Sammanfattning

Ett ärendehanteringssystem är ett komplett system vars mål är att effektivisera och koordinera processer av olika slag. Ett exempel på ärendehantering är försäkringsbolag som använder sig av detta för att snabba upp och kontrollera processen för hantering av skadeärenden. I ett ärendehanteringssystem finns en *ärendehanteringsmotor* som sköter flödet av ärenden, vilket bland annat innebär ansvar för att ärenden hamnar hos rätt person.

Examensarbetets syfte är att utveckla en liten och enkel ärendehanteringsmotor, som ändå är generell nog att fungera i olika kontexter. Ärendehanteringsmotorn levereras i form av en komponent färdig att användas vid utvecklandet av ärendehanteringssystem. Genom att återanvända komponenten och endast behöva anpassa den till det nya systemet är tanken att mycket tid sparas vid utvecklandet.

Rapporten behandlar utvecklingsprocessen från identifiering av ärendehanteringsmotorns roll och kravanalys till arkitekturella beslut. Alternativa lösningar och framtida utvecklingsmöjligheter har diskuterats. Resultatet blev en färdig ärendehanteringsmotor som uppfyller de grundläggande krav som ställdes på komponenten.

Innehåll

1 Inledning

I det här kapitlet presenteras bakgrunden till examensarbetet, dess syfte, de uppgifter som ska lösas och metoden som har använts.

1.1 Ibitec

Ibitec, eller ”Internet Business Intelligence Technology”, heter företaget som examensarbetet har utförs på. Det startades 1999 och har utvecklats till att bli ett medelstort teknikorienterat IT-konsultbolag med cirka 60 anställda. Ibitec är främst verksamma i Linköping men har också kontor i Stockholm och Göteborg. Företaget marknadsför sin verksamhet inom fyra olika affärsområden:

- Business Solutions (Verksamhetslösningar)
- E-business (Webbaserade verksamhetslösningar)
- Development Partner (Utvecklingspartner och Konsulttjänster)
- Education & Infrastructure (Utbildning och Teknikkonsulttjänster)

Ibitec levererar helhetslösningar inom många applikationsområden vilka utvecklas med tjänstebaserad arkitektur. Den typen av arkitektur utgår ifrån att behovet av snabba förändringar kommer att öka och att kraven på att hålla kostnader nere kommer att kvarstå. Verksamhetslösningarna är komponentbaserade, vilket innebär att de skapas med hjälp av löst kopplade komponenter som samverkar under samma plattform. Främsta anledningen till detta arbetssätt är att kraven på flexibilitet hela tiden ökar och att det blir kostnadseffektivt att återanvända mjukvara. Företaget branschanpassar lösningarna mot hälso & sjukvård, kommuner, tillverkande industri, fastighetsbolag och offentlig sektor.

Ibitec driver också utbildning inom teknik och utvecklarkurser, Microsoft Business Solutions – Axapta och projektledning.

1.2 Bakgrund

Människan har i alla tider sett fördelen med att inrätta system för att fördela arbetsuppgifter. Varje uppgift måste utföras av den person som har bäst kompetens för den specifika uppgiften. Historien visar många exempel på olika sätt att hantera och styra arbetsfördelning. Munkarna i medeltidens kloster hade alla en viktig del i konstruktionen av skrifter och böcker. Abboten fördelade arbetsuppgifterna så att en munk skrev texterna, en annan mer konstnärlig munk illustrerade boken och en tredje hade hand om inbindningen av boken, Plesums (2002).

Vad munkarna egentligen ägnade sig åt var ärendehantering, på engelska kallat workflow. Ett exempel från nutid är försäkringsbolag som använder sig av ärendehantering för att snabba upp och kontrollera processen för hantering av skadeärenden. Ett ärendehanteringssystem är ett komplett

system vars mål är att effektivisera och koordinera processer av olika slag. I ärendehanteringssystemet finns en ärendehanteringsmotor som sköter flödet av ärenden, vilket bland annat innebär ansvar för att ärenden hamnar hos rätt person.

Behovet av datorbaserade ärendehanteringssystem ökar ständigt i takt med att fler och fler företag och myndigheter söker IT-baserade organisationslösningar. För konsultbolaget Ibitec, där examensarbetet har utförts, innebär det mer utvecklingstid då det ofta krävs ett skräddarsytt system som är anpassat till den rutin och de behov som finns hos beställaren. Om utvecklingstiden kan kortas ner genom att återanvända källkod i form av en komponent blir nya lösningar billigare att ta fram.

1.3 Syfte

Syftet med examensarbetet är att ta fram en ärendehanteringsmotor i form av en komponent som uppfyller Ibitecs krav på produkten. Denna komponent ska kunna utnyttjas i ett större sammanhang där den fungerar som en del i ett ärendehanteringssystem.

Fokus ska ligga på att göra ärendehanteringsmotorn liten och enkel men ändå generell. Den ska fungera i olika kontexter som till exempel dokument, post, telefonsamtal och fakturor.

Examensarbetet sträcker sig över hela utvecklingsprocessen. Det innebär att kravbilden ska sammanställas, en undersökning om vilka olika alternativa lösningar som finns ska utvärderas och lösningen ska designas och implementeras.

1.4 Problemställningar

Examensarbetet kommer att behandla följande områden:

Identifiera ärendehanteringskomponentens roll i ett större sammanhang
Som ett första steg i utvecklingen kommer ärendehanteringsmotorns roll att identifieras.

Formulera krav på ärendehanteringskomponenten
För att få en klar bild av vad som ska utföras kommer en kravbild att tas fram. Denna kommer att behandla de aspekter av ärendehantering som ärendehanteringsmotorn ska klara av samt de tekniska krav som ställs på produkten.

Utvärdera alternativa lösningar
Utifrån kravbilden kommer de olika lösningsalternativ som finns att utvärderas.

Ta fram en design och implementera denna
Kravbilden och lösningsalternativet som valts fungerar som utgångspunkt för designen. Denna kommer att utformas i två steg där först arkitekturen

tas fram, vilken en detaljerad design kan utgå ifrån. När designen är klar återstår implementeringen.

Utvärdera produkten
Resultatet av implementationen kommer att utvärderas och framtida förbättringar kommer att presenteras.

1.5 Avgränsningar

Det ingår inte i examensarbetet att utveckla ett komplett ärendehanteringssystem utan enbart själva ärendehanteringsmotorn. Myndigheter måste rätta sig efter offentlighetsprincipen. Det innebär att alla handlingar in- och utgående handlingar måste registreras. Denna process kallas diarieföring. Examensarbetet kommer inte heller att behandla diariesystem hos myndigheter då de juridiska aspekterna av detta är för omfattande.

1.6 Metod

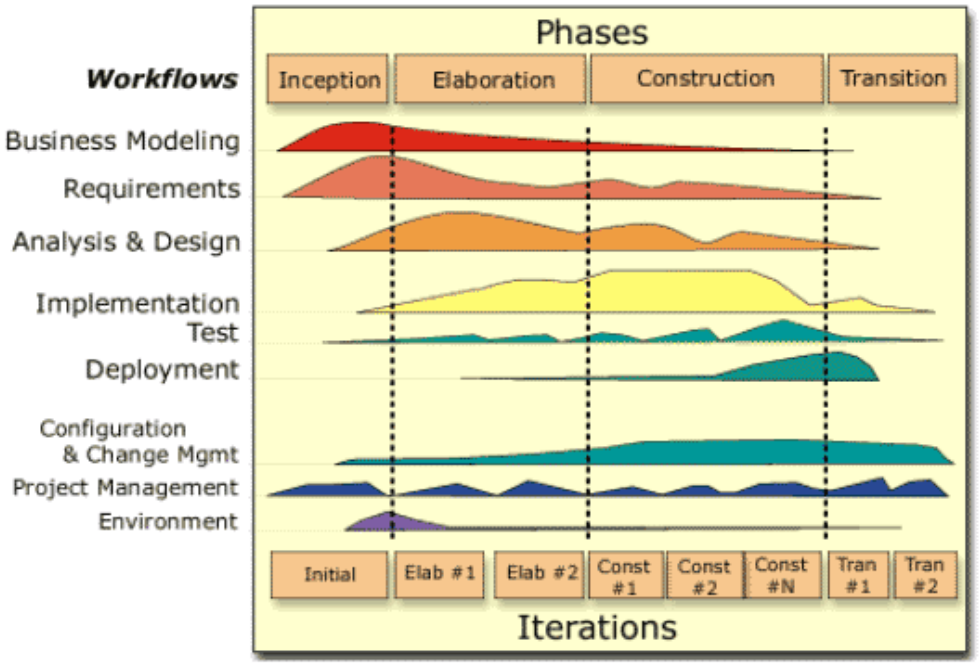
Företaget Ibitec där examensarbetet ska utföras vill att det ska bedrivas enligt samma systemutvecklingsmodell som de själva använder i sina utvecklingsprojekt. Deras systemutvecklingsmodell är en anpassning av RUP, detta därför att RUP är alldeles för omfattande för mindre utvecklingsprojekt. I detta avsnitt beskrivs kort vad RUP är, hur det kom till och hur det används. Därefter redovisas hur RUP kommer att tillämpas på examensarbetet.

1.6.1 RUP (Rational Unified Process)

RUP är en mycket omfattande modell för systemutvecklingsprocesser. Den är framtagen av och tillhör det amerikanska företaget Rational software, som ägs av IBM. Det finns en mängd verktyg framtagna av Rational software som underlättar användningen av RUPs olika delar. För mer information om detta se IBM (2004).

RUP utvecklades i början av 1990-talet utifrån tankarna om objektorienterad design. Starkt förknippad med RUP är modelleringsspråket UML (Unified Modeling Language) och dess diagram. RUP är egentligen inte en process utan ett ramverk för en process. Den konkreta processen som ett projekt följer uppstår när RUP konfigureras. Modellen är mycket omfattande och anpassad för att kunna appliceras på mycket stora utvecklingsprojekt. Därför skapar ofta mindre företag en nerbantad version av RUP som passar för mindre projekt.

Utvecklingsprocessen enligt RUP kan åskådliggöras med hjälp av nedanstående figur.



Figur 1. Utvecklingsprocessens olika faser i RUP.

Till vänster i figuren presenteras de olika processerna som pågår parallellt under projektets gång. Diagrammet visar hur mycket resurser som läggs ner på respektive process över tiden. För att få struktur i planeringen är tidsaxeln indelad i faser, som i sin tur är indelade i iterationer. Nedan presenteras de olika faserna.

Förberedelse (Inception)
Under förberedelsefasen initieras projektet. Syftet är i princip att komma överens om vad projektet ska åstadkomma. Fokus ligger på att visualisera och förstå de mest centrala kraven. Utifrån dessa kan systemets komplexitet och omfattning identifieras och projektets risker kan tas fram. Vidare ska användningsfallen prioriteras och projektgruppen ska bestämma vilka användningsfall som ska implementeras under etableringsfasen.

Etablering (Elaboration)
Under etableringsfasen ligger fokus på att utforma och realisera de krav som har stor betydelse för systemets arkitektur. Efter denna fas ska arkitekturen vara införd och testad vilket i sin tur innebär att de största riskerna är eliminerade. Merparten av kraven ska beskrivas i detalj och resten av projektet ska planeras detaljerat. Utifrån användningsfallen tas informationsstruktur och grafisk utformning fram. Detta ligger till grund för framtagning av prototyper som används för att utvärdera och testa användargränssnitt.

Konstruktion (Construction)
Under konstruktionsfasen byggs systemet utifrån den tidigare framtagna arkitekturen samt den specificerade användarfunktionaliteten. Konstruktionsfasen delas ofta upp i flera iterationer för att få bra kontroll och styrning under utvecklingsarbetet. Systemets funktionalitet testas

löpande i de olika iterationerna. I slutet av konstruktionsfasen dokumenteras systemet och utbildningen planeras.

Överlämning (Transition)
Under den avslutande överlämningsfasen genomförs ett antal aktiviteter för att föra in systemet i verksamheten. Under denna fas genomförs bland annat installation i driftsmiljön, utbildning och acceptanstest.

1.6.2 Examensarbetets systemutvecklingsmodell

Eftersom RUP är anpassad för hantera mycket stora projekt krävs det en modifiering av modellen för att passa det mindre omfång som examensarbetet innebär. Projektet är tänkt att följa nedanstående faser.

Teoriinhämtning
Första steget är att inhämta nödvändig information allmänt om ärendehantering och ärendehanteringssystem för att få en bättre uppfattning om uppgiften som ska lösas. Även teknisk information om aktuell plattform samt utvecklingsverktyg inhämtas.

Förstudie
Förstudiens syfte är att ge en överblick över vilka ärendehanteringsmotorer och ärendehanteringssystem som finns på marknaden samt att få viss insikt i hur dessa är uppbyggda och fungerar.

Framtagning av användningsfall och kravspecifikation
Nästa steg är att ta fram en kravbild för produkten. Användningsfall samt en kompletterande kravspecifikation ska tas fram. Dessa ska sedan godkännas av samtliga kravställare.

Utvärdering av olika metoder och lösningar
Möjliga metoder och lösningar ska jämföras och utvärderas. Som exempel kan nämnas att beslut måste tas om systemet ska byggas på en befintlig produkt eller om en helt egenutvecklad produkt ska tas fram.

Design av systemet
Hur designen går till beror lite på vilket lösningsalternativ som valdes i föregående steg. Tanken är att en databasmodell först ska designas. Databasmodellen är sedan grunden i ärendehanteringsmotorn och utifrån denna modell designas ärendehanteringsmotorn. Vid designen av ärendehanteringsmotorn tas först en arkitektur fram. Arkitekturen är en övergripande beskrivning av systemets uppbyggnad. Systemets olika komponenter och deras beroende av varandra identifieras och lagerstrukturen bestäms. Därefter tas en detaljerad design fram som innehåller systemets alla klasser samt som beskriver vilken funktionalitet varje klass innehåller.

Implementering av systemet
Implementering av ärendehanteringsmotorn. Även testning utförs, dels varje komponent för sig under själva utvecklingen och dels hela systemet i slutfasen av utvecklingen.

Utvärdering
Till sist utvärderas hela arbetet, hur det utfördes och vad som kunde ha gjorts bättre. Förslag på utveckling och förbättringar av produkten i framtiden ska också redovisas.

1.7 Begrepp

I detta avsnitt behandlas begrepp som är centrala för den tekniska förståelsen av examensarbetet. Vid framtagandet av kravbilden för projektet gjordes en begreppsmodell som behandlar begrepp relaterade till ärendehantering, se avsnitt 3.1.

1.7.1 Assembly

Ett assembly är ett byggblock i en Microsoft .NET-framework-applikation. Ett exempel på hur ett assembly kan realiseras är som en dll-fil. Ett assembly är en samling funktionaliteter som är kompilerad och realiserad som en enskild enhet. Alla resurser är märkta med åtkomst antingen endast inom assemblyt eller också från kod utanför assemblyt. Indelning i olika assemblies kan användas när en applikation delas upp i flera komponenter. För mer information om detta, se avsnitt 5.3. Microsoft .NET Framework FAQ (2004).

1.7.2 IIS (Internet Information Services)

Webbserver från Microsoft. För mer info, se webbserver avsnitt 1.7.8.

1.7.3 Microsoft .NET Framework

Microsoft .NET framework är ett ramverk för att utveckla och köra olika typer av applikationer och tjänster som webb- och windowsapplikationer. Ramverket består av tre huvuddelar: En runtimemiljö kallad Common Language Runtime (CLR), ett klassbibliotek och ASP.NET som används vid webbutveckling. Microsoft .NET Framework FAQ (2004)

De vanligaste programspråken som stöds är:

- C# (.NET specifikt språk)
- Microsoft Visual Basic .NET
- C++ .NET

De klassbibliotek som finns kan användas av alla .NET-kompatibla programmeringsspråk. Tanken är att utvecklare ska kunna skriva sina program i valfritt programspråk och fortfarande kunna utnyttja .NETs klassbibliotek.

Vid kompilering av .NET kod, oavsett språk, kompileras koden till MSIL (Microsoft Intermediate Language) kod. MSIL är ett mellanläge mellan källkod och exekverbar kod som gör att koden från de olika språken kan kommunicera med varandra. När MSIL koden ska exekveras laddas den in i CLR. Där kompileras koden till exekverbar kod under programmets körning.

Idén är densamma som i Java, där MSIL motsvarar Javas bytekod och CLR motsvarar Javas Virtual Machine. Andra koncept som är snarlika i Java är minnesallokering och skräpinsamling (garbage collection). CLR tar hand om skräpinsamlingen och utvecklare behöver sällan bekymra sig över minneshanteringen. Javaworld (2004).

1.7.4 Microsoft SharePoint

SharePoint är en portalserver som kopplar samman personer och grupper mellan olika affärsprocesser. SharePoint integrerar information från olika system genom möjligheter till enkel inloggning och genom integration mellan olika program, främst Microsofts Office produkter. Det finns ett antal distributions- och hanteringsverktyg som underlättar uppbyggnad och anpassning av portalen. Användarna kan också göra personliga anpassningar av portalens innehåll och utseende. För mer information om SharePoint, se Microsoft SharePoint (2004).

1.7.5 SQL (Structured Query Language)

SQL är standardiserat av ANSI (American National Standards Institute) och är ett språk för dataåtkomst och manipulation av relationsdatabaser som används av majoriteten av alla kommersiella databasprodukter.

1.7.6 Tjock klient

En tjock klient sköter funktionaliteten själv och i de fall som den har kontakt med en server är det enbart för att spara eller hämta data.

1.7.7 Tunn klient

Begreppet tunn klient kan ha lite olika betydelse beroende på i vilket sammanhang det förekommer. I denna rapport har det betydelsen att det är en passiv klient som inte innehåller någon intelligens, utan funktionaliteten finns istället på en server. Klienten är bara ett skal som fungerar som gränssnitt mot användaren.

1.7.8 Webbserver

En webbserver är en värddator med program som kan hantera protokollet HTTP. Webbservern svarar på begäran från webbläsare och levererar den önskade webbsidan. En webbserver kan innehålla flera domäner och webbsidor. Exempel på webbservrar är Microsoft IIS och Apache.

2 Ärendehantering

I detta kapitel beskrivs närmare vad ärendehantering är, syftet med ärendehantering samt olika typer av ärendehantering. Ett exempel presenteras också för att belysa hur ärendehantering kan användas.

2.1 Vad är ärendehantering?

Workflow Management Coalition (2004) spelar en central roll för standardisering av arkitekturer, begrepp och gränssnitt som rör ärendehantering. Organisationen är icke-vinstdrivande och har över 300 medlemsorganisationer i hela världen. De definierar ärendehantering enligt följande:

”The automation of a business process, in hole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.”

