Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Integrationskatalog

- teknisk dokumentation av ett integrerat system

av

Johan Filipsson

LIU-IDA/LITH-EX-G--13/052--SE

2013-11-21

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Integrationskatalog- teknisk dokumentation av ett integrerat system

av

Johan Filipsson

LIU-IDA/LITH-EX-G--13/052--SE

2013-11-21

Handledare: Samuel Johansson, Ipendo Systems
Examinator: Patrick Lambrix, Linköpings Universitet

Innehållsförteckning

Förord

Denna uppsats är skriven på Institutionen för Datavetenskap vid Linköpings Universitet och omfattar 16 högskolepoäng. Arbetet har varit den avslutande delen i en Högskoleingenjörsexamen. Idén till arbetet framfördes av Ipendo Systems då de var i behov av en ny metod för att hantera teknisk dokumentation över företagets arbete med systemintegration.

Jag skulle vilja rikta ett tack till Ipendo Systems för att jag fick göra mitt exjobb hos dem och ett speciellt tack till min handledare Samuel Johansson för all hjälp och stöd. Ytterliggare ett tack till Universitetet och min examinator Patrick Lambrix. Ett sista tack går till Emilia Johansson och Jakob Willforss för hjälp och stöd och för att ni är så bra på att motivera mig.

Linköping, Sverige, Augusti 2013

Johan Philipsson

Sammanfattning

Systemintegration är arbetet med att få IT-system att samarbeta med varandra. Inom systemintegration finns det idag ett glapp mellan den tekniska dokumentationen och den tekniska lösningen vilket försvårar arbetet med att ta fram och underhålla integrerade system. För att underlätta detta skapades på uppdrag av Ipendo Systems en databasmodell för att kunna lagra teknisk dokumentation om integrationssystem. Lösningen implementerades i Microsoft SQL Server. Under arbetet utvecklades stöd för hämtning av data från integrationsplattformen BizTalk. Utöver detta utvecklades också möjlighet att hantera supportärenden och hämta in incidenter från externa system. För att kunna hantera databasen på ett enkelt sätt så utvecklades en prototyp av ett administratörsgränssnitt för enklare insättning av data och hantering av databasen.

Kapitel 1

Inledning

1.1 Bakgrund

Inom IT-världen är integration en process där man kopplar samman olika datorsystem för att de ska kunna samarbeta som ett stort system.

Flera företag och organisationer har idag väldigt omfattande IT-system som består av många olika individuella program. För att få ut det mesta av dessa system så krävs det att de är sammankopplade och kan utbyta information. Detta brukar refereras till som systemintegration.

1.1.1 Integrationsproblem

I integrationslösningar finns det idag ett glapp mellan verksamhet och teknik. Detta beror på att verksamheten har sin egen modell och sitt eget perspektiv på processerna. Det är också vanligt att företag samlat på sig många datorer och system som sedan integreras med en mindre stabil ad hoc lösning. Så länge ett system fungerar så har det varit okej. Detta glapp har dock skapat tätt kopplade och röriga system. Svårigheter kan då uppstå med att kartlägga mellan den tekniska implementationen och verksamhetens uppfattning. Behovet finns därför att överbygga glappet mellan verksamhet och teknik-nära data. Ytterligare ett problem är att statisk dokumentation lätt blir utdaterad(Cummins, F. 2002).

Som ett exempel kan nämnas att när jag började med mitt examensarbete fick jag ta del av den nuvarande dokumentationen från en av Ipendo Systems kunder. I detta integrerade systemet fanns ungefär 50 olika mindre system. Då var den dokumentationen utdaterad så det verkliga systemet hade växt ytterligare.

1.1.2 IpendoSystems

Ipendo Systems är ett IT företag som är baserat i Linköping. Företaget har ungefär 35 anställda. De har två huvudinriktningar på sin verksamhet,. En del som kallas för Solution Experts och arbetar med portal- och integrationslösningar. Den andra delen kallas Ipendo Solutions. De arbetar med att utveckla och förvalta en branschspecifik portallösning inom immateriella rättigheter. Företaget har specialiserat sig på systemintegration, såväl strategi som genomförande och förvaltning. De förvaltar och driver integrationsplattformar åt 6 kunder. Flera av dessa kunder är stora koncerner med stora IT-system. Det finns därför ett behov av uppdaterad kartläggning av befintliga integrationer för att ha kontroll över informationsflödet.

1.2 Syfte och frågeställningar

1.2.1 Databasutveckling

Detta arbete avser att utveckla en databasmodell för att lagra teknisk dokumentation om systemintegration. Databasmodellen ska vara generell i det avseendet att den är tillämpningsbar för flera olika former av system och kan användas tillsammans med olika integrationsmotorer. Modellen kommer att implementeras i SQL Server för att kunna verifiera funktionalitet. Det kommer även att utvecklas stöd för att importera data från Integrationsplattformen Biztalk och sammankoppla denna data med databasen. För att enkelt hantera databasen är avsikten att utveckla ett administratörsgränssnitt i Microsoft Lightswitch.

Genom att utveckla applikationer som utnyttjar den dokumentation som finns lagrad i databasen kan man skapa ett mervärde av att ha uppdaterad data i databasen. Om den tekniska dokumentationen är en central del av företagets/organisationens verksamhet är sannolikheten större att det kommer att läggas mer energi på att hålla denna information uppdaterad. Idag händer det att processen att ta fram teknisk dokumentation släpar efter. Fokus läggs ofta på att ta fram en fungerande applikation som gör det som avses. Det avsätts ofta inte tillräckligt timmar i utvecklingsarbetet vilket gör att det inte finns möjlighet för utvecklarna att upprätthålla den kvalité som krävs av dokumentationen för ett system. Detta är också något som jag ska försöka underlätta med den här databasen. Genom att skapa möjligheter att importera data från den faktiska integrationsplattformen så minskar jag den arbetsbörda som kommer krävas för att hålla dokumentationen uppdaterad.

1.3 Upplägg på rapporten

Rapporten kommer att börja med ett bakgrundskapitel för att göra läsaren bekant med ämnet. Därefter kommer de krav som ställdes på arbetet att gås igenom. I fjärde kapitlet beskrivs hur utvecklingen av databasen gick till. Först görs en grundläggande analys av hur databasen kommer att vara upplagd sedan presenteras den färdiga databasens olika tabeller. I femte kapitlet beskrivs det administratörsgränssnitt som utvecklades för att kunna hantera data i databasen på ett enklare sätt. Rapporten avslutas med en diskussion där arbetet med integrationskatalogen analyseras och olika möjligheter att utveckla integrationskatalogen tas upp.

Kapitel 2

Bakgrund

Detta kapitel kommer att behandla den tekniska bakgrunden som ligger till grund för arbetet. Kapitlet börjar med en genomgång av några grundläggande begrepp som används i rapporten. Därefter följer ett stycke om systemintegration, ett stycke om relationsdatabaser och en beskrivning av den mjukvara som använts i arbetet.

2.1 Begrepp

Detta är en grundläggande genomgång av de begrepp som kommer att användas i rapporten. Dessa är framförallt härledda ur den begreppsterminologi Ipendo Systems använder i sitt arbete med integration.

System och Information

Grunden för integrationen är flera datorsystem som samarbetar genom att utbyta information. De datorsystem som samarbetar genom integrationen benämns här system. Den information som utbyts mellan systemen refereras helt enkelt till som information.

Interface

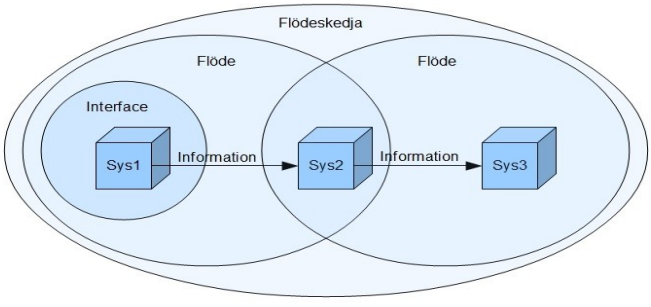
Ett interface är en överföringspunkt för information som skickas eller tas emot av ett system. Ett interface kan vara av två olika typer, antingen *Producer* eller *Consumer*. En *Producer* sänder iväg data till andra system. En *Consumer* tar emot data som skickas av en *Producer*.

Flöde

Två interface med en *Producer* och en *Consumer* bildar ett flöde. Flöde är alltså en beskrivning av processen där information skickas från ett system till ett annat.

Flödeskedja

För större system är ofta utbytet av information mer komplicerat än enbart ett system som skickar information till ett andra system. Det är möjligt att en utväxling av information triggar en andra utväxling som i sin tur triggar ytterligare utväxlingar av information. Det finns ingen begränsning på hur många utväxlingar som kan inträffa. För att beskriva detta används begreppet flödeskedja. En flödeskedja är en samling av flera ordnade flöden. Flödena är ordnade efter i vilken ordning de kommer att inträffa. Det är möjligt med parallellism i en flödeskedja vilket betyder att en kedja måste inte vara en strikt sekventiell följd. Detta för att göra det möjligt att på så korrekt sätt som möjligt beskriva systemet.



Figur 2-1 Förhållandet mellan Interface, Flöden och Flödeskedjor

Integrationsplattform

Detta är ett system med syfte att underlätta kommunikationen mellan andra system. Den integrationsplattform som varit fokus för det här arbetet heter Biztalk och är utvecklad utav Microsoft.

