of-Sales system with a webshop
Supervisor (Arcada):	Magnus Westerlund
Commissioned by:	HÄT Systems OY / Kari Lehtonen

Abstract:

The purpose of this thesis is to evaluate the Magento 2 framwork and the possibilities it gives the user to connect to an external point of sales system. The first part discusses the different data values in the two systems and what data will be sent between the systems. Then the cloud based POS and its API are covered in closer detail. After this we discuss the structure and implementation criteria for a Magento 2 extension. Finally the thesis descripes the implementation and reflects on the thesis as a whole.

Keywords:	DdD Retail Magento Module Integration ERP SOAP
Number of pages:	29
Language:	Swedish
Date of acceptance:	December 22, 2016

INNEHÅLL

FIGURER

TABELLER

FÖRKORTNINGAR OCH BEGREPP

API Application Programming Interface

DdD Kassasystemet DdD Retail

ERP Enterprise Resource Planning

PHP PHP: Hypertext Preprocessor

PoS Point of Sales

SOAP Simple Object Access Protocol

WSDL Web Services Description Language

XML Extensible Markup Language

1 INLEDNING

Detta arbete behandlar hur man sammankopplar ett externt kassasystem med nätbutiken Magento 2. Arbetet behandlar specifikt kassasystemet DdD Retail. Syftet är att bygga en tilläggsmodul för Magento som har motsvarande funktionalitet som en befintlig resursplaneringslösning, en sk. ERP lösning. Arbetet kommer att utreda vilka möjligheter Magento ramverket erbjuder för integrering med en utomstående API.

Målet är att skapa en implementation som kan ersätta ett befintligt ERP system genom att endast installera en tilläggsmodul på nätbutiken. Arbetet beskriver en teknisk implementation av denna modul. Beställaren har sedan möjlighet att erbjuda modulen som en del av sin produktportfölj. Modulen kommer att användas av beställarens kund för att länka samman deras nätbutik och kassasystem.

Kassasystemet DdD Retail erbjuder SOAP tjänster som kan utnyttjas för kommunikation mellan systemen. Dessa SOAP tjänster har dock vissa begränsningar som måste tas i beaktande. Modulen kommer att kommunicera med kassasystemet och hämta information om produkter, produktkategorier, leverantörer, lagerstatus, etc. Arbetet kommer närmare att behandla de SOAP metoder som används för att hämta denna information. Modulen kommer att hämta data från kassasystemet med jämna intervaller. Denna data uppdateras sedan i nätbutiken. Nätbutiken kommer i sin tur att skicka försäljningstransaktioner till kassasystemet.

Kundens nuvarande ERP system använder sig inte av SOAP tjänsten utan är direkt kopplat till kassasystemets databas. Tilläggsmodulen kommer däremot att använda sig av SOAP tjänsten. Detta medför vissa begränsningar eftersom SOAP tjänsten ej skickar någon data ifall den ej specifikt efterfrågas. Kunden har ej möjlighet att implementera egna lösningar på kassasystemet och kan således inte påverka denna begränsning. Kunden har däremot full tillgång till sin nätbutik. Detta har lett till valet att implementera en tilläggsmodul som utnyttjar nätbutikens modulära ramverk.

Arbetet kommer kommer att granska följande aspekter närmare.

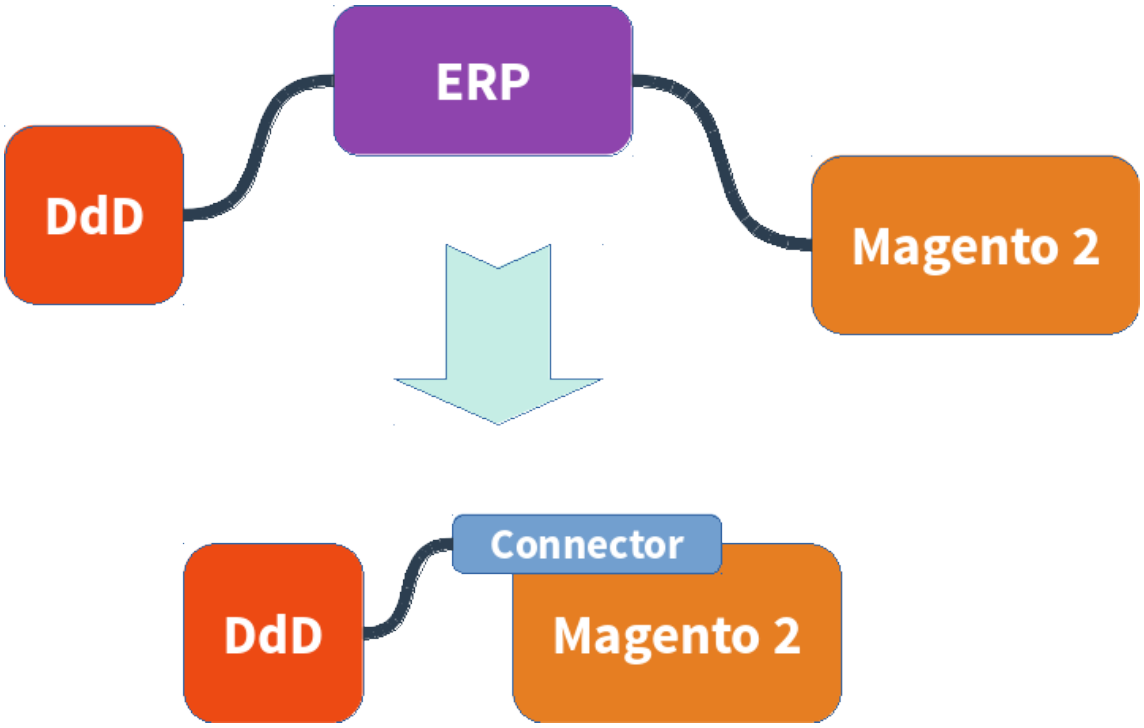
- Datastrukturen i kassasystemet
- Datastrukturen i nätbutiken
- Överföring av data mellan systemen m.h.a. SOAP
- Uppbyggnad av nätbutikens tilläggsmoduler
- Installation av tilläggsmoduler

1.1 Bakgrund

Kundens befintliga ERP lösning utnyttjas ej optimalt. Orsaken är att ERP lösningen är specialiserad för att handska krävande installationer med många försäljningspunkter (eng. Point of Sales, PoS). Dylika installationer återfinns exempelvis i stora dagligvarubutiker med tiotals försäljningspunkter. Kunden har i detta fall endast ett fåtal kassaapparater , det kan röra sig om en till tre stycken. Den befintliga ERP lösningen är således överdimensionerad. Detta leder till en onödigt hög månadsavgift för kunden. Beställaren har därför getts i uppgift att utreda möjligheten att implementera en motsvarande lösning som en del av nätbutiken.