Schäl (1996) s. 12 definierar ärendehantering som:

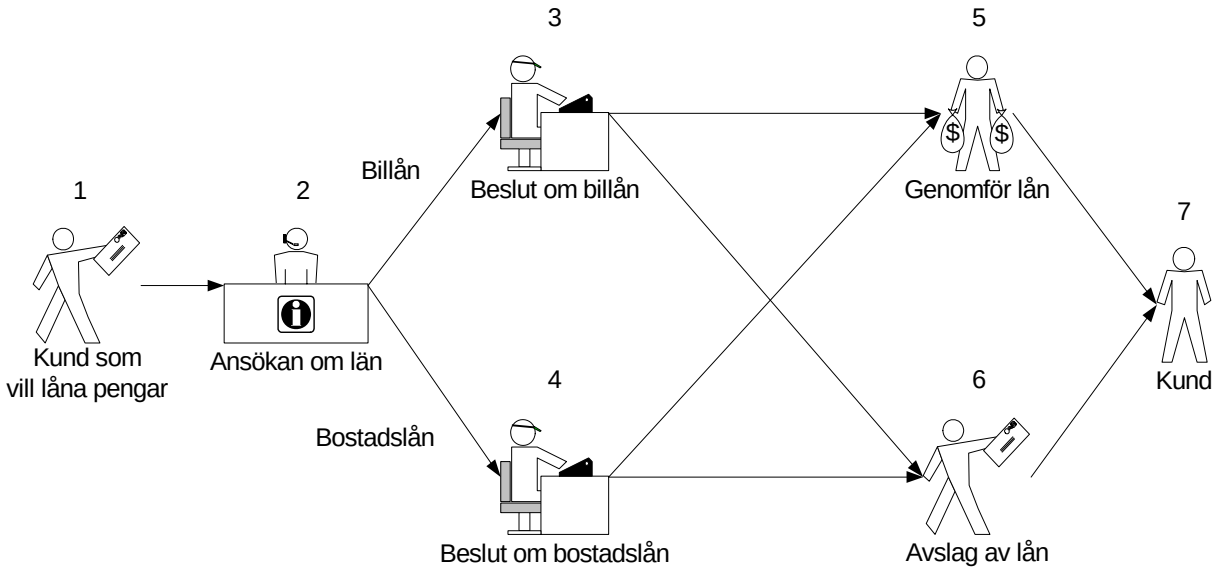
“A workflow is a unit of work generating products or services which are related to, or result in, customer satisfaction. Every workflow has a main customer, who is served by a supplier, or a co-operative network as being a chain of customers and suppliers, working towards the satisfaction of the main customer.”

Vidare skriver Schäl att de flesta arbetsflöden har sekventiella och parallella steg. De involverar förflyttning och spårning av människor, dokument, produkter och information.

2.2 Exempel på ärendehantering

Nedan beskrivs ett exempel på ärendehantering för en låneansökan hos en bank. Låneansökningsprocessen är en mycket förenklad variant av den verkliga processen, för att det ska bli ett enkelt och överskådligt exempel. I avsnitt 6.3 beskrivs hur detta exempel kan realiseras med ärendehanteringsmotorn som detta examensarbete resulterat i.

Låneansökan kan delas upp i följande steg, vilka är illustrerade i Figur 2 nedan:



Figur 2. Process för låneansökan.

1. En kund kommer in med en låneansökan till banken.
2. En tjänsteman tar emot låneansökan på banken kontrollerar om det är en låneansökan för bil eller bostad och skickar den vidare till aktuell beslutsfattare. Det finns en beslutsfattare för respektive typ av lån.
3. Tjänstemannen är ansvarig för billån. Denne tar beslut om kunden ska få lånet eller inte. Om kunden blir beviljad lånet meddelar beslutsfattaren tjänstemannen i steg 5 att pengarna kan betalas ut till kunden. Om kunden inte blir beviljad lånet meddelar tjänstemannen sin kollega i steg 6 som i sin tur underrättar kunden om att denne har fått avslag på sin låneansökan.
4. Tjänstemannen som är ansvarig för bostadslån. Proceduren är densamma som i steg 3.
5. Utbetalningen av lånet till kunden.
6. Kunden meddelas att denne inte är beviljat något lån.
7. Kunden har antingen fått lånet eller avslag. Ärendet avslutas.

Hela arbetet med låneansökan från det att kunden kommer in till banken med en låneansökan till dess att denne får eller blir nekad lånet, kan ses som en process, där syftet med processen är att låna ut pengar till olika kunder. Arbetet med att låna ut pengar till en enskild kund är ett ärende vilket också kan ses som en instans av processen. Representationen av ärendet är i det här fallet först en låneansökan som kunden kommer med till banken, vilket sedan kan resultera i att kunden får låna pengar.

När processen beskrivs anges vilka moment som finns i arbetet och hur arbetet flödar mellan de olika momenten. En sådan beskrivning kallas för ett arbetsflöde. Varje moment i arbetsflödet är ett tillstånd. Ett tillstånd kan

till exempel innehålla uppgifter som att ta ett beslut om ansökan ska beviljas eller inte. En annan uppgift är att betala ut pengarna till kunden efter att denne har blivit beviljad lånet. För att hoppa mellan tillstånden i en process finns vägar (pilarna i figuren ovan). Om det finns flera vägar ut från ett tillstånd måste ett beslut tas om vilken väg som ska väljas. Ett exempel på detta är steg 2, där ett beslut måste tas om det är ett billån ansökan avser eller ett bostadslån som kunden som söks.

Alla personer inblandade i processen är användare. Vissa uppgifter får inte utföras av alla användare och det är därför lämpligt att definiera vilka rättigheter en användare har genom att tilldela dem roller. Ett arbetsflöde kan ange att bara användare som innehar en speciell roll får utföra en bestämd uppgift. I exemplet ovan får bara tjänstemän som innehar rollen att besluta om billån, ta beslut om denna typ av lån. En användare kan inneha flera roller samtidigt. Oftast spelar det ingen roll vilken användare som utför uppgiften utan endast att denne har kompetensen att utföra den.

2.3 Syftet med ärendehantering

Ljungberg (1996) belyser kraften i ärendehanteringssystem genom att peka på områden där de används. Några exempel han tar upp är försäkringsbolagen som använder det för att snabba upp hantering av skadeärenden, banker för att förbättra lånehanteringen och statliga och kommunala förvaltningar för att öka effektiviteten för hantering av ärenden som socialförsäkringar och bygglovshantering.

Listan nedan är hämtat från Ljungberg (1996) s. 13 och sammanfattar några av de vanligaste motiven till att använda ärendehanteringssystem.

- *Ökad produktivitet* genom effektivisering av processer är det mest uppenbara och vanligaste syftet med att använda workflowteknologi. Genom effektiviseringen kan kostnaderna minskas eller produktionskapaciteten ökas.
- *Eliminering av väntetider och fördröjningar*. Många ärenden tillbringar sin mesta tid i olika in- eller utkorgar i väntan på nästa steg i processen. Genom att automatisera flödet så minskas eller elimineras dessa väntetider.
- *Minskade kostnader* genom mindre resursåtgång, både vad gäller mänsklig arbetskraft och förbrukning av papper.
- *Kvalitetsförbättringar* är ett annat mål som man vill uppnå med workflowteknologi. Genom att ge stöd till förbättrad precision, konsekvent agerande och att tidsgränser hålls bidrar workflowteknologi till kvalitetsförbättringar. Fel kan minskas genom standardiserade arbetsprocedurer. Validering och kontrollpunkter är lätta att bygga in i systemen.
- *Ökad kundservice* är ett viktigt kvalitetsmål som på olika sätt kan uppnås med workflowteknologi. All information som rör en viss

kund finns tillgänglig när den behövs, till exempel vid kundkontakter.

- *Ökad kontroll* över processer genom övervakning och uppföljning med workflowteknologi. Genom att statistiska data över processers logistik automatiskt kan genereras, ges möjligheter till analyser för att förbättra prestanda.
- *Bättre processhantering* och möjlighet att förbättra processer. Prestandaproblem görs konkreta och blir lättare att förstå.
- *Ökad arbetstillfredställelse.* Genom reducering av rutinarbete kan tid frigöras till intressantare arbetsuppgifter. Frustration över försvunna dokument minskar.

2.4 Typer av arbetsflöden

Arbetsflöden kan se olika ut beroende på hur processen ser ut. En av de stora utmaningarna i examensarbetet är att finna en lösning som är generell och flexibel nog att fungera på processer oberoende av deras utseende. De krav som ställdes på produkten när det gäller flödet finns behandlade i avsnitt 3.4. En viktig aspekt på processen är huruvida det är ett rutinärende eller om det är ett flexibelt ärende som ska behandlas. Ljungberg (1996) har delat in arbetsflöden i fyra olika typer som redogörs för nedan.

2.4.1 Produktionsflöden

Processer som är välstrukturerade och ofta har en hög grad av repetition kallas produktionsflöden. Eftersom de har få undantag är det relativt lätt att effektivisera denna typ av processer. Ofta tillhör denna typ av process kärnverksamheten i ett företag eller organisation. Det finns ett väldefinierat regelverk för hur olika situationer ska hanteras.

Skadeärenden hos försäkringsbolag och låneärenden hos banker hör typiskt hemma i ett produktionsflöde.

2.4.2 Samarbetsprocesser

Om produktionsflöden var repetitiva och förutsägbara är samarbetsprocesser raka motsatsen. Här är målen definierade men vägen dit är inte given. Det är inte ovanligt att en samarbetsprocess sker i form av ett projekt och kräver mycket kommunikation. Exempel på denna typ av processer är produktutveckling och marknadsföring.

2.4.3 Administrativa flöden

Administrativa flöden är precis som produktionsflöden rutinmässiga, men Ljungberg har valt att införa denna typ av flöden då de inte är kritiska för verksamheten på samma sätt. Ofta är det okomplicerade uppgifter som går bra att effektivisera. Typiska administrativa flöden är attestering av reseräkningar och inköpsordrar.

2.4.4 Ad hoc-flöden

Ad hoc-flöden används i viss litteratur synonymt med samarbetsprocesser. De är liksom samarbetsprocesserna ostrukturerade i sin natur men kan på samma sätt som administrativa flöden betraktas som mindre kritiska för verksamheten.

2.5 Ärendehanteringsmotorns roll

Examensarbetets uppgift är att bygga en *ärendehanteringsmotor*. En ärendehanteringsmotor är inte samma sak som ett *ärendehanteringssystem*.

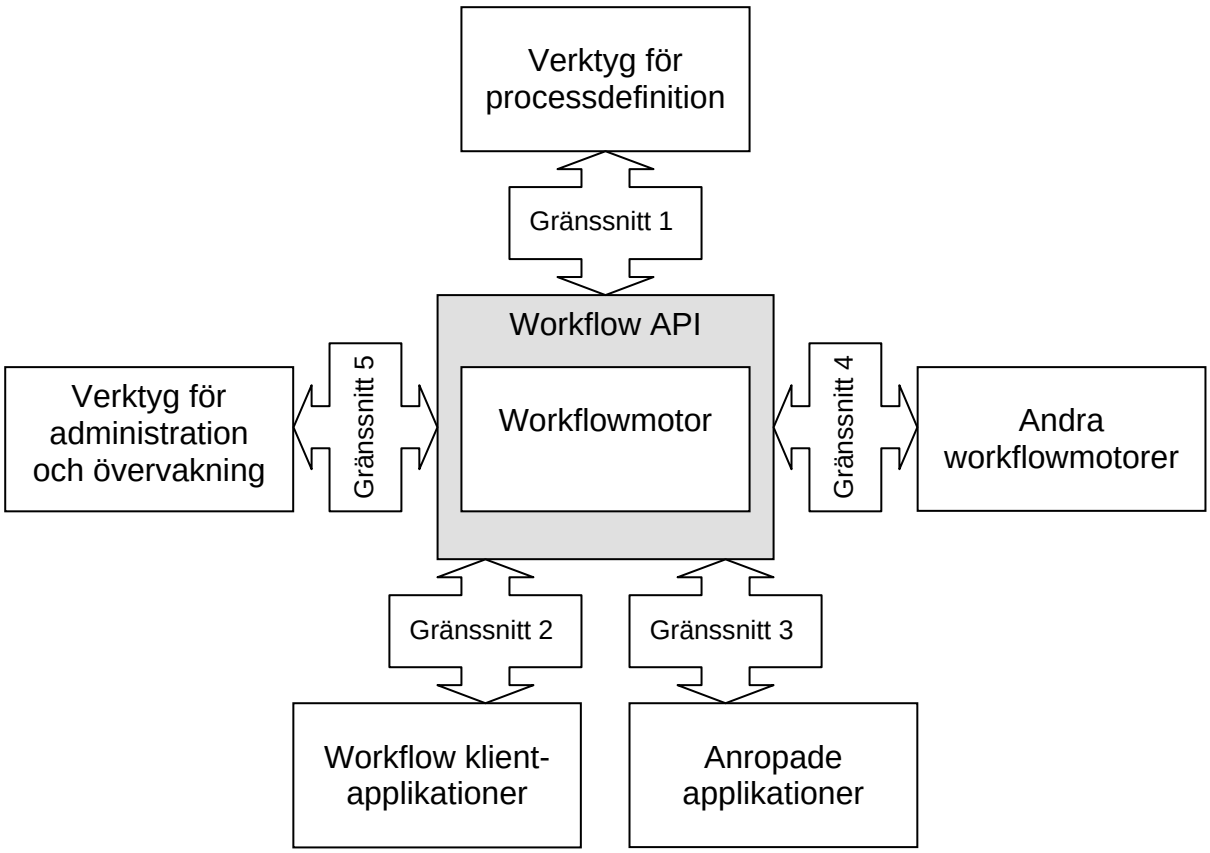
David Hollingsworth på Workflow Management Coalition har skrivit en referensmodell för arbetsflöden. I den definierar han en ärendehanteringsmotor som:

“A software service or "engine" that provides the run time execution environment for a workflow instance.” Hollingsworth (1995) s.22

Detta skiljer sig alltså från ett ärendehanteringssystem som Hollingsworth definierar på följande vis:

“A system that completely defines, manages and executes “workflows” through the execution of software whose order of execution is driven by a computer representation of the workflow logic.” Hollingsworth (1995) s.6

Ärendehanteringsmotorn sköter alltså flödet i en process, och workflowsystemet använder sig av ärendehanteringsmotorn för att åstadkomma detta. Figuren nedan visar referensmodellens syn på ärendehanteringsmotorn.



Figur 3. Workflow Management Coalition referensmodell – Komponenter & gränssnitt. Hollingsworth (1995) s.20

Ärendehanteringsmotorn är placerad centralt och arbetar via fem gränssnitt mot kringliggande komponenter. Kommunikationen med ärendehanteringsmotorn sker via *Workflow API*. Det är en mängd konstruktioner som ger möjlighet att komma åt olika tjänster i ärendehanteringsmotorn. Många av funktionerna i gränssnitten överlappar varandra vilket gör att det kan vara mer korrekt att se det som ett Workflow API istället för fem individuella gränssnitt.

Workflow Management Coalition har definierat gränssnitten hur de ska se ut, och de bygger på XML. I detta kapitel följer en redogörelse för de olika komponenterna som de är beskrivna i referensmodellen, Hollingworth (1995). Eftersom gränssnitten är standardiserade utgör de en viktig mekanism att ta hänsyn till vid utvecklandet av en ärendehanteringsmotor. Läs mer om det i avsnitt 4.4 och 4.5.

Verktyg för processdefinition (Gränssnitt 1)
Genom detta gränssnitt går det att definiera en process. Användning av detta gränssnitt innebär att verktyget för att designa ett flöde blir oberoende av vilken ärendehanteringsmotor som används och vice versa.

Workflow klientapplikationer (Gränssnitt 2)
För användaren är det klientapplikationen som associeras med ett ärendehanteringssystem. Det är klientapplikationen som presenterar uppgifter och ger användaren möjlighet att utföra dem.

Anropade applikationer (Gränssnitt 3)
En typiskt anropad applikation skulle kunna vara en e-posthanterare. Genom att implementera detta gränssnitt vid utvecklandet av en applikation går det att koppla in den direkt mot ärendehanteringsmotorn och således blir den enkel att använda i ett ärendehanteringssystem.

Andra ärendehanteringsmotorer (Gränssnitt 4)
Samarbete mellan olika ärendehanteringsmotorer går genom detta gränssnitt, vilket hanterar utbyte av kontrollinformation och applikationsdata.

Verktyg för administration och övervakning (Gränssnitt 5)
Genom detta gränssnitt går det att hämta en komplett bild av statusen hos arbetsflödet som går genom en organisation. Standardgränssnittet gör att det går att övervaka flera olika ärendehanteringsmotorer med samma verktyg.

3 Kravanalys

Detta kapitel beskriver vad produkten ska ha för funktionalitet.

3.1 Begreppsmodell

Det finns flera anledningar till att ha en begreppsmodell. En anledning är att genom sammanställning av viktiga begrepp erhålls en bättre uppfattning om vad ärendehantering innebär och omfattar. En annan är att kommunikation underlättas och färre missförstånd uppstår.

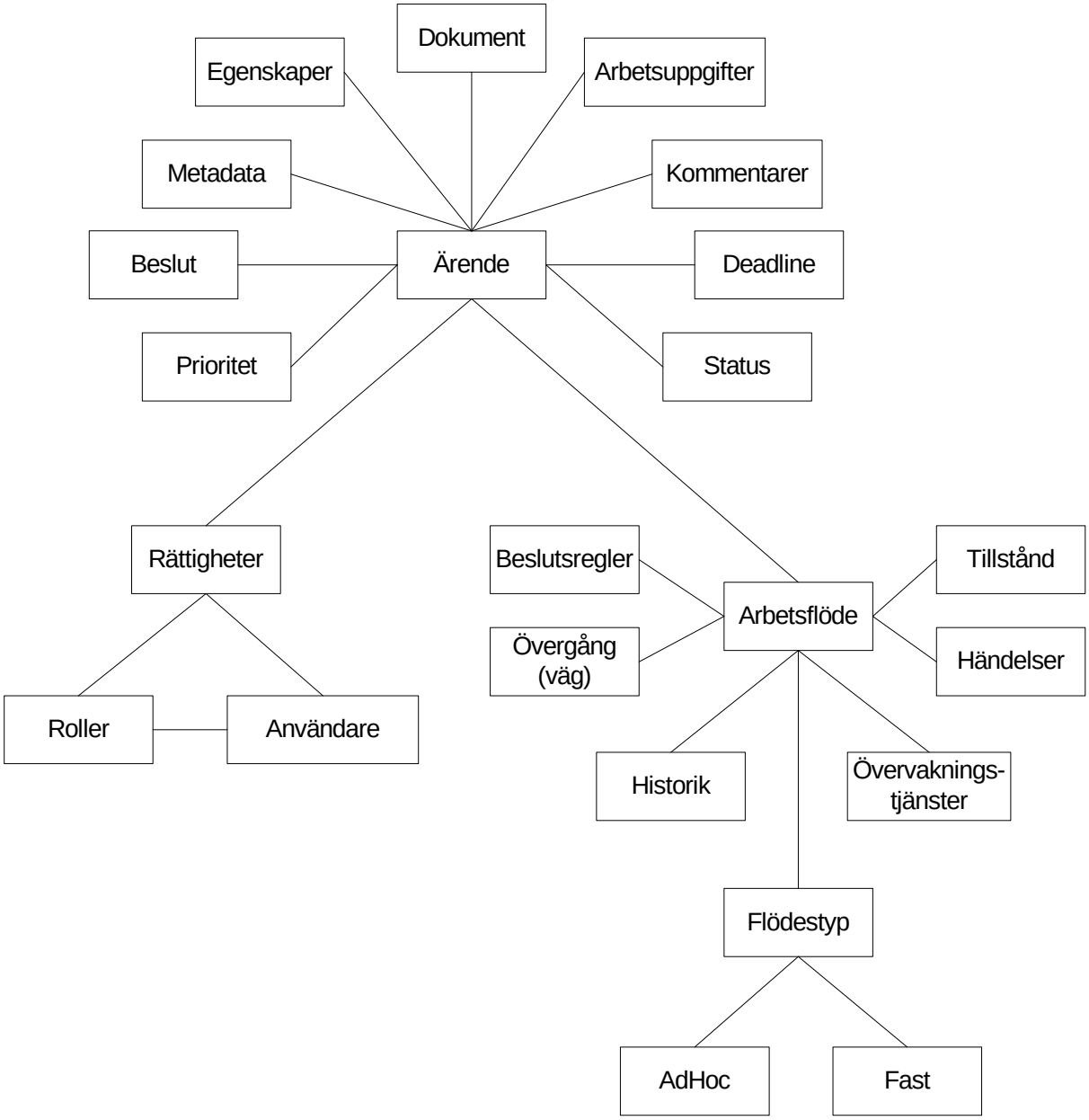
Nedan beskrivs viktiga begrepp i ett ärendehanteringssystem.