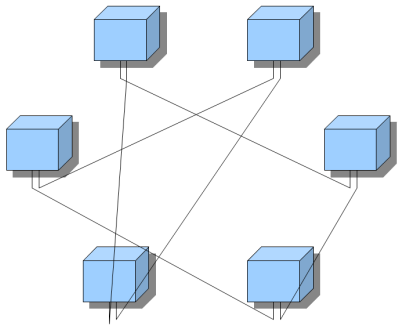
Databasen

När jag i rapporten talar om databasen så avser det den implementationen utav den generella databasmodellen. Implementationen är gjord i Microsoft SQL Server.

2.2 Integration

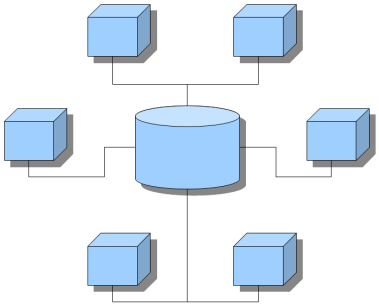
Inom IT-världen är integration en process där man kopplar samman olika datorsystem för att de ska kunna samarbeta som ett stort system.

Det första steget är att låta processer prata med varandra automatiskt. När en kund ringer så kan operatören skriva in kundens uppgifter vilket triggar en uppslagning i kunddatabasen. Det kan också ske en kontroll att det finns tillräckligt många varor i lager eller liknande. Alla dessa delar hanteras av olika system som samarbetar genom att utbyta information. Problemet är att för varje form utav informationsutbyte så krävs en individuell koppling. Ponera ett företag med hundra olika system och program i sin organisation. Många utav dessa program samarbetar och utbyter information med flera andra system, mängden individuella kopplingar kan därför bli mångdubbelt så stor som mängden system(O'Brien, R. 2008). Det märks tydligt på den stora mängden kopplingar att det kommer bli svårt att hantera alla dessa integrationer på ett effektivt sätt. Systemen är väldigt tätt kopplade och om ett problem uppstår någonstans i organisationen så kan det vara svårt att hitta orsaken till detta. Processen att byta ut ett system mot ett annat kan dessutom bli en komplicerad process.



Figur 2-2 Översiktsbild över ett integrerat system med individuella kopplingar

En alternativ lösning för att hantera större system vore att introducera ytterligare ett system vars uppgift är att hantera kopplingar mellan olika system. Varje system kommunicerar då enbart med den centrala integrationsplattformen genom att skicka data till och ta emot data från systemet (O'Brien, R. 2008). Mängden individuella kopplingar kan därför minskas och systemet kommer då att bli enklare att både hantera och felsöka. Detta eftersom de olika delsystemen är mer löst kopplade till varandra. Det finns en central hierarki för hur sammankopplingar sker.



Figur 2-3 Översiktsbild över ett integrerat system med centralt system för att hantera integration

Användandet av en integrationsplattform introducerar dock en ytterligare komplexitet i form av ett extra system. Utöver alla de sammankopplade systemen finns det nu ytterligare ett system att administrera. Överföringshastigheten mellan systemen kan också påverkas negativt eftersom alla paket som skickas mellan systemen måste färdas via integrationsplattformen.

2.3 BizTalk

För att lösa problemet med omfattande integration av många system har Microsoft tagit fram en integrationsplattform som på ett strukturerat och skalbart sätt kan hantera denna integration (Shanmugan, A. 2006). Den använder sig av en horisontell struktur där all kommunikation går genom en central Biztalk-server. Det gör att det bara finns ett flöde för varje typ av information. Data skickas från en applikation till den centrala servern. Därifrån kan den skickas vidare till flera andra applikationer. Biztalk erbjuder möjligheter att definiera affärsdokument som beskriver hur data måste behandlas vid överföring. Data definieras ofta med hjälp av märkspråket XML. Överföring sker med standardiserade internettekniker som HTTP och SMTP. Det finns också möjlighet att bevaka aktivitet på servern för att kunna se vad som händer med flödet av data i realtid.

Det var Ipendo Systems bedömning att det i nuläget inte finns en fullgod lösning inbyggt i BizTalk för att hantera dokumentationen av det systemet.

2.4 Relationsdatabaser

En relationsdatabas är en databas där all data är organiserad i relationer(Groff, J., Weinberg, P., Oppel, A. 2009). Dessa relationer utgörs av en mängd poster med samma egenskaper. I en databas representeras detta av tabeller med rader och kolumner.

2.4.1 SQL – Språket

SQL är ett frågespråk för att hantera data i en relationsdatabas. SQL gör det möjligt att definiera databaser och datastrukturer. Språket kan också användas för att utföra operationer på den data som är lagrad i databasen(Rockoff, L. 2011).

En komplett genomgång av SQL-språket vore inte möjligt i den här rapporten men för att ge alla läsare de grundläggande kunskaperna som behövs för att förstå arbetet kommer här en kort genomgång av de SQL-operatorer som omnämns i rapporten.

CREATE – Detta är en operator för att skapa nya dataobjekt så som tabeller, vyer och även nya databaser.

INSERT, DELETE, UPDATE – Detta är de grundläggande operationerna för att hantera data i en tabell. INSERT används för att lägga till en eller flera nya rader i en tabell. DELETE används för att ta bort en eller flera rader i en tabell. UPDATE används för att uppdatera existerande rader i en tabell.

MERGE – Detta är en operation för att antingen lägga till en rad i en tabell eller uppdatera en befintlig rad beroende på om ett villkor är uppfyllt eller inte.

En vy är en virtuell tabell som består av resultatet utav en SQL-sats. Precis som vanliga tabeller så består en vy utav rader och kolumner. Men innehållet är i regel alltid hämtat från en eller flera tabeller. Det är normalt inte möjligt att lagra data i en vy utan det är ett sätt att presentera befintlig data på ett nytt sätt (SQL Views 1999).

2.4.2 Microsoft SQL Server

SQL server är ett databashanteringsverktyg utvecklat av Microsoft(Rouse, M. 2006). Programmet är en relationsdatabas med SQL som grundspråk. SQL Server använder sig dock utav en variant av SQL som heter Transact SQL(T-SQL). Detta har alla egenskaper som standard SQL men har inkluderat ett antal extra funktioner. SQL Server finns i ett antal versioner men den jag har använt i det här arbetet är SQL Server 2008 R2.

2.5 Microsoft Visual studio

Visual Studio är en uppsättning utvecklingsverktyg utvecklad utav Microsoft. Det finns stöd för att utveckla ett flertal olika sorters applikationer inom ASP.NET, XML, skrivbordsapplikationer och mobila applikationer(Introducing Visual Studio c2013). Alla utvecklingsverktyg använder samma IDE.

SQL Server Data Tools(SSDT) är ett tillägg till Visual Studio som används för att utveckla databaser. Det finns möjlighet att utveckla objekt i databaser och exekvera frågor mot databasen. SSDT erbjuder ett enklare verktyg för utveckling än SQL Server(SQL Server Data Tools (SSDT) c2013). Därför har SSDT använts för utvecklingen av databasen medans SQL Server har använts för testning och hantering av data.

2.5.1 Microsoft Visual Studio Lightswitch

LightSwitch är ett tillägg till Visual Studio som kan användas för att snabbt utveckla databasapplikationer. Det är en vidareutveckling utav Microsoft Access. Det inriktar sig både till mindre företag med begränsade utvecklingsmöjligheter och till större företag som vill bygga vidare på LightSwitchs grundfunktioner. Eftersom det går snabbt att utveckla i LightSwitch går det också bra att använda LightSwitch för att ta fram prototyper av applikationer som man sedan önskar vidareutveckla med andra verktyg(Krishnaswamy, J. 2011).

2.6 Nilex

Nilex är ett stödsystem för ärendehantering. Det erbjuder en centrerad lagring av alla ärenden vilket gör det enklare att undvika att data går förlorat. I en mindre centrerad lösning så är det möjligt att data glöms bort eller systemet som datan är lagrad på slutar användas. Nilex erbjuder också en plattform där handläggare kan bevaka nya ärenden som tillkommer. (*Nilex Enterprise*® c2013).

I det här projektet används Nilex ärendehanteringssystem som en källa för att importera incidenter som rör integration till Integrationskatalogen.

2.7 Den gamla Integrationskatalogen

Inom en stor koncern så kan det finnas flera hundra olika system. För att kunna hantera de många kopplingar som existerar mellan de olika systemen så behövs en effektiv och skalbar metod. Innan detta arbete påbörjades använde sig Ipendo Systems utav ett Excelark för att lagra dokumentation över sitt integrationsarbete. Problemet med Excelark är att de har begränsad förmåga att hantera stora mängder data. Det är också svårt att garantera att den data som lagras är korrekt. Med en databas så är det enkelt att hantera stora mängder data. Det finns också en möjlighet att införa restriktioner för att garantera att data som lagras är korrekt.

Kapitel 3

Kravspecifikation

3.1 Kravspecifikation

Följande är de krav som finns för Integrationskatalogen.

Grundläggande krav på databasen

- Databasmodellen ska vara så generell att den går att använda för flera olika typer av integrerade företagssystem. Det ska inte vara begränsad till en viss typ av integrationsplattform utan kunna tillämpas på olika former av integrerade system .
- Databasen ska vara kompatibel med Ipendo Systems metod för att beskriva integration.
- Det ska gå att definiera informationstyper, systembeskrivningar, interface och flöden i integrationskatalogen.
- Modellen bör implementeras i SQL-Server.

Sammankoppling med integrationsplattformar

- Det ska vara möjligt att importera data från integrationsplattformen Biztalk in till Integrationskatalogen.
- Det ska vara möjligt att sammankoppla beskrivna interface i Integrationskatalogen med artefakter från integrationsplattformen.

Incidenthantering

- Det ska finnas möjlighet att hantera incidenter rörande integration i Integrationskatalogen.
- Det ska finnas möjlighet att lagra klassificeringar av kända problem i Integrationskatalogen.
- Det ska vara möjligt att importera data från ett externt ärendehanteringssystem in till Integrationskatalogen.