1.2 Syfte och mål

Syftet med detta examensarbete är utreda ansvarsfördelningen mellan kassasystemet och nätbutiken, och på basen av detta göra en implementation som utvidgar funktionaliteten hos Magento i form av en tilläggsmodul. Arbetet granskar strukturen hos såväl Magento 2 ramverket som tilläggsmoduler för detta. Arbetet har som mål att skapa en tilläggsmodul som motsvarar det befintliga ERP systemet till en grad som uppfyller kundens behov.



Figur 1. Planerad implementation

1.3 Avgränsning

Arbetet kommer inte att ta ställning till verktygen som används, dvs. kassasystemet och nätbutiken eftersom dessa specifieras av kunden. Arbetet tar ej heller ställning till metoden som används för överföringen av data eftersom denna bestäms av kassasystemet. Arbetets syfte är ej att ersätta det befintliga ERP systemet för alla kunder. Målet är istället att skapa ett kostnadseffektivt komplement med motsvarande funktionalitet som det befintliga systemet.

1.4 Konventioner

Detta arbete använder sig av följande typografiska konventioner för att särskilja olika syftningar.

- Kod, kommandon, metoder samt variabler med `Mono-Spaced stil`
- Platshållare omges med större än, mindre än på följande vis <text>
- Termer och filnamn med *kursiv stil*

2 KASSASYSTEMET DDD RETAIL

Kassasystemet DdD Retail är en kassalösning som upprätthålls av företaget Data Detail Danmark A/S. Systemet är utvecklat för beklädnadsbranschen men kan även användas för andra typer av dagligvaruhandel. Kassasystem består vanligen av två delar, en point of sales (PoS) som används vid daglig försäljning i butiken, samt en backoffice del som används för lagerhantering och uppföljning av försäljning. DdD har implementerat ett system där backoffice delen erbjuds som en molntjänst. Backofficen är tillgänglig för butiksinnehavaren genom att denne loggar in på en hemsida och kan sköta backoffice rutinerna där. Molntjänsten har även ett gränssnitt som kan utnyttjas för kommunikation med utomstående system. Butiksinnehavaren klarar dock av att sköta de dagliga rutinerna med hjälp av endast nätgränssnittet.

2.1 Data

Kassasystemet sparar en stor mängd data och systemet kan skräddarsys för kunden beroende på dennes önskemål och behov. DdD ställer inga krav på att data som matas in i systemet är unik mellan olika produkter. Istället används ett fortlöpande unikt ID nummer för systemet med benämningen EDB nummer. Detta nummer gör att produkter kan särskiljas även om datan i övrigt är identisk. Identiska produkter förekommer dock inte i praktiken utan de olika produktattributen varierar. Utöver produktkoderna sparas även information om produktkategorier. Det kan i kundens fall röra sig om kategorier som exempelvis ytterplagg, underkläder, tröjor eller skor. Kassasystemet sparar även attributkategorier. Dessa innehåller olika färg- och storlekskombinationer och gör det enklare för slutanvändaren att snabbt skapa olika produktvarianter. Kassasystemet sparar även information om tillverkare och varumärke, samt information om säsong och material.

Systemet är utformat så att all datainmatning sker i den molntjänstbaserade backofficen. När användaren för in ny produktinformation skapas automatiskt en unik EAN kod för varje produkt. EAN koden är en 13 siffrig kod som består av fyra komponenter. GS1 prefix, tillverkarkod, produktkod, samt checksum. EAN koder med prefixen 200-299 är ämnade för internt bruk och det är sådana koder som DdD använder sig av. Orsaken till

detta är att den riktiga EAN koden som finns på produkterna ej nödvändigtvis är känd i det skede som produkten förs in i systemet. EAN koder genereras i stället på basen av produktens EDB nummer med prefixet 299. Kassasystemet och SOAP tjänsterna kräver att EAN koden för varje produkt är unik för att undvika förväxling mellan produkter. EAN koderna används vid kommunikation med SOAP tjänsterna som den primära nyckeln.

Eftersom formatet på EAN koderna är känt kan dessa genereras för att föra in nya produkter till nätbutiken Magento. Denna införing förutsätter att produkten använder sig av den av kassasystemet DdD genererade EAN koden. Det kan förekomma fall där kunden har definierat andra EAN koder än de som genererats av kassasystemet. I detta fall måste produkternas EAN koder importeras skilt till Magentos produktdatabas för att den övriga produktinformationen skall kunna hämtas från APIt.

2.2 Tjänster

DdD erbjuder en SOAP baserad nättjänst för extern hantering av data. Tjänsten har tre delar med olika ansvarsområden. `StockService` används för lagerhantering samt produktinformation. `CustomerService` används för att hantera kunddata. `SaleService` används för hantering av försäljningstransaktioner. (DdDservice 1.4, 2012)

För att minska belastningen på SOAP tjänsten har DdD implementerat en timeout mellan upprepade anrop. Tiden för timeouten varierar från allt från 0 sekunder till 5 minuter beroende på tjänst och metod. Timeout tiden är konstant för de olika ändpunkterna. Tabellerna 1 och 2 redogör för ändpunkternas olika metoder och eventuella tidsbegränsningar.

