Självständigt arbete på grundnivå

Independent degree project - first cycle

Datateknik
Computer Engineering

Hypermediaorienterade verksamhetsprocesser

Björn Robèrt

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Institutionen för Informationsteknologi och medier

Examinator: Patrik Österberg, patrik.osterberg@miun.se
Handledare: Per Ekeroot, per.ekeroot@miun.se
Författare: Björn Robèrt, bjho0801@student.miun.se
Utbildningsprogram: Programvaruteknik, 180 hp
Huvudområde: Datateknik
Termin, år: VT, 2014

Sammanfattning

Webbens framgång har lett till tendens att stora IT-system byggda enligt en centralistisk arkitektur är på väg att ersättas av samverkande distribuerade system. De processororienterade, verksamhetsstödjande systemen utgör inget undandag. I enlighet med denna tendens så finns det ett behov för processtödjande verksamhetssystem att synkronisera och utbyta resultat med med andra verksamhetssystem. Målet med denna undersökning har varit att utreda hur kommunikationen mellan distribuerade verksamhetsprocesser byggda enligt BPMN 2.0-standarden kan implementeras. Ansatsen har varit att en tjänsteorienterad arkitektur baserad på principerna för REST skulle vara en framkomlig väg för att lösa detta problem. Teoretiska studier av tjänsteorienterad arkitektur, BPMN 2.0 samt REST har lett till en minimal implementation baserad på en ”proof-of-concept”-arkitektur i tjänsteplattformen Motrice från Motrice AB. Det konkreta lösningsförslaget utvärderas enligt Richardsons mognadsmodell, där mognadsnivån för lösningsförslaget avgörs i enlighet med denna modell. Slutsatsen är kommunikation mellan distribuerade verksamhetsprocesser bygger på meddelandeflöden, där dessa meddelandeflöden är helt förenliga med principerna för REST.

Nyckelord: BPMN, SOA, REST, hypermedia, distribuerade system.

Abstract

The success of the web has led to a tendency by which monolithic systems constructed according to a centralistic architecture,are replaced by distributed cooperating systems. Process and business oriented systems are no exception in this regard. This tendency has led to increased demand for process oriented systems to be able to synchronize and exhange results with other business systems. The purpose of this study has been to investigate how communication between distributed business processes built according to the BPMN 2.0 standard can be implemented. The hypothesis was that a service oriented architecture based on the principles of REST would be a viable approach to solve the problem. The theoretical study of service oriented architectures, BPMN and REST have resulted in a minimal implementation based on a proof-of-concept architecture. The concrete solution is then evaluated using Richardsons maturity model, and the maturity level of the solution is assessed according to this model. The conclusions are that communication between distributed BPMN 2.0 processes are based on message flows, where the concrete implementation of message flows are fully compatible with the principles of REST.

Keywords: BPMN, SOA, REST, hypermedia, distributed systems.

Innehållsförteckning

Sammanfattning ..... **iii**

Abstract ..... **iv**

Terminologi ..... **vii**

1 Inledning **1**
1.1 Bakgrund och problemmotivering 1
1.2 Övergripande syfte 2
1.3 Detaljerad problemformulering 2
1.4 Avgränsningar 2
1.5 Översikt 2
1.6 Författarens bidrag 3

2 Hypermediaorienterade verksamhetsprocesser **4**
2.1 Verksamhetsprocesser 4
2.1.1 BPMN 2.0 och exekverbara processer 4
2.2 Tjänsteorientering 6
2.3 Tjänsteorienterade verksamhetsprocesser 7
2.4 Tjänsteorientering och distribuerade system 7
2.4.1 Webbtjänster och webbaserade API:er 8
2.4.2 Representational State Transfer 8
2.5 Distribuerade system och hypermedia 10
2.5.1 Resurser och representationer 10
2.5.2 Identifiering av resurser 11
2.5.3 Självbeskrivande meddelanden 11
2.5.3.1 HTTP-metoder 12
2.5.3.2 Metadata och media typer 12
2.5.4 Hypermedia som motor för applikationens tillstånd 13
2.6 Hypermedia och verksamhetsprocesser 14

3 Metod **15**
3.1 Arbetsmetodik 15
3.2 Utvärderingskriterier 15
3.2.1 Richardsons mognadsmodell 15
3.2.2 Verktyg 16

4 Lösningsförslag **18**
4.1 Kravspecifikation 18
4.1.1 Övergripande krav 18
4.1.2 Processmodell 18
4.2 Komponenter 19
4.3 Implementation 20
4.3.1 Övergripande arkitektur 22
4.3.2 HTTP lager och resurser 22
4.3.2.1 Resursen ProcessService 23
4.3.3 Representationer 24