Administrationsgränssnitt

- Utveckla en prototyp av ett gränssnitt för att enklare hantera integrationskatalogen.
- Gränssnittet bör implementeras i Microsoft Visual Studio LightSwitch.

3.2 Avgränsningar

Fokus för det här arbetet är att ta fram en generell databasmodell. Därför har vissa saker förenklats framförallt när det kommer till implementationen. Till exempel så har Microsofts produkter uteslutande använts för implementering av databasmodellen. Anledningen till detta är framförallt att det är de produkter som Ipendo Systems använder i sin verksamhet vilket har gjort att all mjukvara har funnits tillgänglig och varit färdig att använda. Vidare antas att Biztalkservern ligger på samma server som integrationskatalogen. Detta för att göra importen av data så enkel som möjligt.

Denna rapport behandlar inte heller systemintegration som ämne utan fokuserar på att utveckla ett system för att hantera dokumentation över integrationer.

Kapitel 4

Databasdesign

Det primära syftet med detta arbete är att ta fram en databas som kan användas för att lagra integrationsdata. Avsikten är att den ska vara generell för olika situationer och implementeringar. För utveckling har Microsoft SQL Server använts. Jag kommer att börja med att presentera en preliminär design av databasen. Därefter kommer jag att demonstrera ett ER-diagram (*Entity-Relationship*) av databasen. Därefter presenteras de slutgiltiga tabellerna som härletts från ER-modellen.

Jag har valt att utveckla databasen på engelska därför kommer jag för tydlighetens skull använda de engelska namnen på attribut och tabeller genom hela rapporten.

4.1 Preliminär design av databasen

Det första som gjordes var att analysera kraven och de egenskaper som var nödvändigt att beskriva i databasmodellen. Efter detta kunde följande egenskaper identifieras.

1. Den centrala delen i integrationskatalogen är interface. Dessa används för att sammankoppla det integrerade systemet. Ett interface är alltid kopplat till ett installerat system och skickar eller tar emot information. Ett interface som skickar information är av typen *Producer* (producent) och ett interface som tar emot information är av typen *Consumer* (konsument).
2. Det integrerade systemet består av flera mindre system. Det finns flera olika typer av system. Antingen är det ett system som ägs av företaget eller så kan det vara externa system som företaget hyrt in från ett annat företag. Det är också möjligt att det finns andra typer av system. Ett system kan vara installerat i flera instanser så det är bra att kunna skilja på dessa. Det är också intressant att veta vilken server systemet är installerat på, eventuella kontaktpersoner för installation, och vilken avdelning av företaget som använder installationen.
3. Det finns många typer av information som kan skickas inom det integrerade systemet. Det vore bra att kunna dela upp informationen i kategorier så att det är enklare att sortera bland gränssnitten.
4. Interface kan kopplas ihop till flöden och flöden kan sedan kopplas ihop till flödeskedjor. Vi behöver kunna lagra ordningen på flödena i flödeskedjan.
5. Varje interface svarar mot en struktur i det integrerade systemet. Det är intressant att kunna beskriva kopplingen mellan de något abstrakta interfacen och de faktiska portar som används för överföringen.
6. Det är möjligt att ett interface har flera olika extra egenskaper. Dessa egenskaper kan variera efter implementationen av det integrerade systemet och behöver nödvändigtvis inte vara samma för alla typer av integrerade system.
7. Alla IT-system råkar ibland ut för incidenter då något går fel. Vi vill kunna lagra dessa incidenter och knyta samman dem med de interface som drabbats. Då vi inte har intresse av

att utveckla ett komplett incidenthanteringssystem så kommer incidenterna att importeras från ett externt system.

8. Vi vill kunna lagra klassificeringar på kända problem så att när en ny incident kommer in i integrationskatalogen så kan man gå till listan över kända problem och hitta en möjlig lösning till den incidenten.

Från denna minimodell av systemet härledes 9 olika entitetstyper (Figur 4-1).

Interface
Name, InterfaceType, Information, SystemInstallation, FullDiff, Status, Documentation, Synonyms, EngineArtifact, {Incident}, {Property}

SystemInstallation
System(Name, SystemType, IntegrationInterface), Name, ContactPerson, Domain, IntegrationServer

Information
InformationSubGoup(InformationGroup), Name

Flow
Interface1, Interface2, Summary

FlowChain
{Flow}, Name

EngineArtifact
Interface, EngineArtifactType, Name, Data, Active

Property
PropertyType, Interface, Value

Incident
Id, Date, Description

Problem
ProblemType, {Incident}, Description, Cause, Solution, Workaround, Pattern

Figur 4-1 Preliminär design av integrationskatalogens databas

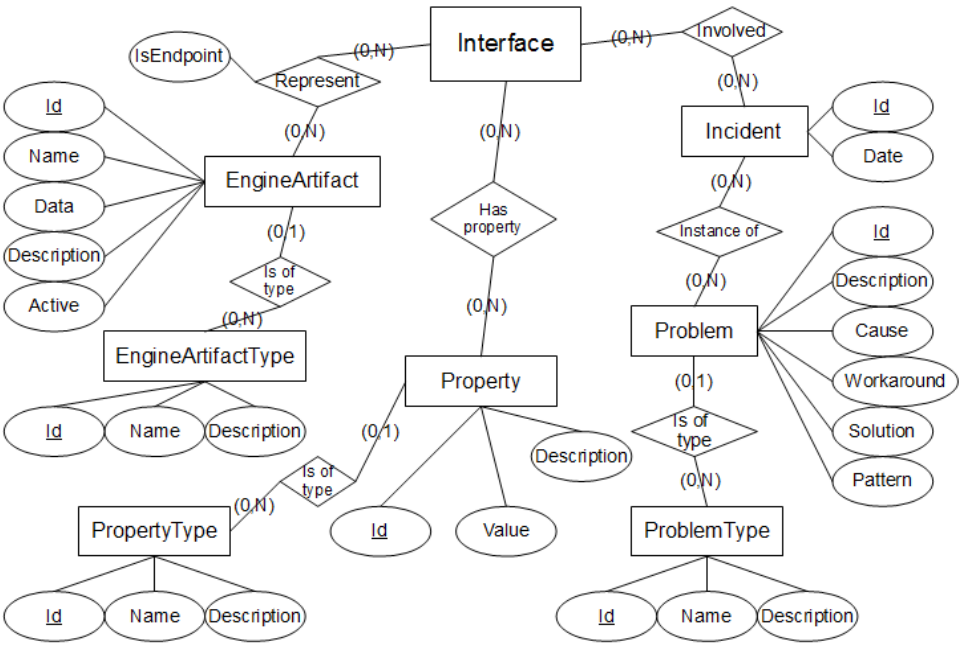
1. En entitetstyp Interface med attributen *Name*, *InterfaceType*, *Information*, *SystemInstallation*, *FullDiff*, *Status*, *Documentation*, *Synonyms*, *EngineArtifact*, *Incident*, *Property*. *Incident* och *Property* är flervärdesattribut vilket betyder att det kan finnas flera instanser av dem som relaterar till samma interface. *Name* är ett attribut som kan härledas från *InterfaceType*, *Information*, *SystemInstallation*.
2. En entitetstyp *Systeminstallation* med attributen *System*, *Name*, *ContactPerson*, *Domain*, *IntegrationServer*. *System* är ett sammansatt attribut som består av *Name*, *SystemType* och *IntegrationInterface*. Det är möjligt att *ContactPerson*, *Domain* och *IntegrationServer* också är sammansatta attribut. Vidare analys får avgöra detta.
3. En entitetstyp Information med attributen *InformationSubGroup* och *Name*. *InformationSubGroup* är ett sammansatt attribut innehållande *InformationGroup*. Detta är en metod för att göra det möjligt att dela in information i undergrupper och på så sätt göra det enklare att sortera de olika sorternas information.

4. En entitetstyp *Flow* med attributen *Interface1*, *Interface2*, *Summary*. En *Flow* lagrar två olika *Interface*, det är nödvändigt att *Interface1* är av typen *Producer* och *Interface2* är av typen *Consumer*. *Summary* är ett deriverat attribut härlett utav *Name* från de båda interfacen.
5. En entitetstyp *FlowChain* med attributen *Flow* och *Name*. *Flow* är ett flervärdesattribut eftersom en flödeskedja består av flera flöden.
6. En entitetstyp *EngineArtifact* med attributen *Interface*, *EngineArtifactType*, *Name*, *Data*, *Active*. *EngineArtifactType* kan vara ett samansatt attribut om det är viktigt att kunna beskriva de olika typerna av artefakter närmare.
7. En entitetstyp *Property* med attributen *PropertyType*, *Interface*, *Value*. Återigen kan *PropertyType* vara ett sammansatt attribut om det är viktigt att kunna beskriva de olika typerna av *Property* närmare.
8. En entitetstyp *Incident* med attributen *Id*, *Date*, *Description*. Det är troligt att det finns fler attribut till *Incident* men dessa beror på vilket ärehanteringsystem som används och är därför beroende på implementation.
9. En entitetstyp *Problem* med attributen *ProblemType*, *Incident*, *Description*, *Cause*, *Solution*, *Workaround*, *Pattern*. *Incident* är ett flervärdesattribut. *ProblemType* kan förlängas till ett sammansatt attribut om det krävs.