Ifall upprepade anrop till en metod görs under tiden denna är i timeout svarar tjänsten med ett `SoapFault`, även om anropet i övrigt är korrekt. I detta fall finns det två möjliga tillvägagångssätt, att kontinuerligt upprepa anropet ända tills tjänsten returnerar önskad data, eller att vänta den utsatta tiden innan nästa anrop görs. I fallet HatSystems Connector gjordes ett beslut att vänta den utsatta tiden innan nästa anrop.

Tabell 1. StockService tjänsten.

Tjänst	Metod	Timeout	Övrigt
StockService	ArticleAveragePrice		
	DeleteImages		
	GetArticleInformation		
	GetArticleInformationAsObjects	60s	max 1000 rader
	GetBookedLinesOnDeliverynote		
	GetClerkAndShopInfo		
	GetClientShopsStockInfos		
	GetFirstMachineNumber		
	GetImage		
	GetItemgroups		
	GetLastItemGroupIncludedInSale		
	GetOpenProductTransfers		
	GetOpenShipments		
	GetShipmentConfirmations		
	GetSimpleClientShopsStockInfos		
	GetStockBySupplier		
	GetSuppliers	30s	
	GetWebshopArticles		
	GetWebshopArticlesFilterByShopId		
	GetWebshopArticlesInEdbNumberRange		
	GetWebshopArticlesPaged		
	GetWebshopArticlesSinceEdbNumber		
	GetWebshopArticlesSinceLastRequest		
	GetImportDeliveryNoteHistory		
	MakeDeliveryNote		
	MakeDeliveryNote_Bulk		
	PriceChange		
	StockCount		
	StockCountByEdbnumber		
	StockCountByParameter		
	StockCountInShops		
	StockCountSingleEan	0s	
	StockValueByBrand		
	UpdateProductTransfers		
	ValidateArticleFile		

Tabell 2. SaleService och CustomerService tjänsterna.

Tjänst	Metod	Timeout	Övrigt
SaleService	GenerateCertificatenumber GetCertificates GetCustomerTransactions GetKeyFigures GetKeyfiguresByTerminal GetSales GetSalesSinceDateTime GetSpecificTransaction GetTransactions GetTransactionsBySupplier GetTransactionsPaged GetTransactionsSinceLastNumber GetTransactionsSinceLastNumberBySupplier GetWebOrders RemoveWebPosParkedReceipt RetrySales SaleCouldNotBeDelivered SaveSale SaveSale_Bulk SaveWebshopOrder StartSale ValidateCertificate WebOrdersReadyForPick WebPosParkedReceipt		
CustomerService	GetCustomerClubMembers GetCustomerCollection GetCustomerCollettionSince GetCustomerInformationById GetCustomerInformationByNumber GetCustomerTransactions GetEmployeeRegistrations GetLoyaltyMembers GetLoyaltyMembersByCriteria GetLoyaltyMembersSince GetPointsForLoyaltyMember GetStores InsertLoyaltyMembers InsertManualPoints InsertTransactions RecalculatePoints RegisterCustomer RegisterEmployeeTime SaveClerksAsCustomers SetCustomers SetStores ValidLoyaltyLogin		

2.3 Hårdvara

Data Detail Danmark A/S erbjuder en hårdvarulösning för sitt kassasystem och säljer detta som en helhet tilsammans med molntjänsten. Eftersom programmet kräver kundspecifik konfiguration är det inte möjligt för kunden att använda egen hårdvara. Kassasystemet har utöver införskaffningsutgifterna även en fortlöpande månadsavgift för molntjänsten.

3 NÄTBUTIKEN MAGENTO 2

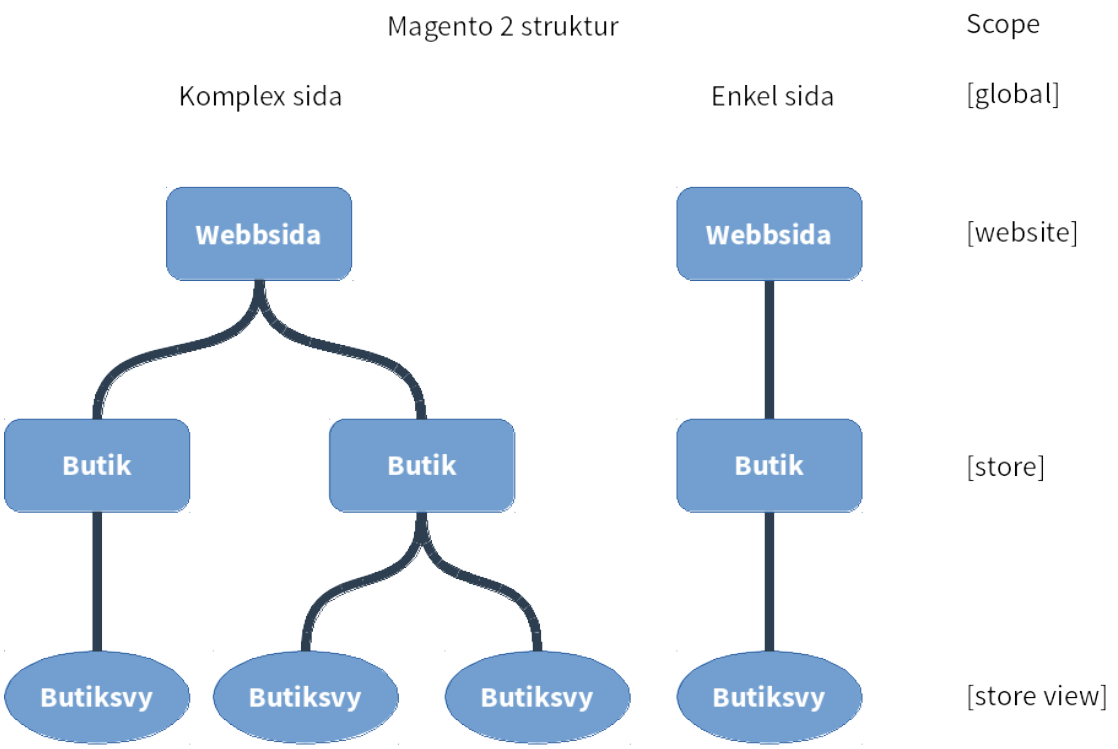
Magento är en nätbutik som baserar sig på öppen källkod. Nätbutiken är ett omfattande ramverk med avancerad lagerhållnings- och rapporteringsfunktionalitet. Funktionaliteten hos Magento 2 kan ytterligare utökas i form av externa tillägg. Magento 2 är byggt på flera ramverk, bland dessa ramverk finns bland andra Zend och Symfony ramverken. Magento 2 finns i två varianter, Community Edition som är en gratis version av nätbutiken, samt Enterprise Edition som är en betald version med utökad funktionalitet. Detta arbete behandlar Magento Community Edition 2.1.