Term	Förklaring
Arbetsflöde (Process)	Ett arbetsflöde är en i förväg definierad process tillämpad på ett givet ärende. Arbetsflödet beskriver hur ett ärende kan vandra i organisationen. Det består av tillstånd och övergångar mellan dessa.
Arbetsuppgift	En arbetsuppgift kan tillhöra ett tillstånd och är något som ska utföras manuellt av en människa.
Beslut	Ett beslut är ett vägval som används för att välja till vilket tillstånd som ärendet ska komma till härnäst.
Händelse	En användare utför arbetet och flödet flyttas fram. Detta kan antingen ske manuellt eller automatiskt. En händelse kan utlösas genom att en arbetsuppgift slutförts eller av att ett ärende precis kommit till ett nytt tillstånd. Händelser innebär till exempel att e-post (påminnelser) skickas.
Metadata	Metadata är information som är kopplad till ett ärende som till exempel datum, pris, eller artikelnummer.
Prioritet	Ett ärende kan ha olika prioritetsnivåer som anger vilken prioritet ärendet har.
Roll	En roll är en grupp användare som utför en viss typ av arbetsuppgifter i en organisation. Typiska roller kan vara ”assistent”, ”föredragare”, ”handläggare”, ”granskare” och ”chef”. Rollerna är centrala för möjligheten att skapa

	återanvändbara processdefinitioner.
Status	Statusen anger om ärendet är inaktivt, aktivt eller arkiverat.
Tillstånd	Ett tillstånd är ett delsteg i ett arbetsflöde, det vill säga vad som ska utföras när en process nått ett visst steg. Tillstånden binds samman av de möjliga övergångarna dem emellan. Varje tillstånd har som syfte att föra arbetet med processen lite närmare syftet för hela processen. Ett tillstånd kan innehålla arbetsuppgifter, händelser samt beslut.
Workflow	Workflow är en synonym för arbetsflöde. Dessa två termer används ofta parallellt inom IT-stöd för processtyrning.
Ärendehanteringsmotor	En ärendehanteringsmotor är en komponent i ett ärendehanteringssystem som gör det möjligt att hantera digitalt lagrade ärenden i ett arbetsflöde.
Ärende	All funktionalitet utgår från ärendet och syftar till att underlätta ärendehanteringen. Med ett ärende kan avses samlingen information rörande ett gemensamt syfte.
Ärendehantering	Ärendehantering kan definieras som strukturerad hantering av information i syfte att få beslutade saker utförda alternativt att skapa underlag för beslut.
Övergång	En övergång är kopplingen mellan två tillstånd i ett arbetsflöde. Övergångarna i ett flöde definierar de vägar längs vilka en process kan utvecklas.
Övervakningstjänster	Övervaknings- och larmtjänster kan till exempel övervaka och larma om meddelande saknas, regler inte följs eller tidsgränser överskrids.

Tabell 1. Viktiga begrepp i ett ärendehanteringssystem.

Nedan visas ett diagram på begreppsmodellen. Syftet är att illustrera hur centrala begrepp är relaterade till varandra.



Figur 4. Begreppsmodell.

3.2 Kravbild

Systemet avser att tillhandahålla en generell ärendehanteringsmotor som ska fungera som en komponent till ett komplett ärendehanteringssystem, anpassad för Ibitecs framtida lösningar med behov av ärendehantering.

Följande krav för mjukvara och utvecklingsmiljö finns:

- **Utvecklingsmiljö:** Microsoft Visual Studio .NET 2003
- **Runtimemiljö:** Microsoft .NET Framework 1.1
- **Databasserver:** Microsoft SQL Server 2000

Här ges en sammanfattning av ärendehanteringsmotorns viktigaste funktioner:

- **Skapa ärendetyper**
  Olika typer av ärendetyper ska kunna definieras samt vilka metadata som tillhör ärendetypen.
- **Skapa arbetsflöden**
  Det ska gå att definiera arbetsflöden ett ärende av en viss typ ska vandra i systemet. Detta utförs lämpligen av en konsult innan systemet sätts i bruk hos kunden. Flera olika arbetsflöden ska kunna köras parallellt i systemet. Ett arbetsflöde består av ett starttillstånd och ett sluttillstånd. Emellan dessa ska flödet kunna definieras med vilka tillstånd som ska finnas, hur dessa är kopplade till varandra samt allt som ett tillstånd kan innehålla. För mer information om detta se avsnitt 3.4.
- **Hantera användare och roller**
  Rättigheter för hela systemet samt för enskilda ärenden ska kunna hanteras.
- **Registrera/bearbeta/söka/avsluta ärenden**
  Ärenden ska kunna skapas och bearbetas. När ett ärende har tagits igenom hela arbetsflödet (processen är slutförd) ska ärendet avslutas och arkiveras. Det ska dessutom vara möjligt att söka efter både aktuella och arkiverade ärenden.
- **Loggning av data**
  Alla förändringar av ärenden måste loggas. Möjlighet måste finnas att ta fram historik som visar vad som hänt med ett ärende.
  Exempel på vad som ska loggas:
  - För samtliga ändringar av data som tillhör ett ärende loggas vilken användare som utfört ändringen.
  - Tid och datum när ärenden skapas.
  - All metadata som tillhör ärenden och hur dessa förändras.
  - Kommentarer till ärenden.
  - Vem som utför arbetsuppgifter och beslut.
  - Hur ett ärende har vandrat i arbetsflödet.

3.3 Användningsfall

I de flesta projekt på Ibitec arbetas det efter modellen att först ta fram kravbilden och därefter ta fram alla relevanta användningsfall. Dessa kompletteras sedan med en kravspecifikation med de krav som inte användningsfallen täcker. Examensarbetet har bedrivits enligt denna metod.

Användningsfallsmodellen är en modell av systemets alla funktioner och dess omgivning och fungerar som ett kontrakt mellan kravställaren och utvecklarna. Modellen består av systemets alla användningsfall.

Följande inkluderas i den användningsfallsmodell som Ibitec använder:

- Aktörer och deras beskrivningar.
- Användningsfallsdiagram som visar relationer.
- En detaljerad användningsfallsbeskrivning med namn och en beskrivning av händelseförloppet.
- Process- och aktivitetsdiagram.

Användningsfallen kan sedan användas för att verifiera databasmodellen som tas fram i designfasen och i slutet testas slutprodukten mot användningsfallen för att kontrollera att dessa uppfylls.

Här beskrivs endast de mest relevanta användningsfallen för systemet:

- **Skapa ärende**
  Skapa ett nytt ärende av en viss ärendetyp.
- **Skapa arbetsflöden**
  Skapa arbetsflöden som bestämmer hur ärenden av en viss ärendetyp ska vandra i systemet.
- **Sätta rättigheter till ett ärende**
  Som standard ärvs de rättigheter som finns definierade för ärendetypen när ett nytt ärende skapas. Dessa ska kunna uppdateras. Rättigheter till ett ärende kan kopplas till en eller flera roller och/eller till enskilda användare.
- **Uppdatera ärende**
  Ändra i ett ärende, till exempel dess metadata.
- **Söka efter ärenden**
  Leta upp ärenden exempelvis för en viss användare.
- **Hämta information om ett ärende**
  Hämta information om ett ärende, till exempel vilka arbetsuppgifter som finns i olika tillstånd.
- **Skapa/uppdatera arbetsuppgift**
  Det ska gå att skapa arbetsuppgifter/beslut som måste vara slutförda innan ärendet lämnar tillståndet. Flera arbetsuppgifter ska kunna finnas för ett och samma tillstånd men endast ett beslut.
- **Tilldela arbetsuppgift/beslut**
  Varje arbetsuppgift/beslut måste någon eller några tilldelas, antingen en eller flera användare och/eller en eller flera roller. Även uppdatering av aktuella tilldelningar ska kunna ske.

- **Utföra arbetsuppgift/manuellt beslut**
  Arbetsuppgifter och beslut ska kunna utföras.

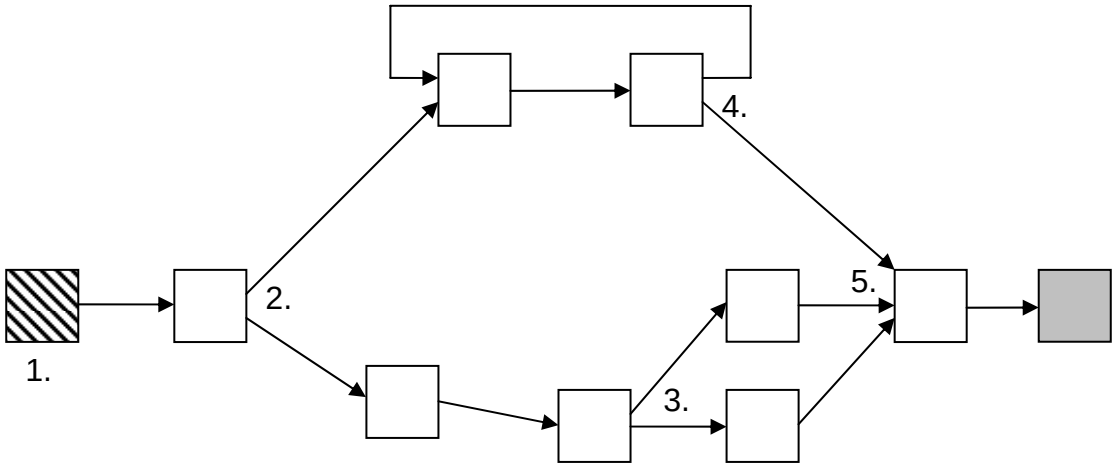
3.4 Krav på arbetsflödet

I detta avsnitt beskrivs de krav som finns dels på det fasta flödet och dels på ad hoc-flödet.

3.4.1 Fast flöde

För varje ärendetyp krävs ett flöde som definierar hur en instans av ärendetypen ska vandra i organisationen. Flera olika arbetsflöden ska kunna definieras och köras parallellt. Konfigurationen sker innan systemet sätts i bruk. Om konfiguration av flödet behöver göras i efterhand när det redan finns aktiva ärenden i flödet, kommer inget stöd finnas för att hantera det nya och gamla flödet parallellt. Enbart ett arbetsflöde per ärendetyp kommer att stödjas.

Nedan beskrivs ett exempel på ett arbetsflöde för att illustrera vad systemet ska klara av.



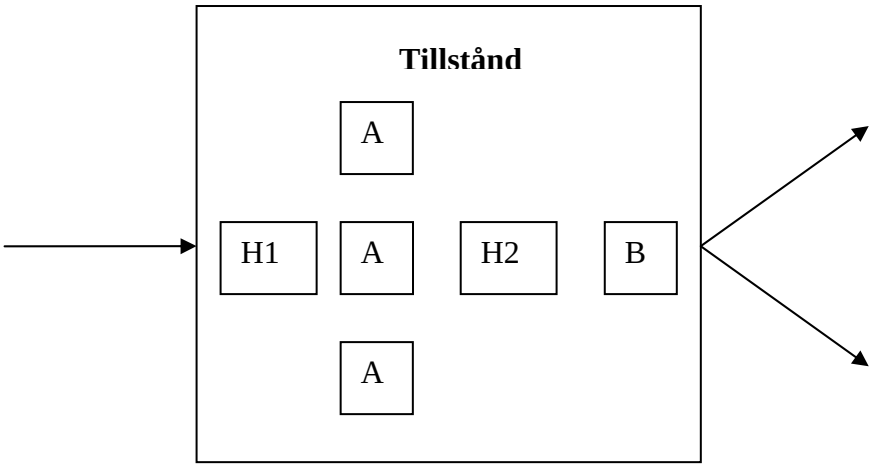
Figur 5. Ett exempel på ett fast arbetsflöde.

1. Starttillståndet. Här börjar flödet. I varje tillstånd kan det finnas noll, en eller flera arbetsuppgifter. När alla arbetsuppgifter är slutförda i ett tillstånd går det över till nästa. Det går endast att utföra arbetsuppgifter i aktiva tillstånd.
2. Vid punkt 2 delar sig arbetsflödet. Från att ha haft ett aktivt tillstånd blir det två aktiva. Detta möjliggör parallellt arbetsflöde.
3. Vid punkt 3 fattas ett manuellt beslut. Det innebär att en användare beslutar om vilket väg som ska väljas.
4. Vid punkt 4 fattas ett automatiskt beslut. På samma sätt som vid punkt 3 ska det väljas vilken väg som flödet ska ta, men denna gång sker beslutet automatisk med hjälp av en beslutsregel.

5. Vid punkt 5 sammansmälter två aktiva tillstånd till ett. Innan tillståndet blir aktiverat måste båda de vandrande ”aktiveringarna” ha kommit fram.

Ett tillstånd ska kunna innehålla följande:

1. En eller flera arbetsuppgifter. Alla arbetsuppgifter i ett tillstånd måste vara genomförda för att ärendet ska gå vidare i flödet.
2. Händelser som till exempel att skicka e-post. Händelser ska kunna köras både direkt när tillståndet blir aktivt och precis innan tillståndet lämnas.
3. Ett tillstånd måste alltid innehålla information om vilket som är nästa tillstånd i flödet. Det kan vara ett eller flera nästa tillstånd. Om det är flera måste en beslutsregel finnas som avgör vilket tillstånd som ska väljas (se punkt 4).
4. Om flödet från ett tillstånd kan resultera i flera olika vägar ska antingen ett automatiskt eller ett manuellt beslut finnas som avgör vilken väg som ska väljas.



Figur 6. Ett tillstånd och dess innehåll.

Symbolförklaring:

H1: Händelse som körs när tillståndet blir aktivt
A: Arbetsuppgift
H2: Händelse som körs innan tillståndet lämnas
B: Automatiskt eller manuellt beslut om vilken väg som ska väljas

Flödet definieras antingen i en XML-fil eller direkt i databastabellerna. För att detta ska vara möjligt måste det gå att

1. Ange vilken ärendetyp som ska tillämpas.
2. Ange ärendetypens olika tillstånd.
3. Ange när och vilken händelse som ska köras.

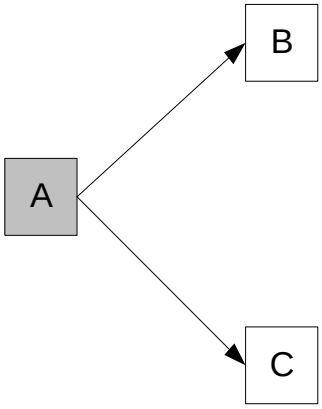
4. Ange när och vilken beslutsregel som ska tillämpas, automatisk eller manuell.
5. Ange vilka roller som har:
    a. Fulla rättigheter till alla ärenden av en viss ärendetyp.
    b. Läsrättigheter till alla ärenden av en viss ärendetyp.
    c. Rättighet att utföra en arbetsuppgift i ett visst tillstånd (här kan även enskilda användare tilldelas rättigheter).
    d. Rättighet att utföra ett manuellt beslut (här kan även enskilda användare tilldelas rättigheter).

3.4.2 Ad hoc-flöde

Kravet för ad hoc-flödet skiljer sig en del från det fasta flödet. I ad hoc-flödet krävs endast att varje tillstånd kan tillhöra en användare. Tillståndet innehåller ingenting förutom just en användare. Tillståndet kan enbart ha statusen att det är aktivt, användaren har ärendet, eller inaktivt, användare har inte ärendet.

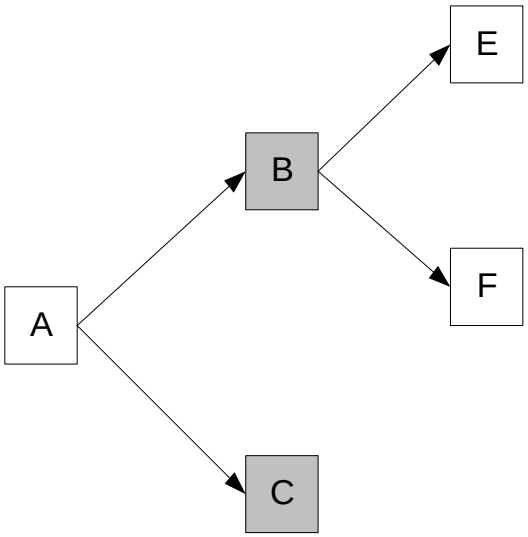
Ett ad hoc-flöde är ett icke förutbestämt flöde som byggs upp när ärendet skickas mellan olika användare. Krav finns att flera användare ska kunna förfoga över ärendet samtidigt vilket betyder att flera tillstånd måste kunna vara aktiva samtidigt. För att ett ärende ska komma framåt i flödet måste en användare skicka det vidare. Användaren ska själv kunna välja till vilken eller vilka användare som ärendet ska skickas till. Eftersom ärendehanteringsmotorn inte kan veta när ärendet är slutfört måste någon användare avsluta det manuellt. Ett ärende måste också kunna skickas tillbaka i flödet, då enbart till någon eller några av de användare som ärendet kom ifrån. Dessa användare ska i sin tur kunna skicka tillbaka ärendet till de användare som de fick ärendet av och så vidare. Detta betyder att flödet måste kunna backas ända till starttillståndet. Krav finns också att samtliga användare som har ett specifikt ärende ska kunna listas. Vidare ska alla ärenden som en användare har också kunna listas.

Nedan illustreras ett exempel som visar hur ett ad hoc-flöde kan se ut. Alla tillstånd i figurerna nedan är representerade som användare.