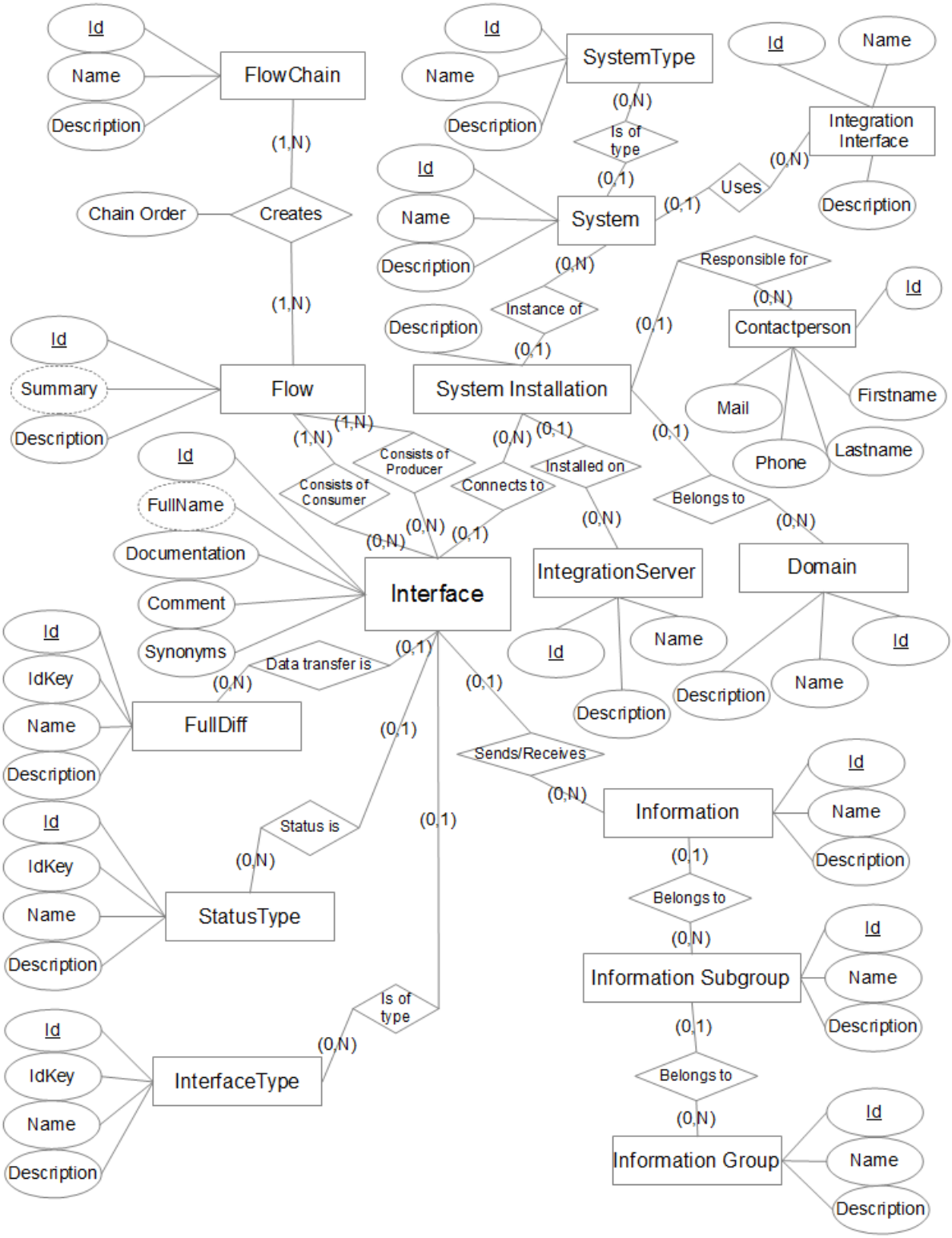
4.2 ER-Modell av databasen

Från den preliminära modellen konstruerades en ER-modell av databasen (Figur 4-2 och 4-3). Här går det att se förhållandet mellan entiteter och attribut samt relationerna mellan entiteterna (Elmasri, R., Navathe S. 2000). På grund av platsbrist så är diagrammet uppdelat i två delar.

Ett par saker är intressanta att uppmärksamma. Till att börja med så har flera entiteter fått ett ytterligare attribut nämligen *Description*. Detta för att möjliggöra för att lagra beskrivningar av entiteterna i databasen. I relationen mellan *Flow* och *FlowChain* lagras ytterligare ett attribut nämligen *ChainOrder*. Detta är för att kunna ordna flödena i flödeskedjan. *Incident* är inte tilldelat några attribut. Detta eftersom incidenter hämtas in från ett externt system och det därför kan variera beroende på ärehanteringssystemet vilka attribut som är intressanta för *Incident*.



Figur 4-2 ER-modell över databasen

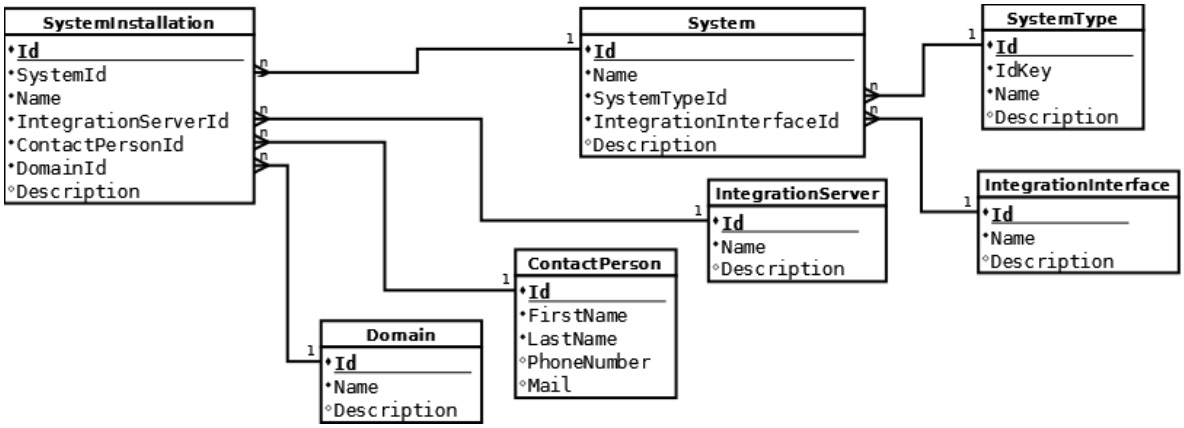


Figur 4-3 ER-modell över databasen

4.3 Databasensdelar

Utifrån ER-diagrammen i figur 4-2 och 4-3 kunde sedan den slutgiltiga databasen konstrueras. Databasen består av ett flertal olika element. För att skapa en hanterbar översikt har jag valt att dela in databasen i 8 olika delar. Varje del fyller ett specifikt syfte och är därför relevant att beskriva individuellt. En större översiktsbild över hela databasen går att se i appendix A.

4.3.1 System

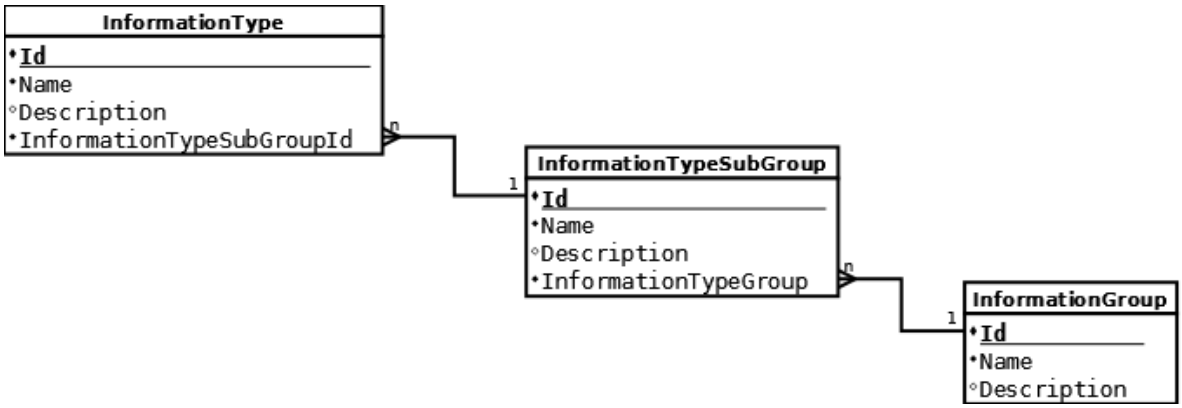


Figur 4-4 Beskrivning av strukturen för System

För att kunna beskriva systemen används en tredelad hierarki. Detta möjliggör skapandet av tydliga grupper av system. Figur 4-4 visar en översiktsbild av strukturen kring system så som det är implementerat i SQL Server. Den lägsta indelningen är *SystemType* vilket beskriver vilken typ systemet är av. Indelningen består av externa system och system som ägs och underhålls av företaget själva. Den andra nivån beskriver systemen. Varje system kan vara av en typ beskrivet av *SystemType*. Den tredje nivån är *Systeminstallation*. Varje *System* kan vara installerat på flera ställen och användas av olika delar i det integrerade systemet. Det är därför viktigt att kunna beskriva dem på ett korrekt sätt. *Systeminstallation* har även ytterligare ett par relationer knutna till sig. Varje *Systeminstallation* har en kontaktperson knuten till sig som kan kontaktas utifall att det uppstår ett problem med systemet eller om någon skulle ha några frågor rörande det specifika systemet. Det finns också en tabell för att hantera integrationsservrar. Företaget som äger det integrerade systemet har troligtvis flera olika servrar. Då kan det vara intressant att kunna se vilka system som ligger på vilken server. Detta kan framförallt vara användbart för felsökning. Ytterliggare finns en tabell för domän som kan knytas till systeminstallationerna. Detta är för att kunna dela upp större organisationer i mindre delar för enklare organisation.