Magento installationen har en strukturerad hierarki med nätsida, butik och butiksvy. En enskild Magento installation kan ansvara för flera nätsidor med en eller flera butiker. Butikerna kan i sin tur bestå av en eller flera butiksvyer. Butiksvyerna kan bestå av exempelvis olika översättningar för butiken. För att definiera var i denna hierarki olika delar befinner sig används benämningen scope. Nätsidor, butiker och vyer har sinsemellan en en-till-flera relation i nedstigande led. Denna hierarki beskrivs närmare i Figur 2. Grundtanken bakom denna uppdelning är att en butiksinnehavare ges möjligheten att administrera ett gemensamt lager för alla sina butiker. Produkterna kan variera mellan butikerna och nätsidorna men finns alla sparade på en och samma Magento installation. Detta underlättar lagerhantering och uppföljning för butiksinnehavaren. I kundens fall rör det sig dock om en enkel installation med endast en butiksvy i likhet med figuren längst till höger i figur 2.

Magento installationen och de olika butikerna och vyerna administreras från Magentos Admin område. Admin området är lösenordsskyddat och använder sig av sessionsnycklar för att skydda sig från CSRF (Cross Site Request Forging) attacker. Inloggningen till admin området kan även förses med ett CAPTCHA för ytterligare skydd.

3.1 Data

Magentos databas är av EAV typ (Entity, Attribute, Value). Databasen består av skilda tabeller för objekt, attribut och värden. Dessa tabeller är sammanlänkade med hjälp av id nummer. Orsaken till detta är att man försöker minimera databasens storlek samtidigt som



Figur 2. Exempel på butiksstruktur i nätbutiken

man behåller all data för de produkter eller kunder som sparas. Denna uppdelning gör att man snabbare kan söka information om en produkt genom att endast ladda den tabell som är av intresse iställe för att man laddar alla attribut som finns i databasen.

Magento sparar produkter och produktattribut skilt i databasen. Produkterna sparas i en egen tabell `catalog_product` i databasen. Produkternas attribut som t.ex färg, storlek, etc. sparas i sina egna tabeller. Attributens värden sparas sedan i sin tur i en egen tabell (storlek 36, färg blå, etc.). Dessa länkas sedan samman till en helhet med hjälp av Magentos datamodeller.

Magento avråder utvecklare från att direkt modifiera data som finns i databasen och använder sig i stället av s.k. Service contracts. Dessa kontrakt kopplar loss logiken från data och gör systemet mera modulärt. Det ger även en försäkran om att koden med större sannolikhet fungerar i framtiden även om den underliggande logiken ändras. (Magento 2 Service Contracts, 2016)

Service kontrakten förespråkar användning Factory paradigmen för att generera objekt vars modell starkt hänger ihop med datan. Magento använder sig av automatisk kodge-

neration för att skapa dessa Factory klasser efter behov. Klasserna skapas på basen av modellerna och behöver ej skapas skilt om inte specifik funktionalitet krävs.

Datamodellerna som finns i Magento liknar de i DdD även om upplägget varierar något och namnen på data, variabler och kolumner är annorlunda. En närmare förklaring på sammankopplingen av data ges i kapitel 4.2 *Val av relevant data*.

3.2 Tillägg

Magentos modulära struktur gör det möjligt för användaren att utöka eller modifiera grundfunktionaliteten hos nätbutiken genom att implementera egna tillägg. Dessa tillägg delas upp i tre olika områden och förklaras i figur 3.

Magento kräver att modulerna följer riktlinjerna som det rådgivande organet PHP Framework Interoperability Group ger i sin PHP Standard Recommendation PSR-4 (PHP FIG PSR-4, 2016). Magento erbjuder utvecklare ett centraliserat sätt att marknadsföra och distribuera sina tillägg i form av Magento marketplace. Utvecklare kan dock själva fritt bestämma vilken kanal de vill använda för att distribuera sina tillägg. Tillägg som distribueras via Magento marketplace har inspekterats och godkänts av Magento och kan allmänt ses som säkra att använda.

3.2.1 Uppbyggnad

Tillägg i Magento befinner sig i mappen `<Magento>/app/code/`. Det är kutym att använda mappstrukturen `<Magento>/app/code/<Namespace>/<Modulnamn>/` för att vidare särskilja olika moduler. För att Magento ska känna igen tillägget krävs filerna *registration.php* samt *etc/module.xml*. Användning av PHP verktyget composer kräver även filen *composer.json*

Tilläggen implementerar Service Contracts och meddelar sina dependencies i composer.json filen. På så vis kan modulen garantera kompatibilitet mellan olika Magento versioner. Service kontrakten har också den fördelen att de förenklar datamodellen. Användningen av servicekontrakt gör att konflikter mellan olika versioner bättre kan undvikas

Tabell 3. Olika typer av Magento 2 tillägg.

Tilläggstyp	Användning
Moduler	Samverkar med andra delar av systemet Modifierar systemets funktionalitet Implementerar egen tilläggsfunktionalitet
Teman	Ändrar det visuella utseendet på nätbutiken
Språkpaket	Erbjuder översättningar av nätbutiken till olika språk

eftersom kontrakten föreskriver väl definierade interfaces istället för att använda interna metoder vilka möjligtvis ändrar i framtiden.

Magento erbjuder flera olika möjligheter att implementera tilläggsfunktionalitet.