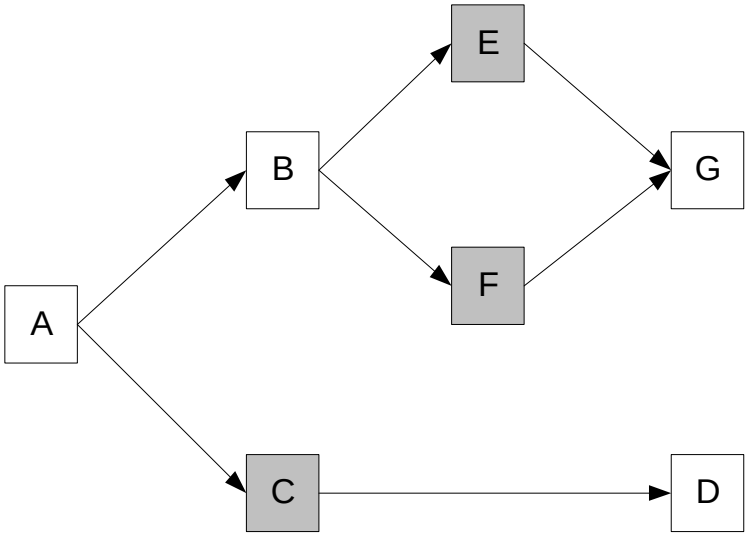
Figur 7. Exempel på ett ad hoc-flöde - steg 1.

Ärendet befinner sig hos användare A har ärendet som skickar det till användarna B och C.



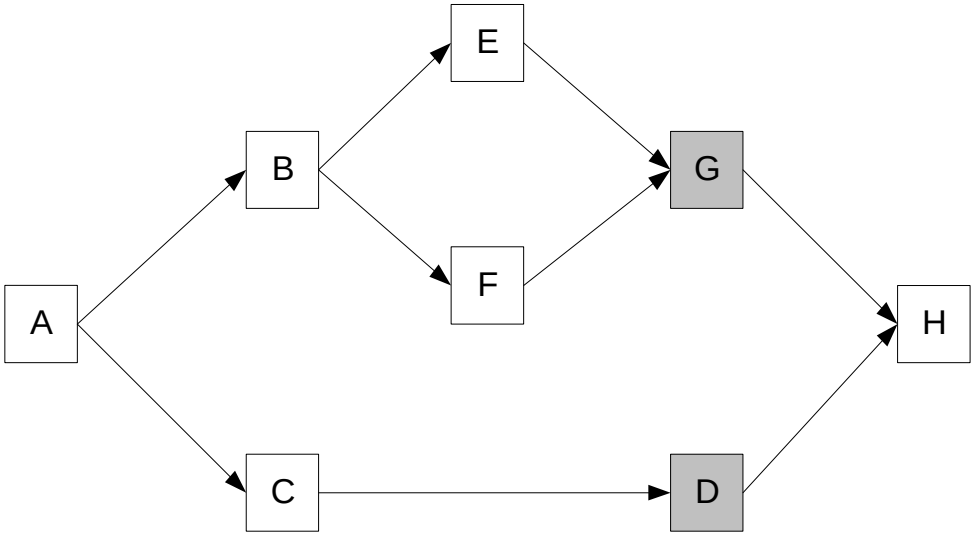
Figur 8. Exempel på ett ad hoc-flöde - steg 2.

Efter att användare A har skickat ärendet kommer A inte att ha kvar det längre. Nu har istället B och C ärendet. B väljer att skicka ärendet vidare till E och F.



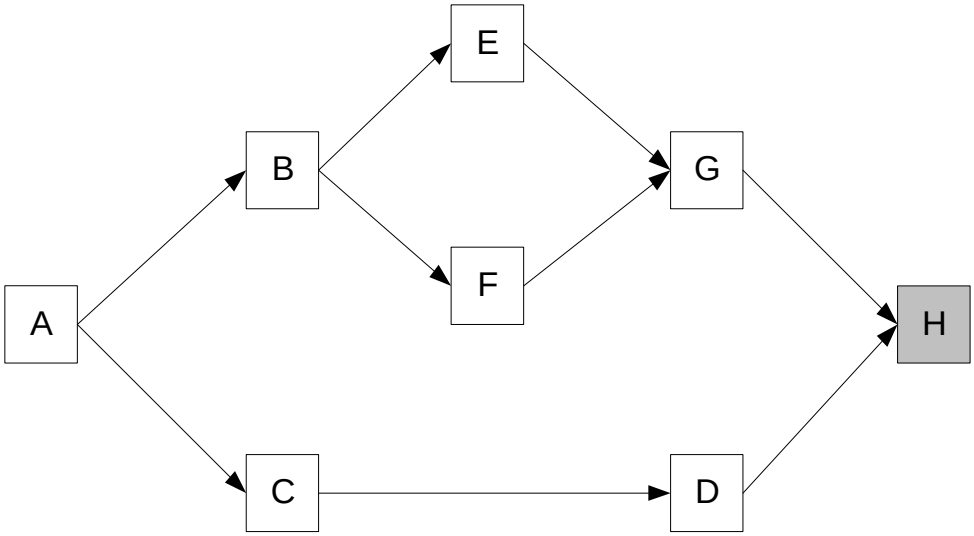
Figur 9. Exempel på ett ad hoc-flöde - steg 3.

Nu befinner sig ärendet hos C, E och F. Både E och F skickar ärendet till G. C skickar det till D.



Figur 10. Exempel på ett ad hoc-flöde - steg 4.

Ärendet befinner sig hos G och D. Båda skickar det till H.



Figur 11. Exempel på ett ad hoc-flöde - steg 5.

Ärendet befinner sig hos H som väljer att avsluta det. Ärendet kommer då att arkiveras.

4 Utvärdering av möjliga lösningar

I det här kapitlet behandlas några av de möjliga lösningar som finns för att uppfylla de krav som beskrivs i kapitel 3.

4.1 Standardprodukt

Ett sätt att uppfylla kraven för systemet är att använda sig av en befintlig standardprodukt och sedan anpassa den så att den uppfyller kraven.

Fördelar med en standardprodukt är:

- Sparar tid eftersom ärendehanteringsmotorn inte behöver utvecklas från grunden.
- Det kan vara billigare att köpa en produkt än att utveckla en egen.
- Extra funktionalitet kommer på köpet som ger mervärde åt lösningen och kan komma till användning i framtiden.
- Om uppgraderingar av produkten tillhandahålls erhålls dels ny funktionalitet och dels åtgärdas fel utan extra insatser.

Nackdelar med en standardprodukt är:

- Anpassningar av produkten kanske måste göras för att den ska passa in i den befintliga miljön vilket kan vara tidskrävande.
- Produkten kanske inte går att anpassa så mycket som skulle behövas och den kan då upplevas som mindre flexibel.
- Om inte behov finns för all funktionalitet i produkten blir den onödigt dyr.
- Ingen kontroll över hur produkten kommer att utvecklas i framtiden och vilken support som kommer att tillhandahållas.
- Det kan vara dyrt med licenser om det är många som kommer att använda systemet.

Ett antal standardprodukter har undersökts och utvärderats. Eftersom krav finns att ärendehanteringsmotorn ska köras på plattformen Microsoft .NET är utbudet kraftigt begränsat. Om någon annan plattform hade varit aktuell som till exempel Java hade utbudet varit ett helt annat. De enda alternativen som kunde påträffas var Microsoft BizTalk, Microsoft BizTalk (2004), och Skelta, Skelta Workflow.NET (2004).

Microsoft BizTalk var egentligen aldrig något alternativ eftersom den innehåller mycket extra funktionalitet, vilket produkten som ska tas fram inte har något behov av. Tanken är istället att den framtagna ärendehanteringsmotorn ska kunna kompletteras med Microsoft BizTalk om behovet av en mer avancerad ärendehanteringsmotor skulle uppstå.

Skelta är en ärendehanteringsmotor som är utvecklad av ett indiskt företag. Skelta har följande egenskaper:

- Utvecklat och baserat på plattformen Microsoft .NET.
- Enkelt att skapa arbetsflöden. Innehåller dessutom ett grafiskt verktyg för att skapa arbetsflöden.
- Funktionalitet för att hämta, analysera och spåra ärenden.
- Kan kopplas mot de vanligaste databasservrarna.
- Stöd för automatiserade flöden.

4.2 Egenutvecklad produkt

Ett annat alternativ till en standardprodukt är att utveckla en egen som är anpassad för de behov och krav som finns på systemet.

Fördelar med en egenutvecklad produkt är:

- Till att börja med kan en egen enkel och billig prototyp utvecklas som sedan kan utökas när behov uppstår. Produkten är då från början anpassad för de behov och krav som finns på systemet.
- Full kontroll erhålls på vilka funktioner som ska ingå i produkten och ingen onödig funktionalitet behöver implementeras.

Nackdelarna med en egenutvecklad produkt är:

- Det tar tid och kostar pengar att utveckla en egen ärendehanteringsmotor.
- Om behov av ny funktionalitet uppstår måste detta utvecklas på egen hand, vilket tar tid och kostar pengar.

4.3 Standardprodukt eller egenutvecklad produkt?

Om en standardprodukt ska användas är Skelta enda alternativet som uppfyller plattformskraven. Skelta innehåller en hel del funktionalitet som produkten inte är i behov av och det bedömdes att det skulle behövas åtskilligt med tid för att anpassa Skelta. Det är också osäkert med support och hur stabil Skelta är eftersom produkten är helt nyutvecklad. Dessutom kostar Skeltas licenser pengar och därför valdes istället alternativet med en egenutvecklad produkt. Det är då möjligt att enbart implementera den funktionalitet som systemet är i behov av och bestämma hur gränssnittet mot övriga komponenter ska se ut. Fördelen med en egenutvecklad produkt för Ibitecs del är dessutom att de inte behöver betala några licenspengar när ärendehanteringsmotorn i framtiden ska användas i olika kundprojekt.

4.4 Format för konfiguration av arbetsflöde

Alternativen är att antingen använda sig av ett standardiserat format för att beskriva arbetsflödet eller utveckla ett eget format. Nedan beskrivs de två olika varianterna.

4.4.1 Standardiserat format

Branschorganisationen Workflow Management Coalition (2004) som nämns i avsnitt 2.1 har utvecklat ett standardiserat format för hur ett arbetsflöde kan beskrivas. Med deras format definieras arbetsflöden med hjälp av XML, Workflow Process Definition Interface (2004) - XML Process Definition Language. En sammanfattning av gränssnittens funktionalitet finns i avsnitt 2.5.

Fördelarna med att stödja formatet är att ärendehanteringsmotorn blir kompatibel med ett standardiserat format. Det blir lättare att använda ärendehanteringsmotorn med andra komponenter som också stödjer samma standardiserade format. Det blir dessutom enklare att byta till en annan standardiserad ärendehanteringsmotor om behov skulle uppstå. En ytterligare fördel är att det går att använda andra produkter för att definiera, visa och ändra arbetsflödet. Det innebär att inget eget verktyg för detta ändamål behöver utvecklas.

Nackdelen med att stödja formatet är att det tar åtskilligt med tid att utveckla. Formatet är dessutom relativt omfattande med mycket funktionalitet som systemet inte är i behov utav.

4.4.2 Egenutvecklat format

Alternativet till att använda ett standardiserat format är att utveckla ett eget, anpassat för just de krav som finns på arbetsflödet. Det egenutvecklade formatet kan byggas precis som standardformatet med XML. Ett annat sätt är att skapa arbetsflöden direkt i databastabellerna.

Fördelarna med ett egenutvecklat format är att det går snabbt att utveckla eftersom endast den funktionalitet som behövs implementeras. Nackdelen är att formatet inte blir kompatibelt med andra produkter som grafiska gränssnitt eller ärendehanteringsmotorer.

4.4.3 Val av format

Standardformatet som finns är komplext och innehåller mycket funktionalitet som produkten inte är i behov av. Det bedömdes vara för tidskrävande att skaffa sig den kompetens som behövs för att utnyttja hela eller delar av formatet. Därför valdes att utveckla ett eget format för att definiera arbetsflöden. Valet stod mellan att använda XML eller att skapa flödet direkt i databasen. Det bedömdes vara omständligt att skapa flödet direkt i databasen, eftersom den som konfigurerar arbetsflöden då måste ha kontroll över hur id:n från olika tabeller hänger ihop. Modellen med att skapa flödet i XML-filer bedömdes vara en bättre lösning. Lösningen blir mer överskådlig vid konfigurering av olika arbetsflöden, metadata, arbetsuppgifter, regler för beslut, händelser med mera.

Ytterligare ett val som måste genomföras är om ett grafiskt användargränssnitt för att skapa, visa och ändra arbetsflöden ska utvecklas. Fördelarna med ett grafiskt gränssnitt är att det blir enklare och mer överskådligt att skapa, visa och ändra i arbetsflöden. Det finns nackdelar

med att utveckla ett eget grafiskt gränssnitt. En är att det tar tid. En annan är att om formatet för ärendehanteringsmotorns konfiguration ändras, måste även det grafiska gränssnittet byggas om. Det fanns inte något krav på ett grafiskt gränssnitt och därför togs beslutet att inte utveckla en egen applikation för ändamålet.

4.5 Ärendehanteringsmotorns gränssnitt

Branschorganisationen Workflow Management Coalition har även tagit fram ett gränssnitt för hur ärendehanteringsmotorn ska kommunicera med övriga komponenter i ärendehanteringssystemet, för mer information se dokumentet Workflow Management Application Programming Interface Specification (2004). Det är gränssnitt 2 och 3 som visas i Figur 3 avsnitt 2.5, som specificerar gränssnittet mot klientapplikationer. Fördelen med att stödja formatet är att användning av standardkomponenter blir då möjlig tillsammans med ärendehanteringsmotorn. Ärendehanteringsmotorn kan också enklare bytas ut mot en annan standardmotor, vilket inte är fallet om inte standardformatet stöds.

Gränssnittet som Workflow Management Coalition har tagit fram är för omfattande för att det ska vara användbart till det system som ska utvecklas. Istället utvecklas ett eget gränssnitt. Gränssnittet ska bestå av metoder som tillhandahåller all den funktionalitet som ärendehanteringsmotorn erbjuder övriga komponenter i ett ärendehanteringssystem. Ett krav är också att gränssnittet enkelt ska gå att tillgängligt via Web services vilket blir enklare att stödja om ett egenutvecklat gränssnitt tas fram.

5 Systemutveckling

I det här kapitlet beskrivs arkitekturen och designen av ärendehanteringsmotorn samt lite kort om arbetet med att implementera den.

5.1 Arkitekturgruppens roll

Efter att användningsfallen tagits fram arbetar Ibitec utifrån att de först tar fram en SAD (Software Architecture Design) och därefter en detaljerad design. Ibitec har en så kallad arkitekturgrupp där flera erfarna programmerare ingår. Dess uppgift kan se olika ut i olika projekt. I detta projekt fick en person ur gruppen ansvaret att ta fram en SAD utifrån den kravbild och de användningsfall som tagits fram. SAD-ens innehåll ska visa på generella arkitekturprinciper och specificera de grova dragen i arkitekturen.

Den detaljerade designen var examensarbetarnas uppgift att ta fram. Här fungerade arkitekturgruppen som ett bollplank för att diskutera olika lösningar, samtidigt som granskning av designen gjordes.

Det finns flera anledningar till att en central grupp tar fram arkitekturen i ett projekt. Fokus för SAD-arbetet är generellt att finna en lämplig uppdelning av funktionalitet för att skapa en stabil lösning, samtidigt som de delar som utvecklas i största möjliga utsträckning ska kunna gå att återanvändas.

Om utvecklarna själva tar fram SAD-en är det stor risk att de vill sätta sin personliga prägel på hur applikationen ska vara utformad, mest kanske för att de som programmerare har ett inarbetat tillvägagångssätt som de trivs med. En av konsekvenserna blir att återanvändningsgraden av koden blir i låg. Ytterligare ett problem är att viktig kunskap återfinns hos ett fåtal personer, vilket skapar problem när de inte finns kvar på företaget.

5.2 Tjänsteorienterad arkitektur

Ibitec bygger sina system efter principen med tjänsteorienterad arkitektur. Definitionen av tjänsteorienterad arkitektur (på engelska kallad SOA, Service Oriented Architecture) kan i sin enklaste form ses som en arkitektur som kan delas in i tre kategorier av komponenter enligt Modi (2004). En tjänst, en tillhandahållare av tjänsten och en användare av tjänsten. SOA är en samling tjänster i ett nätverk vilka kommunicerar med varandra. En tjänst måste även vara möjlig att upptäcka och använda på ett dynamiskt sätt. En registertjänst måste därför finnas tillgänglig för både den som tillhandahåller tjänsten och den som använder tjänsten. Tillhandahållare registrerar sina tjänster i registret och användarna söker i registret för att hitta en tjänst.

5.2.1 Tjänst

För att verkligen förstå vad en tjänsteorienterad arkitektur är behövs förståelse för vad en tjänst är. En tjänst är en applikation som har en lös koppling till andra applikationer. Lös koppling innebär att tjänsten och de applikationer som använder tjänsten kan förändras helt oberoende av varandra. Applikationerna behöver inte inneha några tekniska detaljer om andra applikationer för att kunna kommunicera med dem. Tjänsterna ska ha väldefinierade, plattformsoberoende gränssnitt och vara återanvändbara. En tjänst och dess användare uppnår integration genom att utbyta meddelanden. Det enda som delas av dem är beskrivningar av hur dessa meddelanden skall se ut. Webber (2004).

5.2.2 Web services

Ett extrakrav på ärendehanteringsmotorn var att den skulle vara åtkomlig som Web services. Fördelen med att kunna komma åt den via Web services är att den då kan användas i ett distribuerat system som inte behöver köras på samma server som det övriga ärendehanteringssystemet. Dessutom finns då möjlighet att använda den mot andra system än sådana som är baserade på plattformen .NET.

World Wide Web Consortium, W3C (2004), definierar en Web service enligt följande: “A Web service is a software application identified by a URL, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols."

Ett annat sätt att beskriva en Web service är att se det som en distribuerad mjukvarukomponent som nås via standardiserade webbprotokoll.

Web services består av fyra tekniker: XML, SOAP, WSDL och UDDI. Web Service Basics (2004).

XML (Extensible Markup Language)
(Informationsbäraren – språket)

XML används för att strukturera och märka information så att den blir lättare att hålla reda på, söka i, publicera och återanvända. Dess struktur baseras på för ändamålet definierade etiketter eller så kallade taggar. Med hjälp av XML kan uppsättningar av taggar definieras som är speciellt anpassade till den information som ska beskrivas. Dessa taggar delar upp ett dokument i en hierarkisk struktur och identifierar dokumentets olika delar eller element. Till skillnad från HTML som är ett rent märkspråk kan användaren själv definiera taggarna och därigenom skapa sin egen struktur anpassad för lagringen av data. XML används ofta för att beskriva information som skickas på Internet.

SOAP (Simple Object Access Protocol)
(Paketeringen av informationen – kuvertet)

SOAP är ett enkelt XML-baserat protokoll som är designat för utbyte av strukturerade data på webben. Kommunikationen mellan Web servicen och de anropande klienterna fungerar vanligtvis som en klient/server-modell. Klienten gör en förfrågan gentemot Web servicens metoder. Web servicen bearbetar förfrågan och returnerar data eller eventuellt felmeddelande tillbaka till klienten. SOAP fungerar i likhet med ett kuvert som innehåller meddelandet. Beträffande transporten av SOAP-meddelanden är de i grunden utvecklade för HTTP men med version 1.1 av SOAP, kom möjligheten att använda andra protokoll såsom SMTP eller FTP. HTTP är dock det vedertagna protokollet vid kommunikation med Web services. En stor fördel med HTTP är att den vanligtvis tillåts att passera brandväggar utan att dessa behöver modifieras.