4.3.3.1 Media typen application/vnd.motrice+xml....24
4.3.3.2 JAXB och representationer....25
4.3.3.3 Link....25
4.3.3.4 Representation....26
4.3.3.5 ProcessRepresentation....26
4.3.4 Domänobjekt....26
4.3.4.1 Avsändaraktivitet....26

5 Resultat....27
5.1 Utfall av test i testmiljö....27

6 Analys....29
6.1 Efterlevnad mot Richardsons mognadsmodell....29
6.2 Efterlevnad mot BPMN 2.0 och processmodell....29

7 Diskussion....31
7.1 Resurser, kommunikation och REST....31
7.2 Begränsningar och förbättringsmöjligheter....31
7.3 Vidareutveckling och framtida forskning....32
7.4 Samhälleliga effekter....33

Källförteckning....34

Bilaga A: Dokumentation av egenutvecklad programkod....37
ProcessService....37

Bilaga B: Testmiljö....39
Hård- och mjukvarukrav....39
Installation av Motrice....39
Administration och installation av processdefinitioner....40
Test av lösningsförslag....42

Terminologi

Nedan presenteras de termer och begrepp som förekommer i rapporten.

Akronymer/Förkortningar

API	Application Programming Interface. Exponering av en programmoduls funktioner i form av en uppsättning programfunktioner eller procedurer. Genom att kombinera API:er och program kan mer komplexa tjänster och domänorienterade applikationer utvecklas modulärt
BPM	Business Process Management. En disciplin för att analysera, utveckla och förvalta verksamhetsprocesser.
BPMN	Business Process Management and Notation. En industristandard för att beskriva exekverbara verksamhetsprocesser.
HATEOAS	Hypermedia as the engine of application state. En princip för att
HTTP	Hypertext Transfer Protocol. Protocol för att skapa länkade dokument (webbsidor). Är i dag det protokoll som ligger till grund för hela Internet.
REST	Representational state transfer. Principer för distribuerade system som baserar sig på HTTP-protokollet och hypermedia.
SOA	Service-oriented architecture / Tjänsteorienterad arkitektur. Principer för att exponera komponenter som väldefinierade och återanvändbara tjänster.
SOAP	Simple Object Access Protocol. En standard för att implementera webbtjänster.
URL	Uniform Resource Locator. En sträng som agerar referens till en given resurs, t ex en webbsida, på Internet.
WAR	Web Application Archive. Ett filformat som används för att paketera och distribuera Java-baserade webbapplikationer.

1 Inledning

I alla moderna organisationer är de verksamhetsstödjande systemen (affärssystemen) en kritisk faktor för att nå framgång och leverera nytta till kunder och andra avnämnare. Som användare av affärssystem förväntar vi oss att de kan interagera sömlöst med oss och varandra för att lösa de problem vi ställs inför. Denna förväntning leder till att enskilda digitala redskap tenderar mot att innehålla allt mer komplex logik för specifika ändamål. För att hantera den ökande komplexiteten är det en trend att dela upp stora, monolitiska system i system som är mindre, och mer specialiserade. För att kunna realisera detta krävs att specialiserade eller lokala verksamhetssystem kan kommunicera och synkronisera med andra system för att kunna koordinera sig och utväxla resultat som led i att bygga mer komplexa verksamhetsprocesser.

1.1 Bakgrund och problemmotivering

Tjänsteplattformen Motrice från Motrice AB är en plattform för att bygga processdrivna webbapplikationer som är anpassningsbara till olika verksamheters områden. Plattformen bygger på ett antal komponenter för att definiera verksamhetsprocesser, skapa användargränssnitt och kombinera dessa i en applikation som exekverar på plattformen. Specifikt består Motrice av processmotorn Activiti [1], en formulärbyggare i form av Orbeon Forms [2] samt en innehållshanterare i form av Hippo [3]. Komponenterna används sig av PostgreSQL [4] för persistens, Spring [5] som persistenslager och Apache Tomcat [6] som applikationskontainer. Relationen mellan verksamhetsprocesser och användargränssnitt är medvetet löst kopplad i syfte att förenkla ändringar i någon del - användargränssnittet kan ändras med minimal påverka på verksamhetsprocessen och vice versa. Den lösa kopplingen mellan användargränssnitt och verksamhetsprocesser lägger även grunden för att skapa en tjänsterorienterad plattform där hela eller delar av processer exponerar sig som tjänster. En sådan tjänsteorientering skulle göra det möjligt för andra verksamhetssystem att maskinellt interagera med applikationer skapade på plattformen för att utföra arbetsmoment eller själva utföra arbetsmoment som är definierade i applikationen och kommunicera tillbaka resultatet. Med en sådan tjänsteorientering kan Motrice agera som en delkomponent i ett distribuerat system eller ansvara för koordineringen av olika distribuerade system.