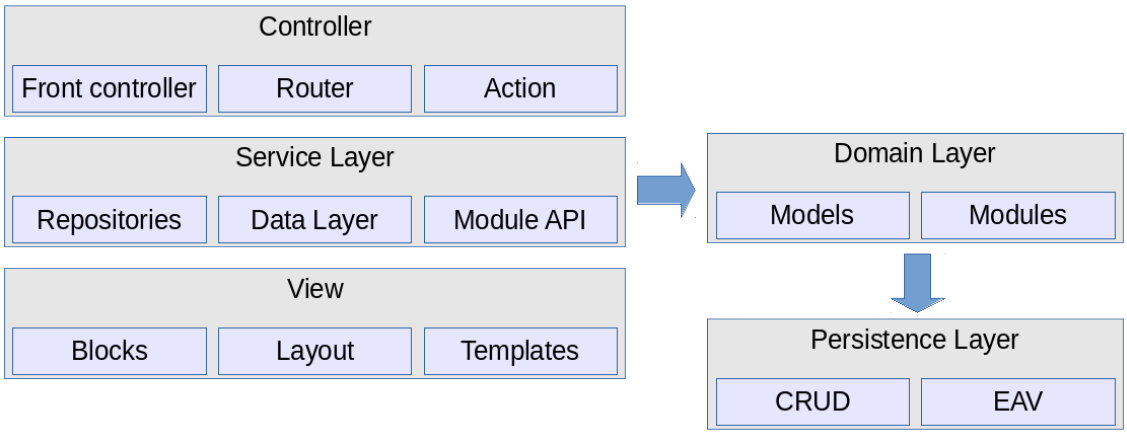
Magento följer Model-View-Controller paradigmen men har utöver dessa grundstenar även andra klasser med särskild funktionalitet.

Cron klasserna används för att köra olika metoder i utsatta intervaller.

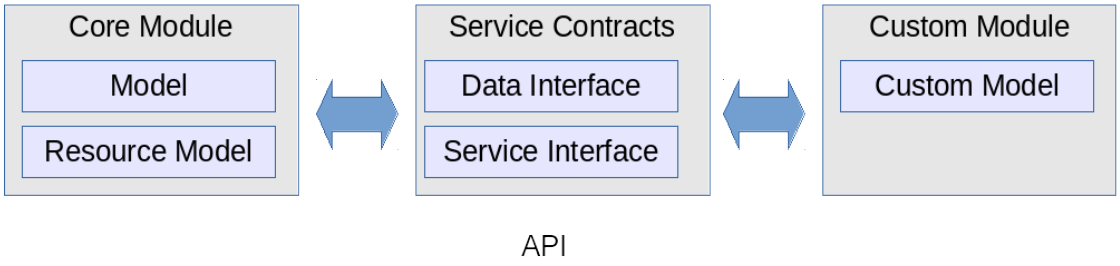
Händelser (eng. Events) i Magento skickar meddelanden baserade på publish-subscribe modellen. Dessa meddelanden kan övervakas av Observer klasser vilka prenumererar på händelsemeddelanden. Observer klassen agerar sedan på detta meddelande. Exempel på en händelse är att kunden genomför ett köp, en s.k. transaktionshändelse.

Plugin klasserna kan användas för att modifiera befintlig grundfunktionalitet. Plugin klasserna modifierar data i modellen innan det skickas till view klassen. Användningen av plugins gör att utvecklaren kan undvika att direkt modifiera grundläggande funktionalitet i Magentos kod, något som avråds av utvecklardokumentationen. Plugins gör det även möjligt för flera moduler att samtidigt modifiera samma funktionalitet utan att skapa konflikter i koden.

Setup klasserna används i de fall där tillägget kräver att nya tabeller skapas i databasen. Detta görs med klassen *InstallSchema.php*. För att installera data i samband med installationen av modulen används klassen *InstallData.php*. Klasserna *UpgradeSchema.php* samt *UpgradeData.php* används för att modifiera befintliga användardefinierade tabeller och data i samband med uppgraderingen av tillägget.



Figur 3. Uppbygnad av Magento Modul



Figur 4. Uppbyggnad av Magento API

3.2.2 Installation

Magento använder sig av php verktyget composer för att installera insticksmoduler och deras dependencies. Magento marketplace ger även möjlighet för inloggade användare att köpa moduler och installera dessa rakt till magento installationer länkade till användarkontot. Denna möjlighet har ej utforskats i detta arbete. Det går även att installera manuellt genom att köra flytta filerna till `app/code/<namespace>/<modulnamn>` och sedan köra `php bin/magento setup:upgrade`.

4 SYSTEMINTEGRERING

I den befintliga lösningen är kassadatabasen direkt kopplad till ett externt ERP system. Detta system fungerar som den huvudsakliga knutpunkten och har hand om all data.

Då målet med arbetet är att implementera motsvarande funktionalitet med hjälp av endast DdD och Magento krävs en noggrann genomgång av de olika delarnas ansvarsområden. Den kanske viktigaste komponenten för en lyckad integrering är en genomgång av de olika värdena som sparas i respektive system, samt en kartläggning över hur dessa värden skall länkas samman.

Syftet med modulen är att överföra data. Målet är att undvika duplikation av data i mån av möjlighet eftersom detta kan leda till data uppdateras i ett system men den uppdaterade informationen ej förs över till det andra systemet. Eftersom data som finns i de båda systemen till stor del är statisk kommer detta dock inte att vara ett problem. En viss grad av dataduplikation är dock nödvändig för att kunna länka samman de olika systemen. Data som krävs för kommunikationen mellan systemen är naturligtvis nödvändig att spara, men eftersom ändpunkterna har olika datastrukturer krävs användardefinierade datastrukturer som länkar de tidigare nämda strukturerna samman. I detta fall har modeller skapats i Magento som motsvarar de modeller som används i DdD. Data från dessa modeller länkas sedan till de produktmodeller som används av Magento för att kunna föra in ny data i nätbutiken.

4.1 Procedurer

Eftersom DdD kräver att kommunikationen med ändpunkterna initieras av klienten faller detta ansvar på Magento. Tjänsterna saknar PUSH funktionalitet vilket gör att Magento kontinuerligt måste skicka förfrågningar om nya data och händelser för att hålla sin egen data ajour. Denna kommunikation begränsas av SOAP metodernas väntetider.