WSDL (Web Service Definition Language)
(Beskriver tjänsten och hur den fungerar)

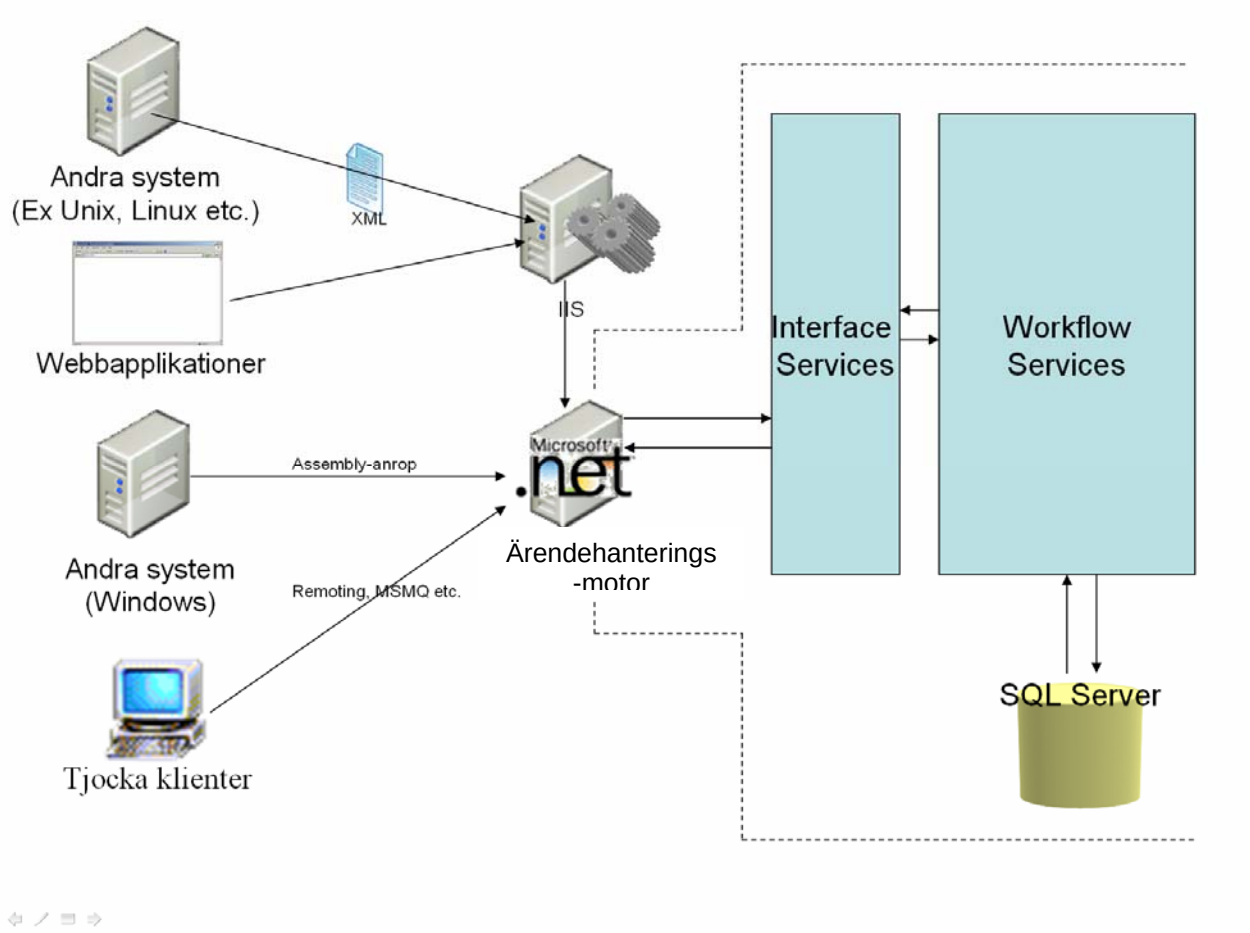
För att applikationer skall kunna utnyttja en Web service behöver de information om var Web servicen är belägen och vilka funktioner den erbjuder. Web servicen är inte självbeskrivande i sig utan förlitar sig på ett tillhörande WSDL-dokument. WSDL är ett språk baserat på XML. Via WSDL-dokumentet kan Web servicen förmedla de metoder/funktioner den tillhandahåller samt hur de anslutna applikationerna kan få tillgång till desamma.

UDDI (Universal Description, Discovery and Integration)
(Tillhandahåller en katalog över tillgängliga tjänster)

För att utnyttja Web servicen är klienterna tvungna att lokalisera WSDL-dokumentet och därigenom skapa en proxy gentemot Web servicen.

5.2.3 Ärendehanteringsmotorns omgivning

Nedanstående bild visar ärendehanteringsmotorns tänkta omgivning:



Figur 12. Ärendehanteringsmotorns omgivning.

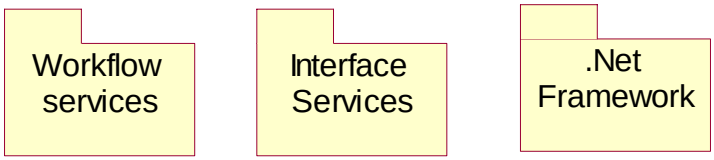
Ärendehanteringsmotorn är en modul som är tänkt att användas av ett ärendehanteringssystem. Ärendehanteringsmotorn ska vara anpassad för att kunna kommunicera med olika system som tjocka klienter, andra Windowssystem eller Microsofts webbserver IIS. Webbservern kan i sin tur kommunicera med till exempel webbapplikationer eller system som Unix eller Linux via Web services.

5.3 Komponentbaserad arkitektur

Med komponentbaserad arkitektur avses utformningen av system bestående av separata och autonoma komponenter som kan kopplas samman. Fördelarna med en komponentbaserad arkitektur är att en komponent kan bytas ut eller läggas till utan att påverka det övriga systemet. Motsatsen till komponentbaserad arkitektur är monolitisk arkitektur, där enskilda självständiga komponenter inte kan särskiljas. Målet med en komponentbaserad arkitektur är att systemet ska bestå av ett antal komponenter som sedan ska kunna återanvändas och på så sätt minska utvecklingskostnaderna. LEGO är en bra jämförelse, där varje legobit är en komponent. När ett nytt LEGO-slott ska skapas behöver bara ett par specialbitar läggas till, men i övrigt återanvänds redan befintliga produkter. Målet bör dock inte vara att göra allt återanvändbart, risken finns att systemet blir onödigt komplicerat.

Moduler är en vanlig komponentteknik som bygger på att varje komponent kompileras för sig. I Microsoft .NET kan varje komponent delas upp i ett eller flera assemblies, där varje assembly skapar en egen dll-fil vid kompilering. Varje komponent har minst ett eget assembly. Uppdelningen i olika fysiska komponenter gör det enklare att bygga ut och ändra i enbart valda delar av systemet utan att behöva kompilera om hela applikationen.

På den mest abstrakta nivån är ärendehanteringsmotorn uppdelad i paketen Workflow Services, Interface Services och Microsoft .NET Framework.



Figur 13. Abstrakt uppdelning av ärendehanteringsmotorn i komponenter.

Beskrivning av de olika paketen:

- **Workflow services**
  Innehåller all logik för ärendehanteringsmotorn. Här finns komponenter för att hantera arbetsflöde, regler, ärende, loggning med mera.
- **Interface Services**
  Innehåller tjänster som agerar gränssnitt mot externa applikationer, exempelvis Microsoft Sharepoint eller Web service konsumenter.
- **.NET Framework**
  Microsoft .NET plattformens klassbibliotek.

5.3.1 Workflow Services

Paketet Workflow services är i sin tur uppdelat i följande paket:

- **Issue**
  Tjänster för hantering rörande ärenden, exempelvis skapa och uppdatera ärenden samt ärendespecifik information som metadata. Innehåller också funktionalitet för att hantera rättigheter för ärenden.
- **Workflow**
  Tjänster för hantering av arbetsflöde generellt, exempelvis funktionalitet för att flytta ett ärende vidare från ett tillstånd till ett annat.
- **WorkflowTask**
  Tjänster för hantering av specifika steg i ett arbetsflöde även kallat tillstånd, exempelvis hantering av arbetsuppgifter och manuella beslut. Här finns också funktionalitet för exekvering av händelser som kan finnas i olika tillstånd.
- **RulesEngine**
  Tjänster för hantering av regelverk för både arbetsflöde och

arbetssteg. Exempelvis regler för hur länge ett ärende får ligga med oförändrad status innan något annat steg i arbetsflödet ska aktiveras.

- **Log**
  Tjänster för hantering av loggning och kommentarer som hör till ärenden.
- **Surviellance**
  Tjänster för övervakning av ärenden, exempelvis tidsgränser för ärenden i systemet.

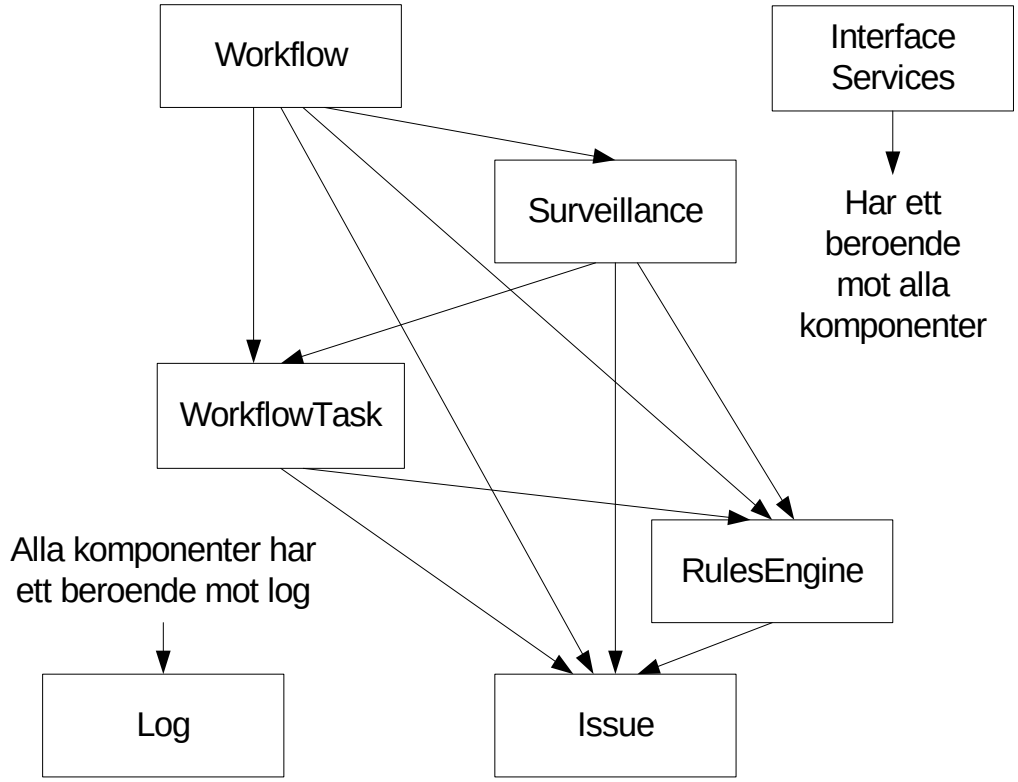
5.3.2 Interface Services

För att i största möjliga mån skapa en stabil kärna för ärendehanteringsmotorn, läggs alla externa gränssnitt till ärendehanteringstjänsterna ut i ett eget paket. Detta paket kan innehålla både Web service-gränssnitt och vanliga .NET assemblygränssnitt. Det finns då möjligheter att i framtiden lägga till motsvarande gränssnitt för kommunikation via andra åtkomsttekniker.

Interface Services används är för att inte blanda ihop åtkomsttekniker med ärendehanteringskärnan. För gränssnittets skull spelar det ingen roll om det är BizTalk, Sharepoint eller någon annan applikation som vill komma åt funktionaliteten, de använder på en viss nivå alla samma gränssnitt. När ärendehanteringsmotorn sedan ska användas i ett nytt sammanhang med delvis nya krav, behöver förhoppningsvis inte kärnan ändras och äventyra kompabiliteten för andra kunder.

5.3.3 Beroenden mellan systemets komponenter

Nedan visas beroenden mellan de olika komponenterna. Eftersom komponenten Surveillance är beroende av Workflow kan inte Workflow vara beroende av Surveillance. Om det var fallet skulle en cirkulär referens uppstå vilket inte är tillåtet mellan assemblies. Komponenterna måste således ordnas i en hierarki där komponenter enbart kan ha beroenden neråt i hierarkin. Samtliga komponenter har ett beroende till komponenten log. Överst i hierarkin är komponenten Interface Services som innehåller gränssnitt mot utomstående system.



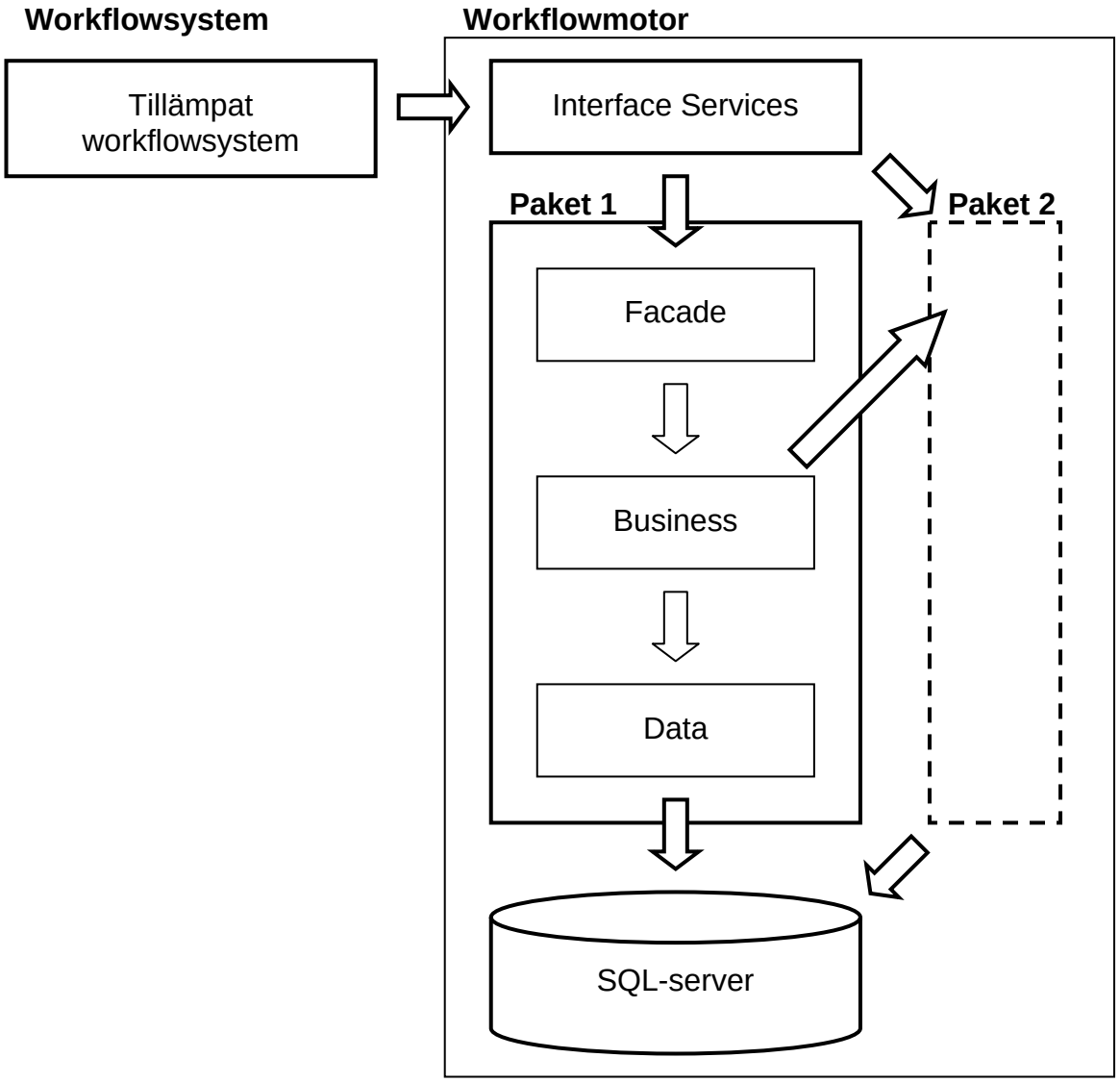
Figur 14. Visar beroenden mellan systemets olika komponenter.

5.4 Lagerstruktur

All funktionalitet i applikationen realiseras i form av tjänster. En tjänst består alltid av en Facade-klass samt en eller flera Business-klasser och Data-klasser.

- **Facade** – Fungerar som en portvakt till tjänsten. Här kan behörighetskontroll ske samt koordinering mot underliggande klasser.
- **Business** – Affärslogik för tjänsten. Här finns all logik som komponenten innehåller. För att kommunicera med databasen används tillhörande dataklass.
- **Data** – Sköter all access mot databasen. Anrop mot databasen sker genom stored procedures som innehåller logik för att hämta, lägga till, uppdatera och ta bort data i databasen.

I figuren nedan illustreras lagerarkitekturen och hur den fungerar i sitt sammanhang. Längst upp i ärendehanteringsmotorn ligger Interface Services. Genom detta lager sker all kommunikation med ärendehanteringsmotorn från kringliggande ärendehanteringssystem. Interface Services använder sig i sin tur av de olika paketens fasadklasser. Om en business-klass behöver information från ett annat paket, sker detta alltid via det paketets fasadklass, för att upprätthålla en ren arkitektur och säkerställa att inga behörigheter överskrids. Alla paket innehåller de tre lagren, och om något ska hämtas från databasen går det alltid genom dessa.



Figur 15. Översikt av paket- och lagerarkitektur

5.5 Dataåtkomst

I detta avsnitt beskrivs hur ärendehanteringsmotorn hanterar databasåtkomst.

5.5.1 Lagrade procedurer (stored procedures)

Ärendehanteringsmotorn använder endast lagrade procedurer vid kommunikation med databasen. Lagrade procedurer kan användas av en del databassystem som till exempel Microsofts SQL Server 2000. Ärendehanteringsmotorn använder sig uteslutande av den tekniken vid kommunikation med databasen. En lagrad procedur är precis som namnet antyder en procedur som innehåller SQL-kod och som lagras i databassystemet. När ett anrop mot databasen görs från en applikation anropas önskad procedur. Fördelen är att SQL-kod kan flyttas från applikation till databasen. En annan fördel är att den lagrade proceduren

körs snabbare eftersom den oftast är kompilerad. MSDN Stored Procedures (2004).

En nackdel uppstår vid byte av databassystem. Då måste de lagrade procedurerna skivas om, alternativt måste SQL-koden lyftas in i applikationen om databassystemet inte har stöd för lagrade procedurer.