4.3.2 InformationType

För att kunna kategorisera informationen som skickas inom systemet så är informationen indelad i tre nivåer. De tre nivåerna är *InformationTypeGroup*, *InformationTypeSubGroup* och *InformationType*. *InformationTypeGroup* är den lägsta nivån och erbjuder möjlighet att gruppera in information i generella grupper. *InformationTypeSubGroup* ger möjlighet att ytterligare dela in grupperna till mindre mer specifika grupper. *InformationType* är den högsta nivån och lagrar det faktiska informationsnamnet. Den här strukturen är visualiserad i figur 4-5.



Figur 4-5 Beskrivning av strukturen för informationstyper

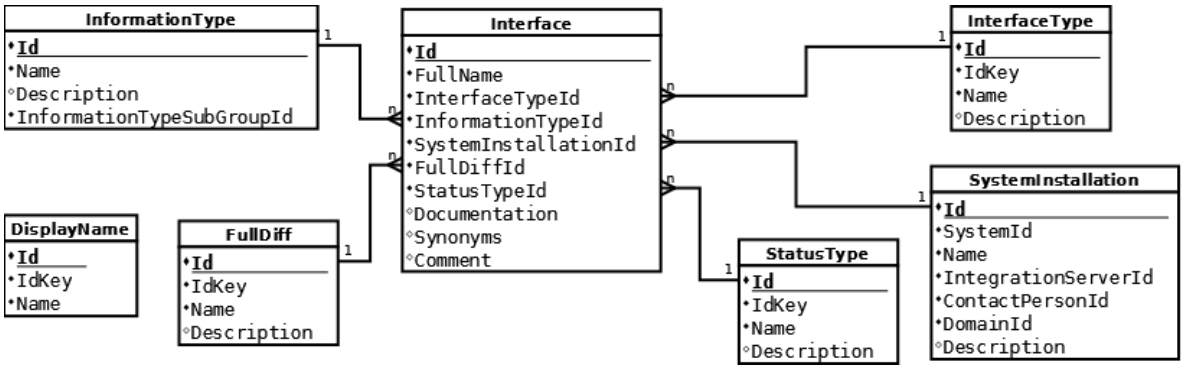
4.3.3 Interface

Interfacen är den centrala delen i integrationskatalogen. Ett interface är som nämndes i den teoretiska bakgrunden kategoriserat som en överföringspunkt för information till/från ett system. Därför innehåller tabellen för *Interface* referenser till *SystemInstallation* och *InformationType*. Det finns även en referens till tabellen *InterfaceType* som avgör om interfacet är av typen *Producer* eller *Consumer*. Detta visas i figur 4-6. Det är möjligt att ett system både producerar information och konsummerar information. För att uttrycka detta så är allt som behövs att definiera två *Interface* för det aktuella systemet, ett av typen *Producer* och ett av typen *Consumer*.

Utöver detta finns det även ett par extra tabeller som används för att ge ytterligare information om interfacet. *StatusType* används för att kategorisera den status interfacet för närvarande har. Exempel på möjliga status är aktiv, avvecklad eller inaktiv. *FullDiff* används för att avgöra hur omfattande den information som skickas är. *Full* betyder att all tillgänglig information skickas vid varje överföring medans *Diff* betyder att enbart den information som inte hittills har skickats kommer att skickas.

För att förenkla hanteringen av interfacen så finns det ett fält som kallas *FullName* med ett beskrivande namn som automatgenereras baserat på *SystemInstallation*, *InformationType* och *InterfaceType*. Formatet är ”*SystemInstallation.Name* Till/från *InformationType.Name*. ”Till” används om interfacet är av typen *Consumer* och ”Från” om interfacet är av typen *Producer*.

Eftersom databasen ska vara så generell som möjligt så finns det en speciell tabell för att hantera att användarna har olika önskemål på vilket språk som ska användas. I *DisplayName* lagras en nyckel med det engelska ordet i *IdKey*. Det namn som man vill ska visas i tex ett gränssnittet lagras i kolumnen *Name*. Denna uppslagning används när *FullName* ska genereras.

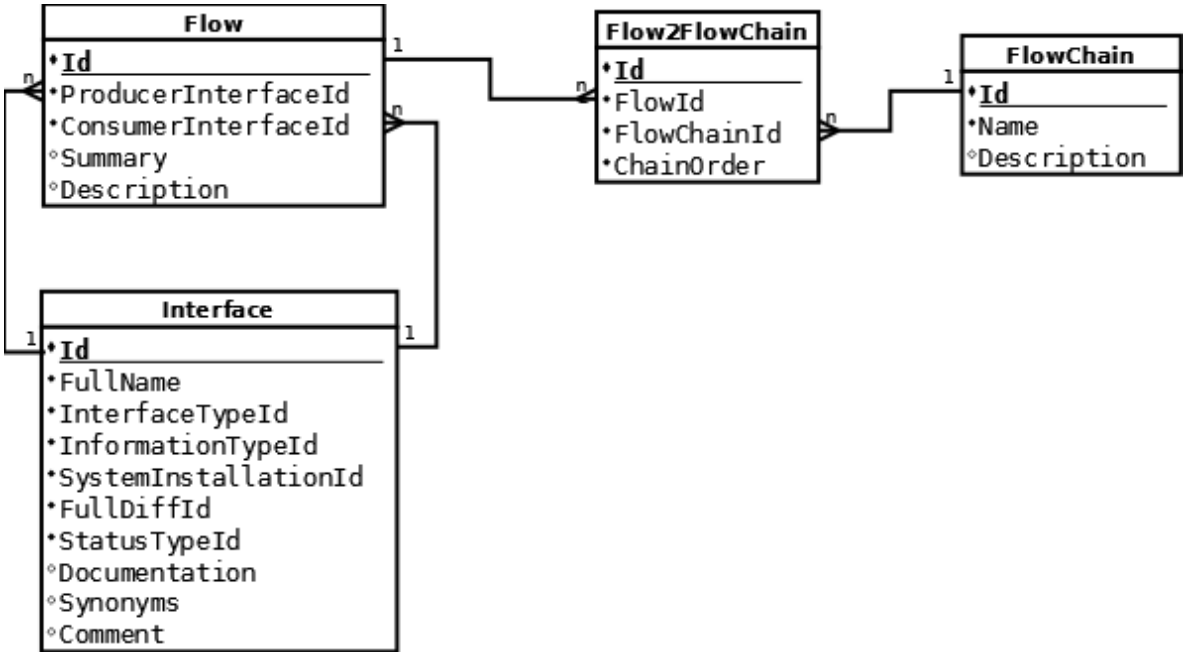


Figur 4-6 Beskrivning av strukturen för interface

4.3.4 Flow

Tabellen Flow innehåller referenser till två interface. Det finns restriktioner som säger att det första interfacet måste vara av typen *Producer* och det andra interfacet måste vara av typen *Consumer*.

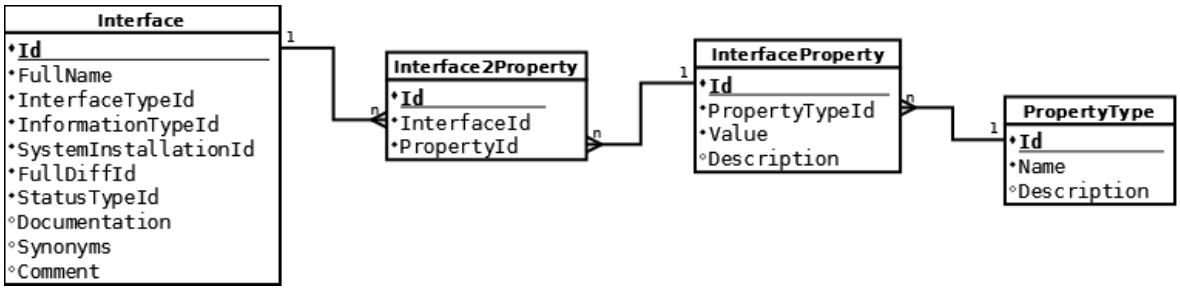
FlowChain används för att sammankoppla flera flöden till en kedja. Eftersom ett flöde kan innehålla flera interface och ett interface kan vara delaktigt i flera flöden så råder en många-till-många relation. Därför används en extra tabell för att länka ihop flödena och kedjorna. I länktabellen finns även ett fält som lagrar vilket ordningsnummer som flödet har i kedjan. Figur 4-7 visar hur den här strukturen är uppbyggd.



Figur 4-7 Beskrivning av strukturen för flöden

4.3.5 Property

För att beskriva extra egenskaper (*Property*) i databasen har en tabell för extra egenskaper skapats. Avsikten är att när man önskar skapa en ny egenskap till Interface så börjar man med att definiera en typ som kan användas för att identifiera egenskaperna. Säg att det för varje Interface kan finnas en mapp med backup-data. Denna mapp vill man kunna lagra i Integrationskatalogen så att den är enkel att hitta om den skulle behöva användas. Vi börjar då med att definiera en rad i *PropertyType* som vi väljer att namnge Backup-mapp. Därefter kan vi skapa fler rader i *InterfaceProperty* där man anger vilket Interface som är kopplat till den här egenskapen, vilken typ av egenskap det rör sig om (i vårt fall ”Backup-mapp”), vilket värde den här egenskapen har (i vårt fall var mappen som innehåller Backup-data är placerad) samt en kort extra beskrivning om man har ytterligare information som inte passar i de andra kolumnerna. Figur 4-8 illustrerar hur detta är uppbyggt i databasen.