Produktens livscykel startar då den förs in i det molntjänstbaserade backoffice systemet. Produkter kommer att föras in i systemet av butiksinnehavaren med hjälp av webbgränssnittet. Den huvudsakliga datan befinner sig i kassasystemet DdD och överförs maski-

nellt till Magento, på så vis minimeras risken för felaktig datainmatning som en följd av mänskliga misstag. Den maskinella överföringen underlättar även avsevärt slutanvändarens arbete då datan ej behöver matas in manuellt på båda platformer utan endast behöver verifieras.

Produktinformationen som sparas i Magento är mera ingående än den som sparas i DdD. Informationen matas först in i kassasystemet och överförs sedan till Magento var den kompletteras av slutanvändaren. Den kompletterade data har dock inget mervärde för kassasystemet och skickas ej tillbaka. DdD fungerar i detta fall som en s.k. Article Master.

DdD ansvarar för lagerstatusen hos produkterna. Denna data måste hämtas till Magento med jämna mellanrum och uppdateras där. Uppdateringen av produktdata sker automatiskt med hjälp av ett cron skript som körs i bakgrunden på utsatta tider. Skriptet uppdaterar lagerstatusen till det värde som rapporteras av kassasystemet. Magento tillägget använder sig även av en plugin klass för att uppdatera en produkts lagerstatus när en användare lägger till produkten i sin köpkorg. Detta försäkrar att beställda produkter kan levereras. Tillägget använder sig av ett motsvarande skript på basen av Observer klassen för att skicka information om beställningar och försäljningstransaktioner gjorda i Magento till kassasystemet.

4.2 Val av relevant data

DdD använder sig av EAN, EDB, Kparam samt Vparam för att hålla reda på produkter i systemet. Kparam består av fem olika fält som är gemensamma för en produkttyp så som exempelvis tillverkare och märke. Vparam består av fem fält där produkternas varianter skapas så som färg och storlek. Dessa fält modifieras från DdDs sida enligt kundens önskemål när kassasystemet implementeras. Kunden har valt att behandla produkter med olika färger som unika produkter i motsats till att definiera produkternas färg och storlek som variabla attribut. Detta val medför inga problem i någotdera systemet men de befintliga datamodellerna används ej till sitt fulla potential.

Valet att endast ha ett variabelt produktattribut (i detta fall storlek) är medvetet och har

gjorts på kundens begäran eftersom deras tidigare data följer denna struktur. Tilläggets uppbyggnad gör det dock möjligt att enkelt implementera flera variabla attribut från DdD i Magento.

EAN koderna är essentiella då de används för att hämta produktinformation från kassan. Magento saknar ett fält för dessa koder i sin grundkonfiguration och dessa måste läggas till som en kolumn eller egen tabell i databasen.

DdD ställer inga krav på den produktdata som genereras där utan använder det fortlöpande EDB numret för att skilja produkterna åt. Magento använder på dylikt sätt ett internt fortlöpande id nummer för att särskilja produkterna. Utöver detta id används även en SKU (Stock Keeping Unit) kod för att skilja produkterna åt. SKU koden måste vara unik för alla produkter.

4.3 Dataöverföring

Magento har det huvudsakliga ansvaret för att bestämma vilken data som skickas samt i vilket skede den skickas. Nätbutiken och den vanliga butiken har ett gemensamt lager. För att hålla lagerstatusen uppdaterad i realtid på nätbutiken krävs att kassasystemet skickar denna information genast när köpet görs. I praktiken är detta dock inte möjligt eftersom Magento ansvarar för initieringen av dataöverföringen. Detta leder till att vi ej kan garantera att lagerstatusen i nätbutiken är omedelbart uppdaterad utan den kommer att uppdateras var 15:e minut.

Varumärken finns som valbara attribut hos produkterna i Magento och listan med varumärken uppdateras samtidigt som produkternas lagerstatus. Denna uppdatering kommer samtidigt att verifiera att den övriga produktinformationen är korrekt. Den enda uppdateringen som kräver direkta åtgärder i Magento är ifall produktens EAN kod byts. Detta kommer dock inte att ske under normala omständigheter.

Tillägget sparar en lista med varumärken i en egen tabell i Magento. Dessa varumärken går under benämningen Suppliers i kassasystemet. Listan uppdateras med ett anrop till metoden `StockService\GetSuppliers`. Slutanvändaren behöver ej mata in varumärken

själv utan dessa ska finnas som en valbar variabel i produktens information. Ändpunkten skickar ej namnet på varumärket utan endast ett nummervärde som motsvarar värdet som finns i kassasystemet. Därför görs ett anrop som hämtar och uppdaterar listan med varumärken i Magento före anrop som hämtar produktinformation.

4.3.1 Produktinformation

För att importera produkter från DdD APIn används SOAP tjänsten `StockService`. Ett anrop bestående av butiks-id, nyckel, samt lista med EAN koder för produkterna i fråga görs och `StockService` skickar tillbaka en array med objekt av klassen `ArticleLines`.

På basen av listan med ArticleLines skapas sedan produkterna i Magento. Ett liknande anrop görs i de fall produktdatan i Magento vill verifieras och uppdateras. Produkterna skapas med den autogenererade klassen `ProductFactory` och implementerar `ProductResourceInterface` för att spara produkterna.

EAN koderna för anropet kommer att variera beroende på vilken controller som initierar anropet. Ifall det är frågan om överföring av nya produkter hämtas en lista med EAN koder från en fil på systemet. Ifall det är fråga om en rutinmässig kontroll av nya produkter genereras EAN koder utgående från det nuvarande högsta EDB numret som finns tillgängligt i Magento. Vid uppdateringen av lagerstatus hämtas en lista på befintliga produkters EAN koder.

Kommunikationen initieras genom en statisk referens till den önskade tjänsten i DdDs SOAP API. Dessa ändpunkter används både vid manuellt initialiserad överföring samt av Cron jobbet som kontinuerligt frågar efter ny data. Cron jobbet anropar Controller klasserna. Samma klasser används även tillsammans med block klasserna för den manuella uppdateringen i Magentos adminområde.