5.5.2 Dataset

All kommunikation med databasen görs via dataset. Dataset ingår i ADO .NET (Active Data Objects) som är Microsoft .NETs API för dataåtkomst. De används framförallt för att hämta och spara information vid kommunikation mot en databas eller XML-fil. Ett dataset kan beskrivas som en databehållare som innehåller en eller flera tabeller. Varje tabell innehåller rader och kolumner, precis som tabellerna i en databas. Tabellerna kan också ha relationer mellan varandra. För att hämta data ur ett dataset anges först namnet på aktuell tabell, sedan vilken rad och till sist vilken kolumn. Det finns också funktionalitet i .NET för att enkelt loopa igenom en hel tabell för att presentera eller ändra på samtliga värden.

Ett typiskt exempel på användning av dataset är att först läsa in data från en tabell i en databas varefter en eller flera rader läggs till, tas bort och/eller modifieras i datasetet. Därefter används funktionalitet i .NET för att uppdatera tabellen i databasen mot datasetet. MSDN Dataset (2004).

5.6 Datamodell

Ärendehanteringsmotorn är uppbyggd kring en datamodell som togs fram under designfasen. Eftersom datamodellen är mycket central för hur produkten kommer att se ut i framtiden, och dessutom är relativt invecklad att ändra på i efterhand, har det lagt mycket tid på att göra den så fullständig som möjligt. Som ett första steg ritades ett ER-diagram, ur vilket tabellerna kunde tas fram. Men hjälp av användningsfallen kunde nu datamodellen testas och modifieras. I Appendix B återfinns de mest centrala tabellerna med beskrivning.

Datamodellen innehåller all funktionalitet som behövs för att spara ärenden, metadata, arbetsuppgifter, beslut och information om hur ärenden har vandrat i respektive arbetsflöde. Ärenden identifieras med hjälp av ett unikt ärendenummer. Datamodellen har stöd för att sätta rättigheter för hela systemet, olika typer av ärenden och för enskilda ärenden. Inget stöd finns för användarkonton eller till vilka roller som olika användare hör. Istället måste ett utomstående användarsystem kopplas till ärendehanteringsmotorn.

Databasen körs på databasservern Microsoft SQL Server 2000. Genomgående används stored procedures för åtkomst av databasen. Stored procedures används av effektivitetsskäl och för att det blir en snygg uppdelning mellan C#-kod och SQL-kod. Nackdelen är, om ett annat databassystem ska användas, att alla stored procedures måste skrivas om i det nya databassystemet.

5.7 Loggning

Loggning av händelser knutna till ärenden är en viktig del i ärendehanteringsmotorn, detta eftersom det ska gå att ta fram historik som visar vad som har hänt med ärenden och hur de har vandrat i arbetsflödet. Ingen viktig information får heller tas bort ur databasen. Om användaren till exempel vill ta bort ett ärende ska det ändå finnas kvar i databasen men markerat som borttaget. Samma sak gäller för information som tillhör ärenden, som exempelvis arbetsuppgifter, beslut och metadata. Metadata ska även kunna spåras vilket betyder att alla gamla värden som ett visst metadata har haft ska finnas kvar.

Eftersom att loggning av data är en central del i ärendehanteringsmotorn finns en egen loggkomponent. Viss data loggas dock direkt i den aktuella komponenten, eftersom det finns risk att det blir onödigt krångligt att implementera annars.

Ett problem med att många händelser ska loggas är att det snabbt kan bli mycket data i databasen. Eftersom inga ärenden med tillhörande information ska kunna tas bort ur databasen finns en risk att vissa av tabellerna kan komma att innehålla många poster. Ett krav är att ett ärende måste kunna finnas kvar i databasen i minst 10 år.

5.8 Felhantering

Någon typ av felhantering måste finnas. Eftersom produkten enbart består av ett gränssnitt av metoder som utomstående system använder sig av, måste detta system på något sätt meddelas när ett fel uppstår. Exempel på enkla fel som kan uppstå är att någon vill hämta ärenden eller arbetsuppgifter som inte finns i systemet. Ett annat fel kan vara att en användare försöker utföra en åtgärd som denne inte har rättigheter till.

Plattformen Micrsoft .NET och språket C# som används vid utveckling av ärendehanteringsmotorn har stöd för undantagshantering (på engelska exception handling). För den typ av fel som uppstår när ärendehanteringsmotorn är i drift har undantagshantering använts för att hantera fel. Ett antal olika undantagsklasser har skapats som kastas beroende på vilken typ av fel som har inträffat. På detta sätt kan utomstående system få reda på vilket typ av fel som inträffat och i sin tur vidta lämplig åtgärd.

En annan typ av fel som kan uppstå är fel i konfigurationen, som består av ett antal XML-filer. Dessa filer ska kontrolleras efter fel när ärendehanteringsmotorn initieras. Om någon eller några av filerna innehåller fel ska ärendehanteringsmotorn inte starta och användaren meddelas om vilka fel som finns i konfigurationsfilerna. Även en extern applikation ska utvecklas för att det ska finnas möjlighet att kontrollera att konfigurationsfilerna är korrekta innan ärendehanteringsmotorn startas.

5.9 Utbyggnadsmöjligheter av regler och händelser

Eftersom beslutsregler, övervakningsregler och händelser kan vara komplexa är det en klar fördel om dessa går att implementera i efterhand och pluggas in i ärendehanteringsmotorn. Detta för att det ska vara möjligt att på ett enkelt och smidigt sätt utöka ärendehanteringsmotorns funktionalitet och därmed dess användningsområden.

En lösning har tagits fram där själva logiken för beslutsregeln, övervakningsregeln eller händelsen implementeras i språket C#, samma som ärendehanteringsmotorn. Detta möjliggör att avancerade och komplexa regler kan implementeras om behov finns.

Om en ny beslutsregel ska skapas görs detta i en ny klass. Klassen ärver av en basklass som används vid skapandet av nya beslutsregler. Basklassen är en abstrakt klass som endast innehåller ett antal metoder som beskriver interfacet mot ärendehanteringsmotorn. När sedan den nya beslutsregeln ska användas måste en konfiguration av denna skapas. Konfigurationen skapas i en XML-fil som beskriver vilka argument som ärendehanteringsmotorn ska skicka med till beslutsregeln. Tanken är att beslutsregeln inte själv ska hämta information direkt ur databasen utan den ska bli försedd med detta av ärendehanteringsmotorn. Argument som kan skickas med är exempelvis ärendets status, deadline eller valfritt metadata som tillhör ett ärende. Flera olika uppsättningar av konfigurationer för en regel kan användas samtidigt.

Principen är likadan för skapandet av nya övervakningsregler och händelser. Endast ett par grundläggande regler och händelser kommer att implementeras i examensarbetet. Ärendehanteringsmotorn får sedan utökas när behov uppstår.

6 Resultat och utvärdering

I detta kapitel behandlas hur resultatet som erhållits uppfyller de mål och krav som ställdes i början av examensarbetet, de problem och förändringar av kraven som uppstått under arbetes gång samt de framtida utvecklingsmöjligheter som finns.

6.1 Resultat

Det har visat sig under examensarbetets gång att det inte var en helt enkel uppgift att designa och utveckla en generell ärendehanteringsmotor. Området är mycket stort och det är svårt att ta fram en riktigt bra och generell produkt på den korta tid som ett examensarbete omfattar. Ska alla standarder följas som rör olika gränssnitt med mera behövs betydligt mer tid att tillgå. Meningen var att ärendehanteringsmotorn skulle vara liten och enkel men ändå generell. Dessutom har inte kraven från kravställarna varit helt tydliga vad gäller funktionalitet och vilka typer av arbetsflöden som skulle stödjas.

Produkten som har utvecklats uppfyller ändå de primära krav som togs fram genom användningsfall och kravspecifikation. Den stora frågan är hur pass flexibel och generell ärendehanteringsmotorn är. Det är givetvis svårt att svara på eftersom produkten ännu inte används i något ärendehanteringssystem. Ärendehanteringsmotorn ska användas i ett annat projekt som är under utveckling på Ibitec. På grund av det projektet kom kravbilden att ändras under implementationen, för mer information om detta se avsnitt 6.6.4. I skrivande stund är det projektet fortfarande under utveckling men har ändå kommit så långt att det drar nytta av viss funktionalitet i ärendehanteringsmotorn. Exempel på funktionalitet som används med lyckat resultat är ärendehantering med tillhörande metadata samt arbetsflöden av typen ad hoc.

6.2 Funktionalitet i ärendehanteringsmotorn

Nedan listas de viktigaste funktionaliteterna som finns implementerat i ärendehanteringsmotorn.

Konfigurering

- Ärendetyper och arbetsflöden konfigureras med hjälp av XML-filer. Till en ärendetyp definieras vilken metadata som ska finnas, vilket arbetsflöde som ska användas samt möjligheten att tilldela roller standardrättigheter för ärendetypen.

Hantera ärenden

- Skapa odefinierat ärende vilket innebär att ärendet inte tillhör någon ärendetyp. Ärendet kan då tilldelas ärendetyp vid ett senare tillfälle. Denna funktionalitet kan användas när ett ärende kommer in i

systemet men då det för tillfället inte är känt vilken ärendetyp som det tillhör.

- Skapa definierat ärende då det är känt vilken ärendetyp som ärendet tillhör.
- Aktivera ärende. Ett ärende får statusen inaktivt efter det skapats och måste aktiveras för att kunna börja sin vandring i arbetsflödet.
- Ett ärendes deadline och prioritet kan hämtas samt uppdateras.
- Ärenden har stöd för kommentarer.
- Rättigheter kan sättas till hela systemet samt till specifika ärenden.
- Det går att söka efter ärenden på en rad olika sätt. Exempelvis kan ärenden hämtas med avseende på status, användare och metadata.
- Metadata som tillhör ett ärende kan hämtas, läggas till och uppdateras.

Fasta flöden

- Arbetsflödet innehåller tillstånd och hur dessa är kopplade till varandra. Varje tillstånd kan i sin tur innehålla automatiska händelser, arbetsuppgifter, underärenden samt ett automatiskt eller ett manuellt beslut. För mer information se avsnitt 3.4.1 som behandlar kraven för det fasta flödet.
- Ett ärende kan nollställas vilket innebär att det återställs till ursprungsläget. Alla arbetsuppgifter, underärenden och beslut nollställs.
- Underärenden går att skapa till ett ärende. Ett underärende är ett vanligt ärende av valfri ärendetyp som läggs till i ett tillstånd i förälderärendet. Detta innebär att när tillstånd som innehåller underärenden blir aktiva aktiveras dess underärenden. Underärendena måste utföras innan förälderärendet kan fortsätta i arbetsflödet.
- Ett ärende kan avslutas manuellt innan det kommit fram till sluttillståndet i arbetsflödet. Att avsluta ett ärende innebär att det markeras som arkiverat i databasen.
- Utöver de arbetsuppgifter som redan finns definierade i flödet går det att skapa och lägga till nya arbetsuppgifter när ärendet är aktivt.
- Arbetsuppgifter och manuella beslut går att tilldela både roller och användare.

Ad hoc-flöden

Ad hoc-flöden i ärendehanteringsmotorn fungerar precis som beskrivits i avsnitt 3.4.2.

- När ett ärende startas anges vem eller vilka användare som ska ha ärendet från start.
- Skicka ärendet vidare till en eller flera användare.
- Skicka tillbaka ärendet till en eller flera användare. Observera att ärendet endast kan skickas tillbaka till någon av de användare som det kommer ifrån.
- Hämta ärenden med avseende på aktiva användare.
- Hämta aktiva användare som har ett ärende.

Utbyggnadsmöjligheter
Möjlighet finns att skapa egna automatiska beslutsregler samt att konfigurera dessa. Nya regler skapas i C#-kod genom arv av en basklass. Konfigurationer av regeln görs med hjälp av en XML-fil. På samma sätt finns det möjlighet att skapa samt konfigurera automatiska händelser och övervakningsregler. För mer information se avsnitt 5.9.

Endast några få grundläggande regler och händelser har implementerats för att exemplifiera hur detta kan gå till. Sedan får ärendehanteringsmotorn utökas med ny funktionalitet i form av nya regler och händelser när behov uppstår.

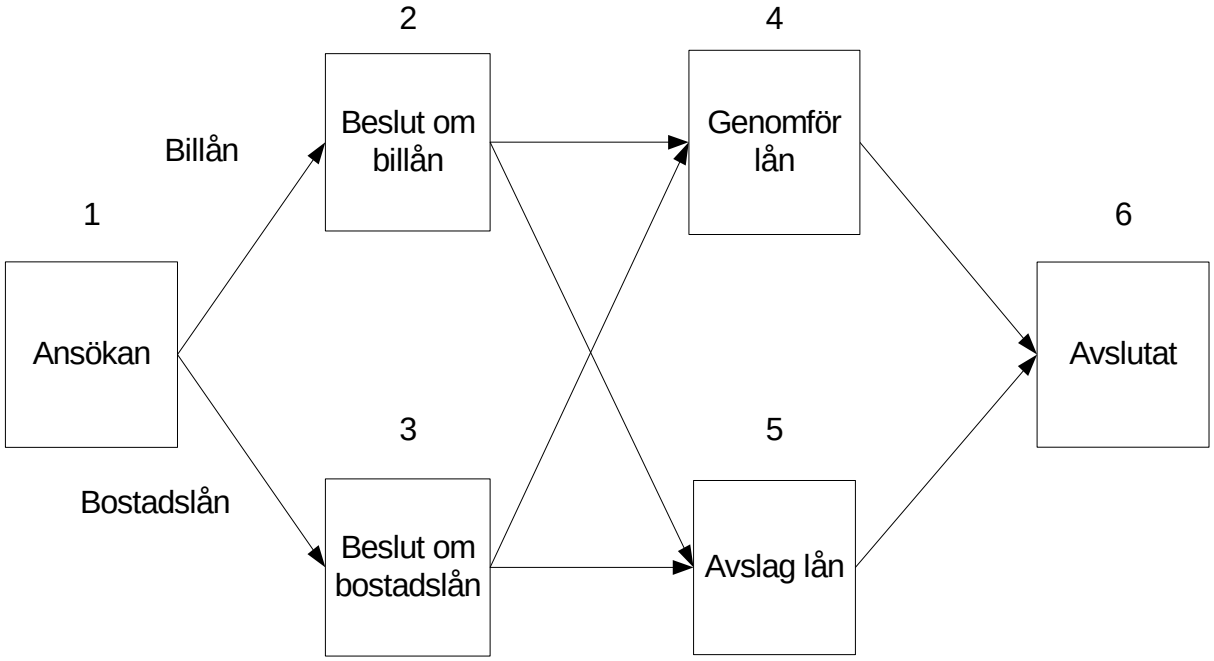
Följande har implementerats:

- **Beslutsregler**
  - CompareTo
    Tar två parametrar som argument och returnerar -1, 0 eller 1 beroende på om första argumentet är mindre, lika eller större än det andra. Båda argumenten måste vara av samma datatyp. I konfigurationen av regeln anges vilka två argument som ska skickas med.
- **Övervakningsregler**
  - CompareTo
    Tar två parametrar som argument och ett alarm aktiveras om dessa är lika. I konfigurationen av regeln anges vilka två argument som ska skickas med.
  - CheckStateIdleTime
    Kontrollerar hur länge ett tillstånd varit aktivt. Om tiden överskriver den som finns angiven i konfigurationen kommer ett alarm att aktiveras.
- **Händelser**
  - SendMail
    Skickar automatiskt e-postmeddelanden.

6.3 Exempel på användning av ärendehanteringsmotorn

I detta avsnitt ges ett exempel på hur ärendehanteringsmotorn kan användas. Genom att konfigurera den för flödet i avsnitt 2.2 som behandlar ett låneärende belyses stora delar av flödesfunktionaliteten. Alla tillstånd i exemplet kommer inte att behandlas och all XML-kod kommer inte att presenteras. I Appendix A0 finns de fullständiga XML-konfigurationerna.

I figuren nedan beskrivs de olika stegen som tillstånd.



Figur 16. Låneärende på tillståndsform.

För att ha något att utgå ifrån definieras först ärendetypen upp enligt nedan.

```
<ErrandType errandTypeId="1" workflowId="1">

    <Metadata metadataId="1" type="string"
      languageDependent="no">
       <MetadataTitle languageId="1">Namn</MetadataTitle>
    </Metadata>

    <Metadata metadataId="2" type="string"
      languageDependent="no">
      <MetadataTitle languageId="1">
         Typ av lån
      </MetadataTitle>
    </Metadata>

    <Metadata metadataId="3" type="float">
      <MetadataTitle languageId="1">Belopp</MetadataTitle>
    </Metadata>

</ErrandType>
```

De olika metadatafälten som definieras för ärendet är kundens namn, vilken typ av lån det rör sig om, och vilket belopp som ansökan avser. För varje

fält anges vilken datatyp som ska användas och en titel som kan anges på flera olika språk om så önskas.

Tillstånd 1: Ansökan
Låneansökan har kommit in till banken. Detta kan ske till exempel via bankens webbsida eller via kundmottagningen. I ärendehanteringsmotorn läggs ett automatiskt beslut som dirigerar ärendet till rätt tillstånd beroende på om det är ett bil- eller bostadslån. Detta beslut konfigureras på följande vis:

```
<AutomaticDecision automaticDecisionId="1"
  decisionRuleName="CompareTo">

  <Argument type="metadata" metadataId="2" />
  <Argument type="value" datatype="string">bil</Argument>
</AutomaticDecision>
```

Som attribut till det automatiska beslutet anges vilket id det ska ha samt vilken beslutsregel som ska användas. I detta fall ”CompareTo” som är en generell regel vilken redan finns implementerad i ärendehanteringsmotorn. De argument som ska skickas med till regeln anges i konfigurationsfilen och kommer att tolkas under körning. I detta fall ska metadatat ”Typ av lån” och strängen ”bil” skickas med för att jämföras.

Vid konfigurering av tillståndet anges vilket typ av beslut som ska appliceras och id på detta.

```
<State stateId="1" decisionType="auto" decisionId="1">
  <Title languageId="1">Ansökan</Title>
</State>
```

Tillstånd 2: Beslut om billån
I detta tillstånd fattas ett manuellt beslut om lånet ska godkännas eller inte. Det konfigureras på följande vis:

```
<ManualDecision manualDecisionid="1">
  <Title languageId="1">Beslut om billån</Title>
  <Description languageId="1">Godkänna låneärende
  </Description>

  <Assignments>
    <Role>3</Role>
  </Assignments>
</ManualDecision>
```

I beslutet anges titel och beskrivning samt eventuella roller som ska ha rättigheter att utföra beslutet som standard. Vilka val som kan göras definieras när övergångarna mellan tillstånden konfigureras senare i detta avsnitt. Som attribut anges att det är ett manuellt beslut som ska användas och vilket id det har. Det ligger även ett alarm på detta tillstånd, mer om det senare i detta avsnitt.

```
<State stateId="2" decisionType="manual" decisionId="1">
  <Title languageId="1">Godkännande bil</Title>
  <Alarm alarmConfigurationid="1" />
</State>
```

Tillstånd 4: Genomför lån

Detta tillstånd innehåller inget beslut utan en arbetsuppgift som ska utföras. Den konfigureras på detta vis:

```
<Task taskId="1" requiresAll="false">
  <Title languageId="1">Utför lån</Title>
  <Description languageId="1">
    Överför belopp till kunden.
  </Description>

  <Assignments>
    <Role>3</Role>
    <Role>4</Role>
  </Assignments>
</Task>
```

Som attribut anges arbetsuppgiftens id och om den behöver utföras av alla som har tilldelats den.