Figur 4-8 Beskrivning av strukturen för propertys

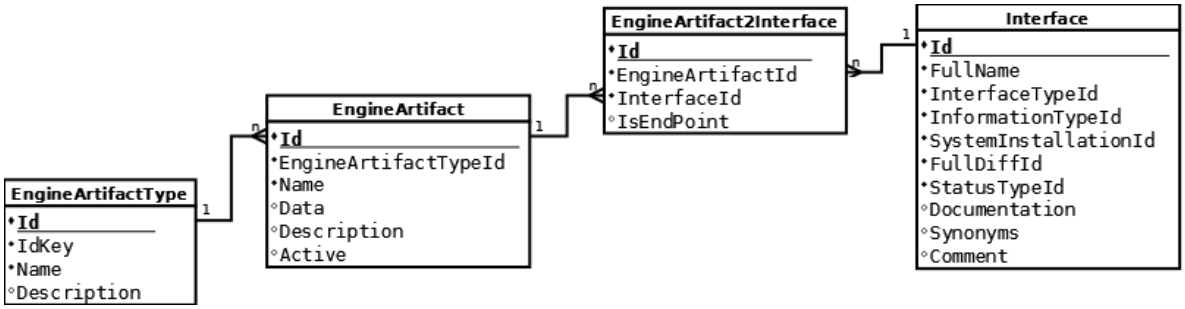
4.3.6 Postdeploy-fil

Databasen innehåller ett flertal tabeller vars främsta syfte är att försäkra att rätt typ av data finns i tabellerna. Exempel på detta är *InterfaceTypeId* i *Interface* som måste referera till en rad i tabellen *InterfaceType*. Denna data är relativt statisk och för att databasen ska gå att använda så behöver de relevanta tabellerna vara fyllda med rätt data.

För att förenkla uppsättningen av databasen på en ny plats så har jag utvecklat ett antal script med syfte att fylla databasen med basdata. Det finns även en uppsättning skript för att fylla övriga tabeller med data från en exempeldatabas. Scripten är av typen MERGE. I skriptet finns en tabell definierad med den data som ska infogas i databasen. Sedan sammanfogas denna tabell med tabellen i databasen som ska innehålla datan. En effekt av sammanfogningen är att om det redan finns en rad med samma id i databasen som i tabellen så kommer raden i databasen endast uppdateras. Om det inte finns någon rad med rätt id så läggs en ny rad till.

4.3.7 Biztalk-integration

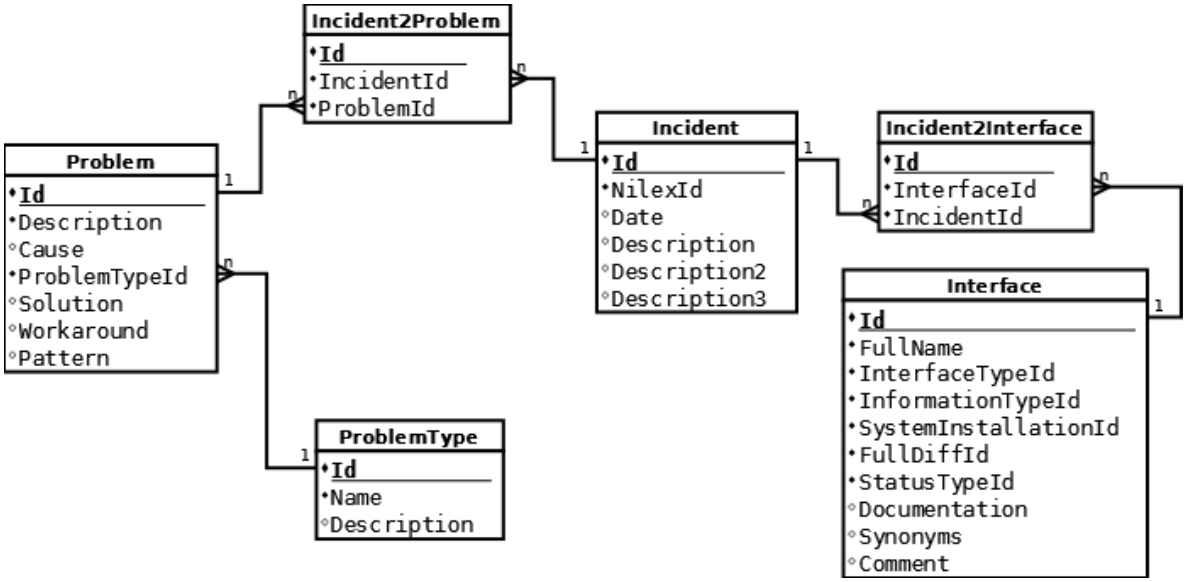
Det finns ett antal olika integrationsplattformar som används för att skapa funktionella integrerade system. Databasen innehåller därför funktionalitet för att länka ihop de objekt som skapas i integrationsplattformen med interfacen i integrationskatalogen. Strukturen för hur detta är uppbyggt visas i figur 4-9. Kopplingen i integrationsplattformen klassificeras som en *EngineArtifact*. En artefakt kan vara av typen *Sendport* eller *Receiveport*. Dessa korrelerar mot *Producer* och *Consumer* i Interface. För att enkelt sammankoppla integrationskatalogen med plattformen så finns det funktionalitet för att hämta in *EngineArtifacts* från integrationsplattformen Biztalk.



Figur 4-9 Beskrivning av strukturen för Biztalk-integration

4.3.8 Incidenthantering

De flesta företag har ett eget system för att hantera supportärenden som uppstår. Därför finns det stöd i databasen för att hämta in ärenden från sådana databaser. För närvarande stöds enbart hämtning från Nilexdatabaser men scriptet kan lätt omarbetas för att hantera andra typer av databaser. Fördelen med att använda Nilex är att det är ett centrerat ärendehanteringsystem vilket gör att det kan innehålla alla incidenter organisationen har dokumenterat. Genom att importera incidenter därifrån hellre än att skapa ett eget system för ärendehantering så försäkrar man att centraliseringen av ärendehantering bibehålls.



Figur 4-10 Beskrivning av strukturen över incidentlagring

Figur 4-10 visar hur strukturen för att hantera incidenter ser ut. *Incident* är den tabell som lagrar själva incidenten. Denna fylls upp med data som hämtas från extern källa. I vårt fall från en Nilexdatabas. Det är möjligt att koppla ihop incidenter med interface, detta för att kunna se historiskt vilka *Interface* som har haft problem samt vilka incidenter som rör vilka *Interface*. En *Incident* kan involvera flera interface men ett interface kan också ha råkat ut för många incidenter. Det rör sig alltså om en många-till-många relation och därför finns tabellen Incident2Interface för att länka mellan *Interface* och *Incident*.

En incident beskriver bara en individuell händelse som inträffat. För att kunna hitta lösningar till incidenter så införs en tabell *Problem* vilket innehåller beskrivningar av kända problem som inträffat i det integrerade systemet. I denna tabell lagras en beskrivning av problemet, orsak, vad det är för typ av problem men också möjliga lösningar och även temporära lösningar som kan nyttjas till problemet är ordentligt löst. Ytterliggare en sak som lagras i denna tabell är en kolumn med nyckelord för varje problem. Dessa skulle kunna användas för att hitta möjliga lösningar till incidenter genom att matcha dessa nyckelord mot beskrivningarna av incidenter.

Det råder en många-till-många relation mellan *Problem* och *Incident* eftersom ett problem kan röra flera incidenter och en incident kan röra flera problem. Därför finns det en tabell *Incident2Problem* för att länka mellan dessa tabeller.

4.3.9 Views

I databasmodellen används också tre olika vyer.

4.3.9.1 ConsumerViewoch ProducerView

Detta är två enkla vyer som kan användas för att skilja på konsument- och producent-interface. Primärt syfte är att ha ett enkelt sätt att plocka fram alla interface av en specifik typ.

4.3.9.2 EngineArtifactUncategorizedView

Kopplingen mellan interface och artefakter från integrationsplattformen kräver ett manuellt arbete där en utvecklare med kunskap om systemet kopplar ihop artefakter med relevant interface. För att underlätta denna sammankopplingen finns en vy som visar vilka artefakter som ännu inte är kopplade till något interface.

4.3.10 Automatisk hämtning av data från Biztalk och Nilex databaser

Kopplingen av artefakter från integrationsmotorn och incidenter med fel kräver data från tabeller i externa databaser. För att samla in denna data har det utvecklats script som utför en automatisk hämtning av relevant data från den externa databasen. Dessa script är sparade i databasen som *Stored Procedures.* Detta gör det möjligt att sedan schemalägga dem så att de kan bli utförda med ett bestämt intervall. Genom att hämta in data med ett regelbundet intervall så slipper man underhålla samma data lagrad i två databaser vilket gör att det blir enkelt att se till att den lagrade datan alltid är korrekt uppdaterad. Risken för datakonflikter är därför aktivt begränsad.