Magento använder sig av factories för att generera objekt vars model starkt hänger ihop med datan. För att skapa nya produkter i magento används klassen `ProductFactory`. Produkter sparas sedan med hjälp av klassen `ProductRepositoryInterface`.

Samma procedur används även vid skapandet av kunder men då implementeras istället

`CustomerRepositoryInterface`.

För att hämta produktinformation från DdD används SOAP funktionen `GetArticleInformationAsObjects`. Ett nytt objekt skapas på basen av DdD modellen som definierats i Magento. Objektet har tre variabler: clientid, psk samt eans. Clientid och psk hämtas från databasen, sedan genereras en lista med EAN koder som sätts in i variabeln eans. Efter detta görs ett anrop till StockService ändpunkten i DdD. DdD skickar tillbaka en lista med produktinformation. Denna lista grupperas på basen av ArticleNumber, Supplier samt färg. Ifall en grupp innehåller flera produkter måste en huvudprodukt (Configurable Product) skapas i Magento. Sedan skapas enkla produkter och dessa länkas som variabla produkter till huvudprodukten. Ifall en grupp endast innehåller en produkt skapas endast en enkel produkt utan varianter.

Produktimporten kan användas för att importera enskilda produkter eller flera produkter samtidigt beroende på listan med EAN koder som skickas i anropet. Tillägget genererar automatiskt kommande EAN koder på basen av det senaste EDB numret som finns sparat i databasen. Ändpunkten returnerar endast produkter som finns i DdD. Ifall ett anrop görs med en EAN kod som ej finns i kassasystemet returneras en tom rad.

4.3.2 Transaktioner

Uppgifter om transaktioner skickas från Magento till DdD. SaleService tjänsten kräver meddelandet innehåller tidpunkt, cashier id, EAN kod, antal, betalningstyp samt eventuell kundkod för att transaktionen ska registreras. Vi använder en skild Cashier ID för Magento för att särskilja försäljningen i nätbutiken på DdD rapporten. Information om transaktioner i Magento skickas till DdD först när beställningens status går från *pending* till *shipped*.

4.3.3 Återkommande kunder

DdD kräver att ett telefonnummer anges för nya kunder som registreras. Inga krav ställs dock på formatet för telefonnumret. Detta medför att vi kan skicka endast en siffra eller en godtycklig sifferström till systemet. Magento använder e-postadressen för att särskilja

olika användare. När en ny användare registrerar sig i systemet skickas kundinformationen från Magento till kassasystemet. Samtidigt hämtas kundkoden som genereras i DdD till Magento och sparas tillsammans med den övriga kunddatan. Kunden kan sedan använda sin kundkod för att identifiera sig i den fysiska butiken för att ta del av eventuella stamkundserbjudanden.

5 TEKNISK IMPLEMENTATION

Magento 2 tillägget HÄTSystems Connector har utvecklats enligt de riktlinjer som Magento ger i sin utvecklardokumentation. Tilläggets inställningar har förts in som en egen sida i Magento installationens admin område under fliken `Store -> Configuration`. Kundspecifika inställningar som t.ex. användarautentisering med SOAP tjänsterna sätts in manuellt i samband med installationen. Orsaken till detta är att undvika att data som är specifik för en viss kund finns i kodbasen. På så vis kan samma kodbas distribueras till flera kunder utan att kundspecifik data hamnar i fel händer. Kundspecifika värden matas in i på inställningssidan efter att tillägget har installerats. Till dessa värden hör kundens DdD användar id, kundens privata DdD API nyckel, DdD cashier id, samt webbadresser för de olika tjänsterna som används.

Inställningssidan har även knappar och områden för manuell import av data från DdD till Magento. Under projektets gång övervägdes att sätta dessa knappar i listan med produktinformation men efter övervägande gjordes ett beslut att flytta dessa knappar till setup-området. Beslutet baserar sig på det faktum att dessa funktioner ej kommer att användas manuellt av slutanvändaren i dagligt bruk och ej kräver större synlighet.

Utöver denna inställningssida finns inga visuella delar i tillägget. Överföringen av data och transaktioner sköts i bakgrunden med hjälp av Cron jobb. Slutanvändaren informeras om lyckad produktimport i form av ett popup meddelande. Detta registreras även till en loggfil på servern tillsammans med information gällande problem med tillägget eller dataöverföringen.

5.1 Klasser och Modeller

Tillägget består av ett flertal olika delar med olika ansvarsområden. Koden följer PSR rekommendationerna som utarbetats av konsortiet fig-psr.

Datamodeller för DdD klasserna finns i mappen Model/DdD, dessa är uppdelade i undermapparna Stock, Sale samt Customer. Modellerna krävs för att kommunikationen med DdD APIn ska fungera och har samma struktur och variabler som de motsvarande änd-

punkterna i kassasystemet DdD.

Två stycken hjälpklasser har skapats för att underlätta hanteringen av kommunikationen med DdD samt hanteringen av attribut i Magento. Klassen `Helper/Attribute` används för att generera och hämta produktattribut så som färger och storlekar. Klassen används även för behandlingen av varumärken. Klassen `Helper/Data` används för att hämta de kundspecifika inställningarna som behövs för att skicka meddelanden till ändpunkterna i DdD så som kundnummer, privat nyckel och dylikt.

Klasserna i Cron mappen används för att hämta information om nya produkter till Magento samt hämta transaktioner som skett på kassan. Ifall det finns nya produkter matas dessa in i systemet och användaren meddelas. Försäljningshändelser hämtas kontinuerligt från kassan och lagerstatusen uppdateras för de produkter som sålts, på så vis kan lagerstatusen i Magento hållas uppdaterad även om lagervärdet ej uppdateras i realtid utan har en fördröjning baserad på hur ofta förfrågningarna hämtas från kassasystemet.