Tillståndet har inget beslut men istället en arbetsuppgift.

```
<State stateId="4">
  <Title languageId="1">Genomför lån</Title>
  <Task taskId="1" />
</State>
```

Övergångar mellan tillstånd

Efter att ha konfigurerat upp tillstånden placeras övergångarna mellan dem. Nedan konfigureras ärendehanteringsmotorn så att den kommer att dirigera ärendet från tillstånd 2 till tillstånd 4 eller 5 beroende på vilket svar som erhålls. Konfigurationen är oberoende av typen av beslut, det kan vara manuellt eller automatiskt. Som attribut anges från vilket tillstånd övergången ska gå och hur många ”aktiveringar” tillståndet måste ha tagit emot för att räknas som aktivt. Vid flöden som inte är parallella kommer detta värde alltid att vara 1.

```
<State stateId="2" mergeNumberOfArrivals="1">
  <NextState answer="approved">4</NextState>
  <NextState answer="rejected">5</NextState>
</State>
```

Övervakning

För att ett beslut om lånet ska godkännas och inte bli liggande, konfigureras ett alarm för detta. I alarmkonfigurationen nedan anges vilken alarmregel som ska användas, som i detta fall är ”CheckStateIdleTime”. Argumenten som skickas med är tidsrymden från det att tillståndet blev aktivt till den aktuella tidpunkten. Regeln jämför detta med nästa argument, som i det här fallet är en tidsrymd på ett dygn, och kommer att lägga ärendet under alarm om det varit inaktivt längre än tidsrymden.

```
<AlarmRuleConfiguration alarmRuleId="1"
  ruleName="CheckStateIdleTime">

  <Title languageId="1">Kolla stillastående ärende</Title>
  <Description languageId="1">
    Ett ärende får inte stå för länge i ett tillstånd
```

```
  </Description>

  <Argument type="stateidletime" />
  <Argument type="value" datatype="timespan">1.00:00:00
  </Argument>
</AlarmRuleConfiguration>
```

6.4 Utvärdering av produkten

I detta avsnitt diskuteras produktens egenskaper ur olika aspekter.

6.4.1 Plattformsberoende

Ärendehanteringsmotorn är implementerad i .NET-miljö vilket i sig är en begränsning då den endast kan användas på maskiner med Microsoft Windows. Detta var ett krav på produkten och plattformsberoendet innebär en nackdel.

6.4.2 Databasberoende

För närvarande körs ärendehanteringsmotorn mot en Microsoft SQL Server databas. Ärendehanteringsmotorn stödjer byte av databassystem, men databasen i sig innehåller lagrade procedurer (se avsnitt 5.5.1). Det innebär alltså att databassystemet måste ha stöd för lagrade procedurer och dessa måste föras över till det nya databassystemet.

6.4.3 Konfiguration av ärendehanteringsmotorn

Ett av de beslut som fattades under utvecklingen av ärendehanteringsmotorn var hur konfigurationen av ärendehanteringsmotorn skulle möjliggöras, se avsnitt 4.4. Det som valdes var konfiguration i XML, utan grafisk gränssnitt och utan stöd för Workflow Management Coalitions gränssnitt, se avsnitt 2.5. Vidare togs beslutet att inte implementera ett grafiskt gränssnitt då det skulle försvåra ändringar av ärendehanteringsmotorn.

XML-konfigurationen görs i flera filer som ansvarar för olika delar i flödet. En fil innehåller konfiguration av arbetsuppgifter, en annan innehåller alla tillstånd och så vidare. Nackdelen med flera filer är att det blir många id:n som länkas mellan filerna. Vinsten är att filerna blir mer överskådliga då det inte bildas en stor hierarki av XML-taggar.

Det naturliga sättet att utvärdera hur konfigurationen fungerar är att använda ärendehanteringsmotorn. Det har naturligtvis gjorts under implementationen men även efteråt. Ett exempel på en konfiguration återfinns i avsnitt 6.3. På de flöden som varit aktuella har konfigurationen skett smärtfritt. Tidsåtgången har inte varit speciellt krävande då flödena varit relativt små. Vid stora och komplicerade flöden skulle även ett grafiskt gränssnitt vara önskvärt. Det skulle också ge en lägre tröskel än inlärningen av XML-taggarna som tar en viss tid att lära sig.

Ärendehanteringsmotorn innehåller ett hjälpmedel, implementerat i projektet, som kontrollerar att konfigurationen är korrekt och lämnar ett meddelande om så inte är fallet. Till exempel valideras XML-filen mot ett

schema och giltigheten hos id:n kontrolleras. Detta visade sig vara mycket värdefullt då det annars är lätt att göra små syntax-fel i XML-filerna.

6.4.4 Arbetsflödet

Ärendehanteringsmotorn är inte beroende av olika typer av ärenden, som exempelvis låneärenden, fakturor eller dokument, då metadata är konfigurerbara med avseende på datatyp och antal kolumner. Dessutom kan flera olika ärendetyper med olika arbetsflöden köras samtidigt i systemet. De flödesmekanismer som finns i ärendehanteringsmotorn går att bygga ihop utan begränsningar. Det vill säga ett arbetsflöde kan ha ett oändligt antal tillstånd, arbetsuppgifter, underärenden och så vidare. Det går alltså att bygga upp mycket komplicerade flöden genom att använda de mekanismer som finns. Det är dock svårt att bedöma om mekanismerna i sig är tillräckligt flexibla.

6.4.5 Rättigheter

Ärendehanteringsmotorn har ett visst stöd för rättigheter. Det är ett mycket enkelt stöd för de allra enklaste systemen. Om systemets rättigheter kräver någonting utöver de mest grundläggande kraven måste detta implementeras utanför komponenten. Tanken från början var inte att rättighetssystemet i ärendehanteringsmotorn skulle vara komplett och det går enkelt att stänga av vid extern implementering.

6.4.6 Arkitektur

Det finns två aspekter av arkitekturen att utvärdera. Den ena är hur ärendehanteringsmotorn fungerar med kringliggande komponenter, se avsnitt 5.2. Den andra är hur ärendehanteringsmotorns arkitektur ser ut internt, se avsnitt 5.3.

Alla kringliggande komponenter går via lagret Interface Services när kommunikation sker med ärendehanteringsmotorn. Det gör att ärendehanteringsmotorns gränssnitt enklare kan anpassas och byggas ut för att agera som olika typer av tjänster, till exempel via ett Web services-gränssnitt.

Den interna komponentbaserade arkitekturen har för- och nackdelar. Fördelarna är att koden blir tydligt uppdelad i mindre moduler och att det inte uppstår ett nät av beroenden mellan olika delar i koden. Nackdelen är att komponenterna måste vara uppdelade i en hierarki av referenser. Det innebär att en komponent kan använda sig av en annan, men den andra kan inte använda sig av den första då det skapar en cirkulär referens. Det visade sig under implementationen att de olika komponenterna har ett starkt beroende av varandra vilket försvårar en uppdelning. En erfarenhet som kan dras av detta är att väl utformad arkitektur innebär mer arbete under implementationen. Lagerarkitekturen i ärendehanteringsmotorn är tydlig och skiljer de olika typerna av funktionalitet åt.

Ärendehanteringsmotorns arkitektur och design är anpassad för att relativt enkelt kunna byggas ut med ny funktionalitet. Som exempel på detta kan

nämnas att den enkelt kan byggas ut med nya beslutsregler, övervakningsregler och händelser. För mer information om detta se avsnitt 5.9. Den är även anpassad för att kunna byggas ut med helt ny funktionalitet, vilket underlättas av att den är uppdelad i flera komponenter som alla har var sitt ansvarsområde. Det blir då lättare att veta var den nya funktionaliteten hör hemma som ska implementeras. Som exempel på detta kan nämnas att den i efterhand fick byggas ut med stöd för ad hoc-flöden på grund av förändrad kravbild, vilket beskrivs närmare i avsnitt 6.6.4. Denna utbyggnad gick enklare än väntat att realisera, vilket vittnar om att flexibiliteten i arkitekturen är stor.

6.4.7 Prestanda

Någon prestandamätning på ärendehanteringsmotorn har inte gjorts. Den är dock designad för att klara av en rimlig mängd ärenden, cirka 30 000 per år. Den kritiska punkten bedöms vara databasen och dess växande storlek. Framförallt loggtabellerna i databasen har en tendens att växa snabbt och bli stora.

6.5 Utvecklingsmöjligheter

I detta avsnitt presenteras framtida förbättringar och utvecklingsmöjligheter av ärendehanteringsmotorn.

6.5.1 Integration med SharePoint

SharePoint en produkt från Microsoft vars främsta syfte är att på ett enkelt sätt erbjuda möjligheten att skapa arbetslagsorienterade portaler (se avsnitt 1.7.4).

SharePoint innehåller dokumenthantering och det är här ärendehanteringsmotorn kan användas. Den skulle kunna integreras och på så vis skulle dokumenthanteringen realiseras som ärenden. Från början skulle även denna del med integrering mot SharePoint ingå i examensarbetet. Eftersom kravbilden kom att ändras under implementationen av ärendehanteringsmotorn, innebär detta att denna del fick skäras bort på grund av tidsbrist. För mer information om förändrad kravbild se avsnitt 6.6.4.

6.5.2 Standardiserade gränssnitt

För att öka kompabiliteten med andra ärendehanteringsmotorer och ärendehanteringssystem kan det vara lämpligt att vid behov implementera stöd för ett standardiserat gränssnitt, lämpligtvis de framtagna av Workflow Management Coalition (WfMC), se avsnitt 2.5. En stor fördel är exempelvis att en standardprodukt kan användas för att grafiskt konfigurera arbetsflöden. Det krävs dock relativt stora ingrepp i ärendehanteringsmotorn för att åstadkomma detta.

6.5.3 Grafiskt gränssnitt för konfiguration

För att underlätta konfigurationen av ärendehanteringsmotorn skulle ett grafiskt gränssnitt kunna implementeras. En möjlig lösning på detta är att

implementera en windowsapplikation. Ett annat alternativ är att använda sig av ett existerande ritverktyg som till exempel Microsoft Visio, Visio (2004). I så fall måste någon form av konvertering göras ifrån Visios filformat till det format som ärendehanteringsmotorn använder för representation av arbetsflöden.

6.5.4 Filhantering

Det kan många gånger vara aktuellt att bifoga filer till ett ärende, till exempel dokument. Det finns därför skäl att bygga ut ärendehanteringsmotorn för att stödja detta.

6.5.5 Web services

Ärendehanteringsmotorn kan byggas ut med stöd för åtkomst via Web services. Detta för att ärendehanteringsmotorn och det övriga ärendehanteringssystemet ska kunna köras på olika datorer. Web services är ett standardiserat protokoll som gör att ärendehanteringsmotorn kan användas av andra system som till exempel Unix och Linux, vilket är en fördel.

Från början var denna del tänkt att ingå i examensarbetet men fick skäras bort på grund av tidsbrist. Det bedöms dock inte vara alltför tidskrävande att implementera stöd för åtkomst via Web services. Eftersom ärendehanteringsmotorn är uppbyggd enligt arkitekturen som beskrivs i avsnitt 5.3, är den anpassad för att kunna byggas ut med stöd för Web services. Det som behöver göras är att skapa en ny klass i komponenten Interface Services som innehåller samma metodgränssnitt som redan finns i det befintliga gränssnittet. Stöd för åtkomst via Web services måste läggas till.

6.5.6 Externt regelverk

För närvarande byggs beslutsregler, alarmregler och händelser i ärendehanteringsmotorn. Det kan i vissa fall bli aktuellt att koppla in ett externt regelverk, stöd för detta saknas i ärendehanteringsmotorn.

6.6 Utvecklingsprocessen

I detta avsnitt presenteras erfarenheter från utvecklingen av ärendehanteringsmotorn.

6.6.1 Metod

Examensarbetet har bedrivits i projektform efter en modifierad variant av RUP-modellen (se avsnitt 1.6). Det har varit en positiv upplevelse och har gett stöd i planerandet och genomförandet.

Ett problem som uppstod var att kraven ändrats under projektets gång och tidsplanen har omarbetats flera gånger på grund av detta. Vidare ändrades starttiden för ett parallellt projekt vilket innebar att kravspecifikationen inte kunde godkännas av samtliga beställare. Dessutom kom inte alla krav

samtidigt utan kraven kom efterhand. Detta fick till följd att utvecklandet av ärendehanteringsmotorn fördröjdes.

6.6.2 Problem under implementationen

Under implementeringen av ärendehanteringsmotorn har inga stora problem uppstått. Arbetet drog ut på tiden eftersom vissa delar tog längre tid att implementera än beräknat. Dessutom ändrades kravbilden under utvecklingen av systemet, vilket beskrivs närmare i avsnitt 6.6.4. Projektgruppen har tidigare erfarenheter av plattformen Microsoft .NET vilket gjorde att det gick relativt snabbt att komma igång med implementationen, för ytterliggare information om detta se avsnitt 6.6.3. Många konsulter på Ibitec har mycket goda kunskaper och erfarenhet av utveckling i .NET-miljön och de kunde hjälpa till när problem uppstod.

De problem som ändå uppstod under implementationen är följande:

- **Beroenden mellan komponenter**
  I designen som togs fram var det inte riktigt genomtänkt hur de olika komponenterna i ärendehanteringsmotorn var beroende av varandra. När väl implementationen kom igång visade det sig att viss funktionalitet fick flyttas mellan de olika komponenterna.
- **Global data**
  I designen var det inte tillräckligt specificerat vilka data som behövde vara åtkomlig i flera eller samtliga komponenter. Exempel på global data är den data som konfigurationsfilerna innehåller, då dessa annars skulle behöva läsas in flera gånger. En gång för varje komponent som är i behov av konfigurationsdata. Problemet löstes efter en diskussion med Ibitecs arkitekturgrupp genom att skapa en ny komponent där all global data placerades. Denna komponent läggs längst ner i hierarkin vilket innebär att den inte får ha något beroende till systemets övriga komponenter. Alla andra komponenter kan då ha en referens till komponenten som innehåller global data.
- **Komponenten Workflow**
  Implementationen av logiken som för ett ärende vidare i arbetsflödet var något mer komplex än vad som först hade uppskattats. Vilket innebar att den delen av implementationen tog längre tid än vad som hade beräknats.

6.6.3 Plattformen

Projektgruppens tidigare erfarenheter av Microsoft .NET och språket C# var inte stora men grundläggande kunskaper fanns vilket gjorde att implementationen kom igång snabbt. Kodningssyntaxen i C# är mycket lik Javas vilket gör att det går snabbt att komma igång om tidigare erfarenheter finns från liknande språk som Java eller C++. Det tog dock tid att få en överblick av .NETs standardbibliotek som är omfattande.

Plattformen .NET innehåller funktionalitet för att automatiskt generera dokumentation i form av webbsidor direkt från källkoden. Detta är likt

Javas motsvarighet Javadoc fast i .NET skrivs kommenterarna som ska vara med i dokumentationen med hjälp av XML-taggar, se exemplet nedan. XML Documentation (2004).

```
/// <summary>
/// Skapar ett ärende.
/// </summary>
/// <param name="userId">Den användare som utför
/// metodanropet.</param>
/// <param name="errandTypeConfId">Ärendetypen.</param>
/// <returns>Ärendets id.</returns>
public int CreateErrand(int userId, int errandTypeConfId)
{
  ...
}
```

Figur 17. Exempel på dokumentation i .NET.

Projektet har använt sig av ett externt open-source-verktyg kallat NDoc. NDoc har större möjligheter att konfigurera generering av dokumentation än det inbyggda stödet i .NET. NDoc (2004).

Att implementera ärendehanteringsmotorn i plattformen Microsoft .NET har varit en positiv upplevelse. De svårigheter som uppstod under implementationen berodde inte på problem med språket C# utan på andra orsaker.

Även konceptet att använda lagrade procedurer (se avsnitt 5.5.1) var helt nytt. Det gick emellertid snabbt att lära sig hur tekniken fungerar eftersom projektgruppen har tidigare erfarenheter av att jobba med SQL och databaser. Efter att ha arbetat med lagrade procedurer är dessa att föredra framför SQL-anrop direkt ifrån koden i utvecklingsspråket, eftersom det då blir en tydligare separering av SQL-kod och C#-kod. För mer information om lagrade procedurer och deras för- och nackdelar, se avsnitt 1.7.5.

6.6.4 Förändrad kravbild

Den kravbild som först togs fram med användningsfall samt kompletterande kravspecifikation förändrades under implementationen av ärendehanteringsmotorn. Det berodde på ett annat projekt som också var under utveckling skulle använda sig av ärendehanteringsmotorn. Det projektet drog ut på tiden och blev försenat vilket medförde att deras krav inte fanns tillgängliga vid framtagningen av arkitekturen och designen av ärendehanteringsmotorn. När väl implementationen av ärendehanteringsmotorn hade pågått ett tag, började nya krav från det andra projektet läggas till. Det stora problemet var att alla kraven inte kom på en gång utan de kom successivt.

Det visade sig då att en del extra funktionalitet måste läggs till för att det andra projektet skulle kunna använda sig av ärendehanteringsmotorn. Från början hade inte ärendehanteringsmotorn stöd för ad hoc-flöden vilket visade sig vara ett måste från deras sida. Detta ledde till att viss funktionalitet måste skäras bort från den ursprungliga kravbilden för att ärendehanteringsmotorn skulle kunna bli klar på utsatt tid.

7 Slutsatser

Här presenteras de slutsatser som har gjorts under examensarbetet. Syftet är att sammanfatta projektet och svara på problemställningarna i avsnitt 1.4.

7.1 Komponentens roll

Ärendehanteringsmotorn är spindeln i nätet. Det är den som hanterar flödet av ärenden i ett ärendehanteringssystem. Ärendehanteringsmotorn är inget komplett system i sig utan kräver ett eller flera omkringliggande system som utnyttjar dess funktionalitet och tillsammans skapar ett komplett ärendehanteringssystem. Det omkringliggande systemet behöver exempelvis tillhandahålla ett användargränssnitt för att användare ska kunna utnyttja ärendehanteringsmotorns funktionalitet.

7.2 Krav på komponenten

Begreppsmodellen har skapat en grundläggande översikt av ärendehantering och dess vokabulär har använts i projektet. Centralt för produkten var att den skulle implementeras i .NET-miljö och använda sig av SQL Server 2000. Användningsfallen som har tagits fram belyste kraven på ärendehanteringsdelen av produkten. Några centrala punkter som är generella för ärendehantering är:

- Skapa ärendetyper
- Skapa arbetsflöden
- Hantera användare och roller
- Registrera, bearbeta, söka och avsluta ärenden

7.3 Alternativa lösningar

Ett viktigt vägval som har gjorts under projektets gång var huruvida en standardprodukt skulle användas eller om ärendehanteringsmotorn skulle byggas från grunden. Några av fördelarna med att använda en standardprodukt är att det sparar tid, det kommer med extra funktionalitet på köpet och att uppgraderingar tillhandahålls. Några av nackdelarna är omvänt, att det kan ta tid att anpassa produkten, flexibiliteten kan vara för liten och att produkten kan vara dyr. Utifrån dessa fakta togs beslut om att utveckla en egen ärendehanteringsmotor från grunden eftersom ingen existerande produkt passade kravbilden.