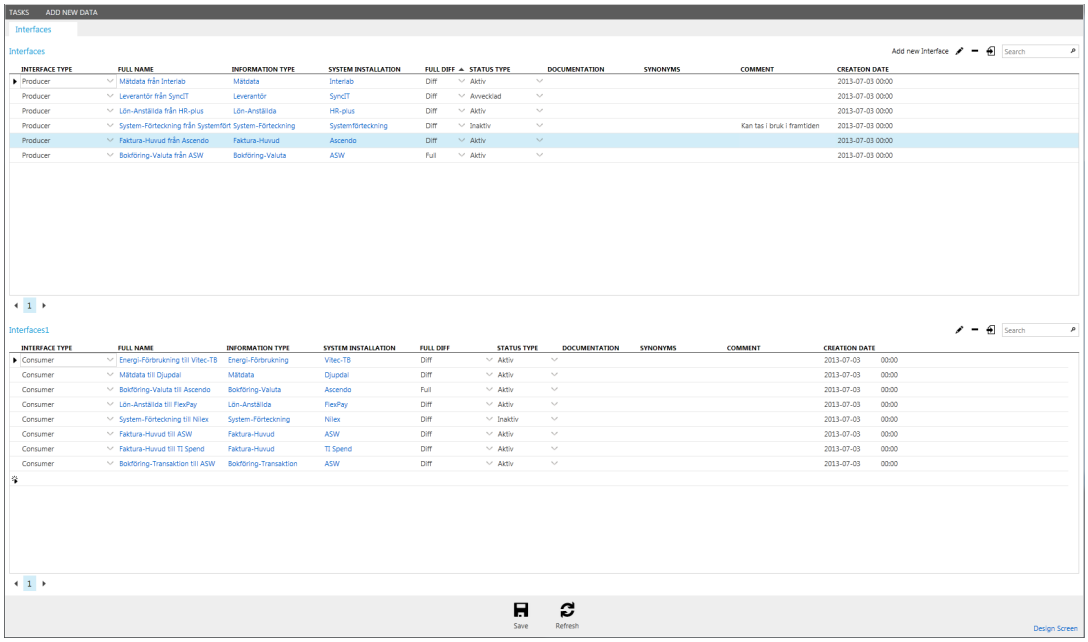
Kapitel 5

Administrationsgränssnitt

För att enklare kunna hantera och arbeta med databasen har jag utvecklat en prototypapplikation som kan användas för att hantera data i databasen. Detta ska göra det möjligt för den mindre tekniskt kunniga användaren att kunna arbeta med databasen. Applikationen är utvecklad i Microsoft LightSwitch. Applikationen är uppbyggd kring ett antal skärmar som man kan navigera mellan i en meny.

5.1.1 Fönsteri gränssnittet

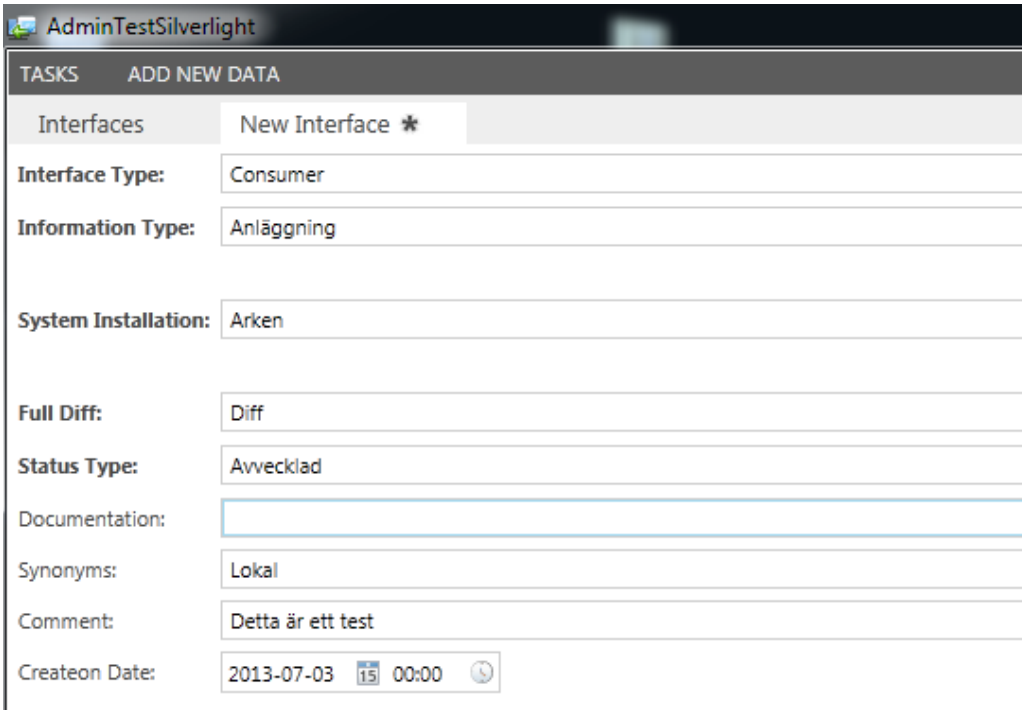
- **Interfaces**
  Detta är en översiktssida för att hantera gränssnitt. Det finns två olika tabeller med interface. En för interface av typen *Producer* och en för typen *Consumer*.



Figur 5-1 En bild som visar en listning av de interface som är lagrade i databasen. Interfacen är uppdelade i Producers och Consumers.

- **Interfaces detail**
  Varje Interface har en egen översiktsfönster för att skapa översikt över det specifika interfacet. Det går att se vilka artefakter, flöden, extra egenskaper och incidenter som är kopplat till interfacet.
- **Flows**
  Ett fönster som används för att överblicka alla flöden i systemet. Ger en översiktsbild med relevant information från de båda interface som utgör flödet.

- **Information Type**
  Ett enkel översiktsfönster för att kunna överblicka de olika informationsindelningar som finns i systemet. Varje informationstyp listas tillsammans med den grupp och subgrupp som de tillhör.
- **System installation**
  Ett översiktsfönster för att hantera systeminstallationer som finns i systemet. Den visar vilka system som finns installerade och även vilket domän systemet tillhör, vem som är ansvarig kontaktperson och på vilken server systemet är installerat.
- **Flow Chain List**
  Detta är en lista för att överblicka de flödeskedjor som finns i systemet. Systemets flödeskedjor listas till vänster och till höger så listats de flöden som är sammanlänkade med den markerade kedjan. Flödena sorteras baserat på ordningsnummer vilket gör det enkelt att överblicka hur kedjan är uppbyggd.
- **Engine Artifact**
  Ett fönster för att hantera artefakter som har importerats från en integrationsplattform. Finns möjlighet att enkelt koppla ihop de olika artefakterna med det interface som de motsvarar.
- **Incidents**
  Detta är ett fönster för att hantera incidenter som inträffar i systemet. Här listas de incidenter som finns till vänster och till höger kan man överblicka vilka interface som är inblandade samt vilka kända problem som länkar till den valda incidenten.
- **Add data**
  För att enkelt kunna lägga till data finns det ett antal fönster som kan användas i samband med insättning av data i databasen. Dessa är ordnade under menyfliken ”*Add Data*”. Ett exempel på hur en skärm från gränssnittet visas i figur 5-1 som visar hur tilläggningen av nya interface ser ut.



Figur 5-2 En del av Administratörsgränssnittet. Här en bild på hur tilläggning av data kan gå till

Kapitel 6

Diskussion och slutsats

Den här rapporten har beskrivit utvecklingen av ett system för att hantera dokumentation över systemintegration. Detta kapitel kommer att analysera kraven som ställdes på integrationskatalogen i kapitel 3. I det här kapitlet kommer jag också diskutera hur modellen ska kunna användas, vilka möjligheter det finns att utveckla den samt hur man skulle kunna använda modellen tillsammans med andra externa applikationer.

6.1 Analys av kraven

Av de generella kraven så är det framförallt frågan om databasmodellens generalitet som är svår att bekräfta. Under arbetet så har flera åtgärder tagits för att försäkra att modellen är så generell som möjligt. Strukturen med interface, system, information och flöden antas vara så generell att den alltid är relevant för att beskriva integrerade system. Men eftersom implementationen av de integrerade systemen kan se olika ut och ställer olika krav på vad som måste gå att beskriva i Integrationskatalogen så finns det möjlighet att lägga till ytterligare egenskaper till systemet. Tabellerna *InterfaceProperty* och *PropertyType* ger denna funktionalitet. Detta borde garantera att integrationskatalogen håller en tillräckligt hög nivå av generallitet för att kunna användas i flera olika former av integrerade system.

Det är bekräftat att databasmodellen är kompatibel med Ipendo Systems metoder genom att data från en av deras kunders integrationsplattform har kunnat infogas i Integrationskatalogen. Möjligheten att beskriva system, information, interface och flöden finns i modellen. Databasmodellen är implementerad i SQL Server.

Sammankoppling med en integrationsplattform finns och är verifierat genom tester med data från en av Ipendo Systems kunders implementationer av Biztalk.

Hantering av incidenter finns i databasmodellen och det finns även möjlighet att lagra lösningar på incidenter. Funktionalitet har blivit testat med hjälp av data från en implementation av Nilex som en av Ipendo Systems kunder använder sig av.

Ett administratörsgränssnitt finns också och har stöd för tilläggning av data. Detta är utvecklat i LightSwitch.

Alla krav har alltså blivit uppfyllda och projektet får då anses som avklarat

6.2 Reflektion över arbetet

Jag är överlag nöjd med arbetsgången i det här projektet. Vissa förutsättningar hade kanske varit bättre, tex hade min erfarenhet av utvecklingsverktygen kunnat vara bättre.

Ett återkommande problem jag hade var att LightSwitch vid upprepade tillfällen utvecklade komplicerade fel som var svåra att lösa. Detta kunde härledas till att databasen utvecklades samtidigt som Lightswitch-applikationen vilket gjorde att databasen som Lightswitch-applikationen använde

ständigt förändrades. Detta verkade ibland orsaka interna problem för Lightswitch vilket gav svårlösta kompileringsfel. Ett bättre alternativ hade varit att börja med att designa databasen och sedan gå över till att designa LightSwitch-applikationen. Det hade förmodligen sparat projektet flera onödiga timmar i felsökning.

6.3 Produktens plats I organisationen

Denna databasmodell har syftet att användas för att lagra dokumentation kring integrationskopplingar i ett större IT-system. Integrationskatalogen kommer framförallt att användas som ett internt system för att hjälpa företaget hantera sitt integrationsarbete. Tanken är att det ska fungera som ett nav för integrationsarbetet. Förutsättningen för att använda integrationskatalogen är att företaget eller organisationen är tillräckligt stor för att ha en omfattande integration mellan ett flertal system. För mindre företag med ett mindre IT-system så är databasmodellen förmodligen något överflödig.