Etc mappen innehåller inställningar för olika delar av tillägget. Filen *acl.xml* används för hanteringen så kallade Acess Control Lists. Dessa beskriver vilka ändpunkter i admin området som är giltiga för olika slutanvändare och krävs för att användaren skall ha åtgång till tilläggets inställningar. Filen *di.xml* används för att precisera hur olika dependencies skall hanteras, samt vilka klasser som beaktas som singletons. Här anges även standardvärden som skall användas när olika klasser instansieras. *module.xml* innehåller information om tillägget och krävs av Magento för att tillägget skall kunna installeras.

Observer mappen innehåller en klass *SalesOrderSaveAfter.php* som lyssnar på händelser som skickas när en beställning uppdateras i admin området. Denna klass skickar sedan information om gjorda beställningar till kassasystemet med hjälp av data modellerna och service klasserna.

Plugin mappen innehåller en klass vars uppgift är att verifiera att en produkt finns i kassasystemets lager innan produkten kan adderas till en köpkorg i nätbutiken. Plugin klassen skickar ett anrop till `StockService\GetSingleStockItem` och uppdaterar nätbutikens lager innan produkten visas för kunden.

Setup mappen innehåller filer som används för att generera tabeller i Magento databasen när tillägget installeras. Install klasserna körs första gången tillägget installeras. Update klasserna körs ifall tilläggets versionsnummer uppdateras.

5.2 Kommunikation

Tillägget kommunicerar med kassasystemet med hjälp av SOAP gränssnittet som definieras av kassasystemet. SOAP protokollet använder sig av XML meddelanden för att skicka information mellan servern och klienten. SOAP protokollet ställer inga krav på innehållet i meddelandena utan tjänsten kan fritt specifiera meddelandets struktur och innehåll. Kravet är dock att meddelandet som skickas är formatterat som korrekt XML kod. (W3C Recommendation, 2007)

PHP har en inbyggd `SoapClient` klass som används för ändamålet. För att underlätta kommunikationen har klasser skapats i Magento som motsvarar de klasser som finns i DdD. Kommunikationen sker genom att instansiera objekt av dessa klasser som sedan skickas med hjälp av soapclient klassen. När data tas emot från ändpunkterna i DdD använder soapclient klassen de motsvarande klasserna i Magento för att instansiera nya objekt. Detta har gjorts för att undvika behovet att göra en egen implementation av en XML parser som går igenom meddelandena som skickas från DdD. Implementationen med PHPs inbyggda soapclient klass fungerar utmärkt och kommunikationen har observerats vara felfri.

Zend ramverket som Magento 2 är byggt på erbjuder även en soap klient implementation som kan användas för samma ändamål men användningen av denna klass har ej undersökts närmare då den i programmeringsspråket inbyggda klassen konstaterats fungera.

6 SLUTSATSER

Målsättningen med detta examensarbete har varit att utreda vilka möjligheter Magento 2 erbjuder för utvecklare att utöka de inbyggda funktionaliteterna och skapa en tilläggsmodul för kommunikation med en extern SOAP API. Slutsatsen av detta examensarbete är en Magento modul som uppfyller målsättningen samt beställarens krav. Magentos modulära struktur gör att tillägg för olika ändamål kan produceras relativt smidigt. Magento har två olika systeminställningar (eng. mode), developer samt production. I production mode genereras objekt och kod färdigt på basen av de olika klassernas bindningar, detta gör att prestandan på webbsidan ökar avsevärt i jämförelse med developer mode. I developer mode genereras all kod anefter som olika objekt skapas, detta gör att prestandan försämras och kan i viss mån leda till frustration ifall hårdvaran som används för utvecklingsarbetet ej har tillräcklig prestanda.

Slutprodukten är ett Magento tillägg som uppfyller de specifikationer Magento ger. Tillägget klarar av att importera produkter till systemet från en extern SOAP tjänst. Denna import körs kontinuerligt i bakgrunden samtidigt som information om försäljning av produkter via nätbutiken skickas till kassasystemet. Tillägget uppdaterar nätbutikens lagerstatus genom att be tjänsten om transaktioner som skett i kassasystemet. Tillägget kommer ännu att utökas för att kunna hämta kundinformation från kassasystemet och sammanlänka dessa kundkonton med kundkonton som finns i nätbutiken.

Magento 2 Community Edition verkar på basen av de observationer som gjorts under arbetets gång vara en välstrukturerad nätbutikslösning med goda möjligheter för utvecklare att skapa egen tilläggsfunktionalitet. Prestandan hos nätbutiken är ej direkt beroende på mängden produkter som finns i databasen men ställer högre krav än förväntat på optimering för att kunna erbjuda en smidig upplevelse för slutanvändaren. Nätbutiken har en logisk struktur beträffande produkter och deras attribut. Dokumentationen för nätbutiken är riktar sig till såväl slutanvändare som utvecklare. Dokumentationen saknar dock riktigt bra exempelkod, istället hänvisas läsaren till källkoden. Källkoden är strukturerad på ett sätt förespråkar testdriven utveckling. Utvecklingsarbetet underlättas dock av det faktum att nätbutikens källkod är öppen.

KÄLLOR

Data Detail Danmark A/S. (2012). *DdD Service Online Documentation.* Hämtad från http://api.dddadmin.com/doc/index.html den 14 december 2016

Data Detail Danmark A/S. (2012). *DdD MakeDeliveryNote example.* Hämtad från http://api.dddadmin.com/doc/Examples/StockService_MakeDeliveryNote.txt den 19 september 2016

Magento 2 Developer Documentation. (2016). Hämtad från http://devdocs.magento.com den 20 december 2016

Magento 2 Developer Documentation (2016). *Developer Documentation for Extensions.* Hämtad från http://devdocs.magento.com den 20 december 2016

Magento 2 Developer Documentation for Extensions. (2016). *Service Contracts.* Hämtad från http://devdocs.magento.com/guides/v2.0/extension-dev-guide/service-contracts/design-patterns.html den 14 december 2016

PHP Framework Interoperability Group. (2016). *PHP Standard Recommendation 4.* Hämtad från http://www.php-fig.org/psr/psr-4/ den 14 december 2016

World Wide Web Consortium. (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* Hämtad från https://www.w3.org/TR/2007/REC-soap12-part1-20070427/ den 14 december 2016