Ett annat beslut som har tagits var hur konfigurationen av arbetsflödet skulle ske. Ett alternativ var att implementera ett standardgränssnitt och på det viset skulle existerande verktyg kunna användas. Ett annat alternativ var att göra en egen konfiguration till exempel i databastabeller eller i XML-filer. Att implementera ett standardgränssnitt bedömdes som ett för

omfattade arbete i relation till produktens omfattning. Därför gjordes valet att konfigurationen utförs i ett antal XML-filer.

Gränssnittet mot användande applikationer kunde också ha gjorts via ett standardiserat gränssnitt men det arbetet bedömdes vara för omfattande och istället gjordes ett eget gränssnitt bestående av metoder.

7.4 Arkitektur

Det finns två aspekter av arkitekturen att tänka på, dels internt i ärendehanteringsmotorn och dels hur ärendehanteringsmotorn fungerar med kringliggande komponenter.

Ärendehanteringsmotorn bygger på en tjänstebaserad arkitektur där dess översta lager är gränssnittet utåt som tillhandahåller ärendehanteringsmotorns funktionalitet i form av tjänster. Det gör att den är löst kopplad och kan på så vis enklare byggas in i olika system.

Komponenten är internt uppbyggd av flera olika komponenter i form av assemblies. Detta medför att koden blir tydligare indelad efter funktionsområde. Det blir även lättare att i framtiden utöka den med ny funktionalitet om ett sådant behov skulle uppstå. Varje delkomponent är i sig uppdelad i en lagerstruktur bestående av tre lager. Facade-lagret som tar hand om rättigheter och koordinerar mot businesslagret. Businesslagret som innehåller affärslogiken och data-lagret som tar hand om databaskommunikationen. Komponenterna är starkt beroende av varandra vilket har gjort att upprätthållandet av arkitekturen försvårade implementationen.

Databasåtkomsten sker alltid med datasets via stored procedures. Detta gör att C#-kod blir väl avskild från SQL-kod och modifiering av data blir flexibel.

7.5 Utvärdera produkten

Efter implementationen uppfyller ärendehanteringsmotorn de grundläggande krav som har tagits fram i kravanalysen. Det finns många aspekter på ärendehanteringsmotorn att utvärdera. Några av de intressantaste presenteras nedan:

- Eftersom ärendehanteringsmotorn är utvecklad i .NET är den bunden till att köras på maskiner med Microsoft Windows.
- Databasen är utbytbar men det krävs ett visst arbete med de lagrade procedurerna som måste överföras.
- Konfigurering av ärendehanteringsmotorn sker i form av XML-filer vilket innebär mer arbete och högre tröskel än om det hade varit ett grafiskt gränssnitt. Produkten levereras dock med en applikation för evaluering och felrättning av konfigurationen innan start av ärendehanteringsmotorn sker.

- Ärendehanteringsmotorn är oberoende av vilken typ av ärenden som ska behandlas och innehåller diverse mekanismer som kan fogas samman och skapa komplexa flöden. Om mekanismerna i sig är tillräckligt kraftfulla är svårbedömt.
- Det rättighetssystem som ingår i komponenten är enkelt och det krävs en extern implementation om större krav ställs.

7.6 Framtida förbättringar

Det finns ett antal områden på vilka ärendehanteringsmotorn kan förbättras och utvecklas. I det här avsnittet presenteras de viktigaste:

- Workflow Management Coalition har tagit fram standardiserade gränssnitt för kommunikation mellan ärendehanteringsmotorn och kringliggande komponenter. Genom att bygga om ärendehanteringsmotorn för att stödja dessa gränssnitt skulle många fördelar uppnås.
- Ett grafiskt gränssnitt för konfiguration av arbetsflöden skulle innebära en snabbare konfiguration och en lägre inlärningströskel. Nackdelen är att om ärendehanteringsmotorn skulle byggas om måste även det grafiska gränssnittet ändras.
- Filhantering är något som ärendehanteringsmotorn saknar stöd för. Det är vanligt att dokument bifogas till ärenden, varför filhantering skulle vara en önskvärd funktion.
- Som produkten ser ut för närvarande går den inte att komma åt via Web services. Endast vissa mindre ändringar skulle krävas för att göra detta möjligt. Fördelen med att nå ärendehanteringsmotorn via Web services är att den då kan användas i ett distribuerat system som inte behöver köras på samma server som det övriga ärendehanteringssystemet. Dessutom finns då möjlighet att använda den mot andra system än sådana som är baserade på plattformen .NET.
- Det kan i vissa fall vara aktuellt att koppla ett externt regelverk till ärendehanteringsmotorn för exempelvis automatiska beslutsregler. Ärendehanteringsmotorn saknar i nuläget stöd för detta.

8 Referenser

Allen (2001) — Allen, Rob (2001). Workflow: An introduction [www] <http://www.wfmc.org/standards/docs/Workflow_An_Introduction.pdf> Hämtat 26/11 2004.

Hollingsworth (1995) — Hollingsworth, David (1995). *Workflow Management Coalition - The Workflow Reference Model*. [www] <http://www.wfmc.org/standards/docs/tc003v11.pdf> Hämtat 29/11 2004.

IBM (2004) — IBM (2004). *Rational Unified Process* [www] <http://www-306.ibm.com/software/awdtools/rup/> Hämtat 14/12 2004

Javaworld (2004) — Javaworld (2004). *C# A language alternative or just J--* [www] <http://www.javaworld.com/javaworld/jw-11-2000/jw-1122-csharp1-p3.html> Hämtat 16/12 2004

Ljungberg (1996) — Ljungberg, Jan (1996). *Workflow Management – State of the Art.* SISU Publikation 96:21.

Microsoft .NET (2004) — Microsoft .NET (2004) [www] <http://www.microsoft.com/net/> Hämtat 30/11 2004

Microsoft .NET Framework FAQ (2004) — Microsoft .NET Framework FAQ (2004) [www] <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/faq111700.asp> Hämtat 7/12 2004

Microsoft BizTalk (2004) — Microsoft BizTalk (2004) [www] <http://www.microsoft.com/biztalk/> Hämtat 30/11 2004

Microsoft SharePoint (2004) — Microsoft SharePoint (2004). *Microsoft SharePoint Technologies* [www] <http://www.microsoft.com/sharepoint/> Hämtat 14/12 2004

Modi (2004) — Modi, Tarak (2004). *Service-oriented architecture* [www] <http://www.javaworld.com/weblogs/javadesign/archives/000130.html> Hämtat 7/12 2004

MSDN Dataset (2004)	MSDN Dataset (2004). *Creating and Using DataSets* [www] <http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcreatingusingdatasets.asp> Hämtat 16/12 2004
MSDN Stored Procedures (2004)	MSDN Stored Procedures (2004). *Stored Procedures*. [www] <http://msdn.microsoft.com/library/en-us/createdb/cm_8_des_07_31vb.asp> Hämtat 16/12 2004.
Ndoc (2004)	Ndoc (2004) *NDoc Online* [www] <http://ndoc.sourceforge.net/> Hämtat 14/12 2004
Plesums (2002)	Plesums, Charles (2002). Introduction to Workflow [www] <http://www.wfmc.org/information/introduction_to_workflow02.pdf> Hämtat 26/11 2004.
Schäl (1996)	Schäl, Thomas (1996). *Workflow Management Systems for Process Organisations*. Berlin: Springer. ISBN 3-540-61401-X.
Skelta Workflow .NET (2004)	Skelta Workflow .NET (2004). *A web based embeddable workflow engine - using Microsoft .NET framework* [www] <http://www.skelta.com> Hämtat 30/11 2004
Visio (2004)	Visio (2004*). Visio 2003 Product Information*. [www] <http://www.microsoft.com/office/visio/prodinfo/default.mspx> Hämtat 30/11 2004
Web Service Basics (2004)	Web Service Basics (2004) [www] <http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pul> Hämtat 7/12 2004
Webber (2004)	Webber, Jim & Parastatidis , Savas (2004). *Demystifying Service-Oriented Architecture* [www] <http://www.sys-con.com/webservices/article.cfm?id=706> Hämtat 7/12 200
Workflow Management Coalition (2004)	Workflow Management Coalition (2004). [www] <http://www.wfmc.org/about.htm> Hämtat 26/11 2004.

Workflow Management Coalition (Interface 2&3) (2004)	Workflow Management Coalition (Interface 2&3) (2004) *Workflow Management Application Programming Interface Specification* [www] <http://www.wfmc.org/standards/docs/if2v20.pdf> Hämtat 1/12 2004
Workflow Process Definition Interface (2004)	Workflow Process Definition Interface (2004), *Workflow Management Coalition* [www] <http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf> Hämtat 30/11 2004
XML Documentation (2004)	XML Documentation (2004). *XML Documentation* [www] <http://msdn.microsoft.com/library/en-us/csref/html/vcoriXMLDocumentation.asp> Hämtat 14/12 2004

Appendix A: XML-filer till låneärende-exempel

config_errandType.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för ärendetyper -->
<ErrandTypes>

  <ErrandType errandTypeId="1" workflowId="1">

    <Metadata metadataId="1" type="string" languageDependent="no">
      <MetadataTitle languageId="1">Namn</MetadataTitle>
    </Metadata>

    <Metadata metadataId="2" type="string" languageDependent="no">
      <MetadataTitle languageId="1">
      Typ av lån</MetadataTitle>
    </Metadata>

    <Metadata metadataId="3" type="float">
      <MetadataTitle languageId="1">Belopp</MetadataTitle>
    </Metadata>

  </ErrandType>

</ErrandTypes>
```

config_state.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för tillstånd -->

<States>
  <State stateId="1" decisionType="auto" decisionId="1">
    <Title languageId="1">Ansökan</Title>
  </State>

  <State stateId="2" decisionType="manual" decisionId="1">
    <Title languageId="1">Godkännande bil</Title>
    <Alarm alarmConfigurationid="1" />
  </State>

  <State stateId="3" decisionType="manual" decisionId="2">
    <Title languageId="1">Godkännande hus</Title>
    <Alarm alarmConfigurationid="1" />
  </State>

  <State stateId="4">
    <Title languageId="1">Genomför lån</Title>
    <Task taskId="1" />
  </State>

  <State stateId="5">
    <Title languageId="1">Avslag lån</Title>
    <Task taskId="2" />
  </State>

  <State stateId="6">
    <Title languageId="1">Avslutat</Title>
```

```
  </State>
</States>
```

config_workflow.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för arbetsflöde -->

<Workflows>

  <Workflow workflowId="1" startStateId="1" endStateId="6">

    <State stateId="1" mergeNumberOfArrivals="1">
      <!-- Om typen är "bil" -->
      <NextState answer="0">2</NextState>

      <!-- Om typen inte är "bil" -->
      <NextState answer="1">3</NextState>
      <NextState answer="-1">3</NextState>
    </State>

    <State stateId="2" mergeNumberOfArrivals="1">
      <NextState answer="approved">4</NextState>
      <NextState answer="rejected">5</NextState>
    </State>

    <State stateId="3" mergeNumberOfArrivals="1">
      <NextState answer="approved">4</NextState>
      <NextState answer="rejected">5</NextState>
    </State>

    <State stateId="4" mergeNumberOfArrivals="1">
      <NextState>6</NextState>
    </State>

    <State stateId="5" mergeNumberOfArrivals="1">
      <NextState>6</NextState>
    </State>

    <State stateId="6" mergeNumberOfArrivals="1">

    </State>

  </Workflow>

</Workflows>
```

config_task.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för arbetsuppgifter -->

<Tasks>

  <Task taskId="1" requiresAll="false">
    <Title languageId="1">Utför lån</Title>
    <Description languageId="1">Överför belopp till kunden.</Description>

    <Assignments>
      <Role>3</Role>
      <Role>4</Role>
    </Assignments>
```

```
  </Task>

  <Task taskId="2" requiresAll="false">
    <Title languageId="1">Ge avslag.</Title>
    <Description languageId="1">Informera kunden om avslag.</Description>

    <Assignments>
      <Role>3</Role>
      <Role>4</Role>
    </Assignments>
  </Task>

</Tasks>
```

config_decision.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för konfiguration av manuella
     beslutsregler som används vid automatiska vägval -->


<Decisions>
<!-- Konfiguration automatiska beslutsregler -->
  <AutomaticDecisions>

    <AutomaticDecision automaticDecisionId="1"
decisionRuleName="CompareTo">
      <Argument type="metadata" metadataId="2" />
      <Argument type="value" datatype="string">bil</Argument>
    </AutomaticDecision>

  </AutomaticDecisions>

  <!-- Konfiguration av manuella beslut -->
  <ManualDecisions>

    <ManualDecision manualDecisionid="1">
      <Title languageId="1">Godkännande bil</Title>
      <Description languageId="1">Godkänna låneärende</Description>

      <Assignments>
        <Role>3</Role>
      </Assignments>
    </ManualDecision>

    <ManualDecision manualDecisionid="2">
      <Title languageId="1">Godkännande bostad</Title>
      <Description languageId="1">Godkänna låneärende</Description>

      <Assignments>
        <Role>4</Role>
      </Assignments>
    </ManualDecision>

  </ManualDecisions>
</Decisions>
```

config_alarm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för övervakningsregler -->
```

```
<AlarmConfigurations>

  <AlarmRuleConfiguration alarmRuleId="1" ruleName="CheckStateIdleTime">
    <Title languageId="1">Kolla stillastående ärende</Title>
    <Description languageId="1">
      Ett ärende får inte stå för länge i ett tillstånd</Description>

    <Argument type="stateidletime" />
    <Argument type="value" datatype="timespan">1.00:00:00</Argument>
  </AlarmRuleConfiguration>

</AlarmConfigurations>
```

config_event.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Konfigurationsfil för händelser -->

<Events>
  <Event eventId="1" eventName="SendMail">

  </Event>
</Events>
```

Appendix B: Databastabeller

Databasen består av totalt 24 tabeller. Nedan beskrivs de mest centrala tabellerna för ärendehanteringsmotorn. Fält som är nycklar i tabellerna nedan är markerade med understrykning.

ERRAND

Huvudtabellen i hela systemet som innehåller samtliga ärenden.

Attribut	**Beskrivning**
errandId	Ärendets id. Unikt för varje ärende.
errandTypeConfId	Anger vilken ärendetyp ärendet tillhör.
status	Anger vilken status ärendet har. Följande status finns: odefinierat, icke aktivt, aktivt, arkiverat.
priority	Ärendets prioritet.
creatorUserId	Användaren som skapat ärendet.
deleted	Anger om ärendet är borttaget.
deadlineDateTime	Deadline för ärendet.
useDefaultPremission	Anger om standardrättigheter används för ärendet.
parentErrandId	Om det är ett underärende pekar detta fältet på förälderärendets id.
parentStateConfId	Om det är ett underärende pekar detta fältet på i vilket tillstånd i förälderärendet som underärendet ligger i.
workflowConfId	Anger vilket flöde som ärendet använder sig av.

ACTIVE_STATE

Innehåller alla tillstånd, både aktiva och inaktiva, som ett ärende har varit eller är i.

Attribut	**Beskrivning**
activeStateConfiId	Nyckel.
errandId	Foreign key: ERRAND.errandId
stateConfId	Pekar på ett tillstånd.

active	Anger som tillståndet är aktivt.
createdDateTime	Datum när posten skapades.
arrivalsToMerge	Används när flera tillstånd går samman till ett och samma tillstånd. Anger hur många som tillstånd som är kvar innan detta tillståndet kan bli aktivt.

TRACK_ACTIVE_STATE

Innehåller från vilka tillstånd som ett tillstånd kommer ifrån.

Attribut	**Beskrivning**
trackActiveStateID	Nyckel
activeStateId	Foreign key: ACTIVE_STATE. activeStateConfiId
oldActiveStateId	Anger föregående tillstånd.

TASK

Innehåller arbetsuppgifter och manuella beslut.

Attribut	**Beskrivning**
taskId	Nyckel.
errandId	Foreign key: ERRAND.errandId
stateConfId	Anger i vilket tillstånd som arbetsuppgiften/beslutet ligger i.
status	Status för arbetsuppgiften/beslutet. Följande status finns: icke aktiv, aktiv och utförd.
requiresAll	Anger om alla som är tilldelade arbetsuppgiften/beslutet måste genomföra denna för att markeras som slutförd. Om en grupp är tilldelad räcker det med en användare som tillhör gruppen utför arbetsuppgiften/beslutet.
isDecision	Anger om det är en arbetsuppgift eller ett beslut.
deadLineDateTime	Datum som anger arbetsuppgiftens/beslutets deadline.
deleted	Anger som arbetsuppgiften/beslutet är borttaget.

TASK_ASSIGNMENT

Tabellen innehåller information om vilka roller/användare som är tilldelande arbetsuppgifter/beslut och om dessa är utförda eller inte. Observera att på varje rad ska endast ett av attributen roleId eller userId vara definierat, det andra ska vara odefinierat (null).

Attribut	**Beskrivning**
taskAssignmentId	Nyckel.
taskId	Foreign key: TASK.taskId
roleId	Id för roll.
userId	Id för användare.
performed	Anger om användaren/rollen har utfört arbetsuppgiften/beslutet. Icke utförd = 0 Utförd = 1

METADATA

Tabell för metadata. Observera att endast ett av attributen dataFloat, dataInt, dataDatetime och dataString ska innehålla ett värde beroende av vilken typ som metadatat är av, övriga attribut ska vara odefinierade (null).

Attribut	**Beskrivning**
metaDataId	Nyckel
errandId	Foreign key: ERRAND.errandId
metadataConfId	Anger metadatatyp.
oldMetaDataId	Pekar bakåt på föregående rad av samma ärende och metadatatyp.
active	Anger om metadataraden är aktiv. Om den inte är aktiv betyder det att det finns ett nyare värde för metadatatypen för samma ärende.
dataFloat	Data av typen float.
dataInt	Data av typen int.
dataDatetime	Data av typen datetime.
dataString	Data av typen string.
languageId	Språk id som kan användas om metadatat är av typen string och om den ska vara språkberoende.

userId	Användaren som uppdaterade metadataraden.
logDatetime	Datum och tid när raden skapades.

ADHOC_STATE

Tabell som används för adhoc-flöden. Tabellen innehåller alla tillstånd (användare) som ärendet passerat samt vilka tillstånd som ärendet befinner sig i för tillfället.

Attribut	**Beskrivning**
userId	Användarens id.
errandId	Foreign key: ERRAND.errandId
active	Anger om tillståndet (användaren) är aktivt.
activeSinceDateTime	Datum som anger när tillståndet blev aktivt.

ADHOC_SEND

Innehåller information om hur ärendet har skickats mellan olika användare.

Attribut	**Beskrivning**
fromUserId	Användaren som ärendet kommer ifrån.
toUserId	Användaren som ärendet skickas till.
errandId	Foreign key: ERRAND.errandId