6.4 Möjligheter att utveckla applikationen

Nedan följer ett par möjliga utvecklingar och tillägg till databasen. Det finns självklart ytterligare möjligheter att utveckla applikationen som inte är inom ramen för den här rapporten.

6.4.1 Implementera uppdateringshistorik

En av företaget efterfrågad funktion till Integrationskatalogen är möjligheten att samla statistik över förändringar i databasen. Det finns i nuläget inget uniformt sätt att hantera historik över ändringar i databasen. SQL server har två inbyggda verktyg för detta, Change Tracking och Change Data Capture(CDC). Båda kan användas för att avgöra vilka förändringar som har skett i databasen. Detta innebär ett stort steg framåt jämfört med det tidigare förfarandet att använda kolumner för tidstämplar och extra triggers vilka var både resurskrävande och komplexa att implementera (Thernström, T., Weber, A., Hotek, M. 2009).

Change Tracking är en lättvikts-applikation som sparar vilka rader och kolumner som har blivit förändrade i databasen. Det finns möjlighet att avgöra om det är ett INSERT-, UPDATE- eller DELETE-query som har utförts på den aktuella tabellen. CDC erbjuder samma funktionalitet som Change Tracking men ger också möjlighet att se vilken information som var lagrad innan förändringen skedde.

Båda applikationerna kan aktiveras utan att behöva göra några förändringar i tabellerna som ska utnyttja loggningsfunktionen. Change Tracking har dock en lägre overhead än CDC men erbjuder inte lika många funktioner. Det blir därför en fråga om vilken mängd information som behövs lagras. Om det för den aktuella implementationen enbart är nödvändigt att veta vilka rader som blivit ändrade i tabellen så borde Change Tracking uppfylla de behov man kan tänkas ha. Men om det är viktigt att veta exakt vilka förändringar som har skett i databasen så är CDC ett lämpligare val.

6.4.2 Automatisk hämtning av Biztalk och Nilex data

En begränsning med den nuvarande utformningen är att de databaser som innehåller den externa databasen måste vara lokaliserade på samma server som integrationskatalogen. Det finns möjlighet att möjliggöra hämtningar utav data även från externa databaser. Den enklaste metoden vore att utföra en

länkning av de båda databaserna. Men att implementera detta bedömdes inte vara inom ramarna för det här projektet.

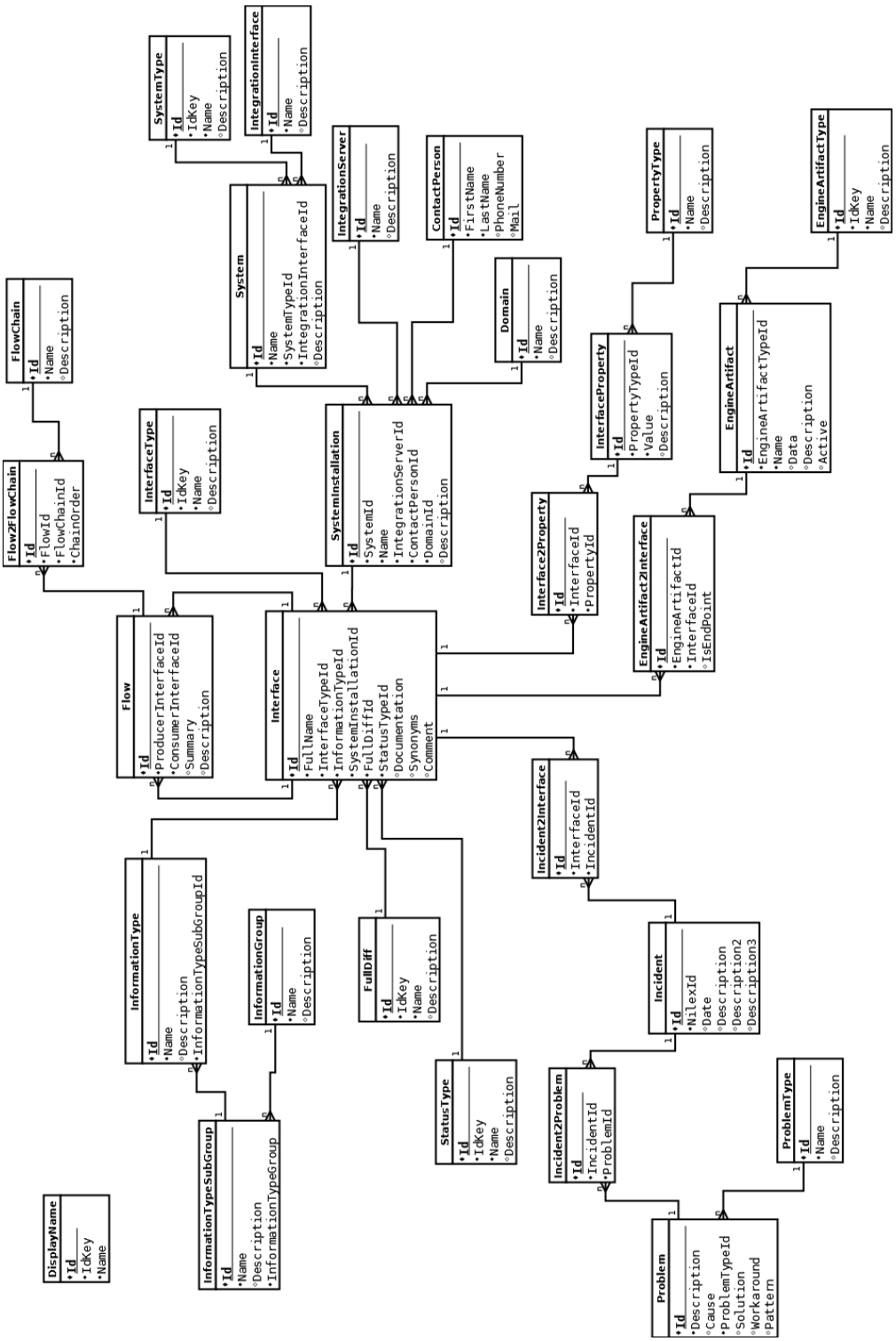
6.4.3 Förslag på lösningar till incidenter

I den nuvarande implementationen sker idag en sammankoppling av incidenter med problem. Det är upp till administratören att bestämma vilken incident som korresponderar mot vilket problem. Ett sätt att underlätta för administratören vore om han/hon kunde få förlag på vilka problem som skulle kunna ha orsakat den aktuella incidenten. Det finns idag en kolumn i tabellen *Problem* kallad *Pattern* som har som syfte att lagra en sträng med nyckelord som kan göra det enkelt att söka bland de kända problem som existerar i databasen. Det finns för närvarande ingen sådan implementation så ett möjligt steg att utveckla integrationskatalogen vore att inkludera detta.

Appendix A

Databasdiagram

A.1 Översiktsbild över databasen



Referenser

Cummins, F. (2002): Enterprise integration : An Architecture for Enterprise Application and Systems Integration. New York: Wiley. ISBN: 978-0-471-40010-3. 468 s.

Elmasri, R., Navathe S. (2000): Fundamentals of Database Systems, 3rd Edition. Boston: Addison-Wesley Longman. ISBN 0-201-54263-3. 955s.

Groff, J., Weinberg, P., Oppel, A. (2009): SQL The Complete Reference, 3rd Edition. McGraw Hill Professional. ISBN: 0071592563. 912 s.

Krishnaswamy, J. (2011): *Microsoft Visual Studio LightSwitch Business Application Developmen*t. Birmingham: Packt Publishing Ltd. ISBN: 1849682879. 384 s.

Krishnaswamy, J. (2011): *Microsoft Visual Studio LightSwitch Business Application Developmen*t. Birmingham: Packt Publishing Ltd. ISBN: 1849682879. 384 s.

Larson, B. *Microsoft SQL Server 2008 Reporting Services*. New York : McGraw-Hill. 2009

Rockoff, L. (2011): *The Language of SQL*. Boston: Course Technology PTR. ISBN: 1435457528.

Thernström, T., Weber, A., Hotek, M. (2009): *Microsoft SQL Server 2008- Database Development*. Washington: Microsoft Press. 456 s.

Elektroniskakällor:

Introducing Visual Studio (c2013): *Microsoft Developer Network*. Hämtad 2013-08-15 från: http://msdn.microsoft.com/en-us/library/fx6bk1f4%28v=vs.90%29.aspx

Nilex Enterprise®. (c2013). Hämtad: 2013-09-09 från: http://www.nilex.se/1/sv/nilex-enterprise/nilex-enterprise.php

O'Brien, R. (2008): HubPages: *Systems Integration Explained*. Hämtad 2013-08-10 från: http://russellobrien.hubpages.com/hub/Systems-Integration-Explained

Rouse, M. (2006): SearchSQLServer: *SQL Server*. Hämtad 2013-08-15 från: http://searchsqlserver.techtarget.com/definition/SQL-Server

Shanmugan, A. (2006): Code Project: *Explaining the BizTalk Architecture to your Grandma*. Hämtad 2013-08-15 från: http://www.codeproject.com/Articles/12854/Explaining-the-BizTalk-Architecture-to-your-Grandm

SQL Server Data Tools (SSDT) (c2013): *Microsoft Developer Network*. Hämtad 2013-08-15 från: http://msdn.microsoft.com/en-us/library/hh272686%28v=vs.103%29.aspx

SQL Views (1999): *w3schools.com*. Hämtad 2013-08-14 från: http://www.w3schools.com/sql/sql_view.asp

Johan Filipsson **Integrationskatalog- teknisk dokumentation av ett integrerat system**