För att möjliggöra en sådan interaktion mellan applikationer byggda på Motrice och andra verksamhetssystem behövs ett tjänstelager utvecklas. Tjänstelagret bör bibehålla de kvaliteter som Motrice har med löst kopplade och väl avgränsade komponenter samt använda sig av principer för tjänsteorientering och distribuerade system.

1.2 Övergripande syfte

Syftet med examensarbetet är undersöka om det är möjligt att applicera principerna för webbaserade distribuerade system på exekverara processer definierade enligt standarden *Business Process Management and Notation 2.0* (BPMN 2.0) för att skapa ett tjänsteorienterat lager för tjänsteplattformen Motrice genom vilket olika och distribuerade processmotorer kan kommunicera och synkronisera med varandra.

En ansats har varit att visa på att arkitekturprinciperna för *representational state transfer* (REST) är en framkomlig väg för att nå det ovan beskrivna syftet, och att de mekanismer som beskrivs för kommunikation mellan processer i BPMN 2.0-standarden därmed kan generaliseras till att även gälla för distribuerade processer.

1.3 Detaljerad problemformulering

Undersökningen har som mål att ta fram ett lösningsförslag för följande problem:

- Är det möjligt att implementera kommunikation mellan verksamhetsprocesser i en tjänsteorienterad arkitektur enligt principerna för REST?

Närmare bestämt:

- Kan en processmotor exponeras i enlighet med REST?
- Kan man kommunikation mellan processmotorer ske baserat på REST?
- Kan man synkronisera processmotorer genom meddelandeflöden baserade på REST?

1.4 Avgränsningar

Denna undersökning begränsar sig till att undersöka hur kommunikationen mellan två affärssystem kan lösas enligt en *klient-server modell* där båda parter har *a priori* kunskap om varandras existens. Vidare fokuserar undersökningen på kommunikation mellan två tänkta affärssystem som är implementerade enlig BPMN 2.0 standarden. Undersökningen avgränsar sig även från felhantering vid kommunikation mellan distribuerade system.

1.5 Översikt

Kapitel 2 beskriver det teoretiska bakgrundsmaterialet som ligger till grund för själva litteraturstudien. Materialet diskuterar vad verksamhetsprocesser är, hur de relaterar till en tjänsteorienterad arkitektur samt vad beröringspunkterna mellan nätverksbaserade system och tjänsteorientering är.

Kapitel 3 beskriver metoden för arbetet som hela studien grundar sig på och den modell som används för att utvärdera lösningsförslaget.

Kapitel 4 beskriver den lösning som har konstruerats i närmare detalj inklusive principer och mönster för själva konstruktionen.

Kapitel 5 presenterar resultaten av lösningsförslaget utifrån tester i en testmiljö.

Kapitel 6 för ett resonemang kring hur väl lösningsförslaget lever upp till principerna för REST.

Kapitel 7 är en slutdiskussion kring hela studien som diskuterar styrkor, svagheter samt intressanta områden att titta vidare på.

Bilaga A innehåller en utförligare beskrivning av valda delar av kodbasen.

Bilaga B är en guide för att konfigurera och installera Motrice och lösningsförslaget i en testmiljö.

1.6 Författarens bidrag

Materialet som ingår i denna stuide har sammanställts och utvecklats av författaren. Roland Hedayat och Björn Molin på Motrice AB har bistått med frågeställningar kring principer, arkitektur och implementation.

2 Hypermediaorienterade verksamhetsprocesser

Detta kapitel redogör för hur verksamhetsprocesser är en central komponent i organisationers möjligheter att generera nyttor, hur tjänsteorientering är en viktig del för effektiva och realiserbara verksamhetsprocesser samt hur löst kopplade och distribuerade tjänster understödjer en sådan tjänsteorientering.

2.1 Verksamhetsprocesser

För alla organisationer gäller det att målet för organisationen som sådan är att realisera någon form av nyttor (vinst, trogna kunder, nöjda medborgare etc). Organisationens mål, struktur och verksamhetsprocesser är det medel med vilken den på ett resurseffektivt sätt försöker uppnå dessa nyttor [7]. Teoribildningen, analysen och utvecklingen av verksamhetsprocesser har historiskt skett under varierade paraplybegrepp som *total quality management*, *lean* med flera [8]. Ett senare inslag i teoribildningen kring verksamhetsprocesser är *business process management* (BPM)som kan summeras som

> [a] management discipline focused on using business processes as a significant contributor to achieveing an organization's objectives through the improvement, ongoing performance management and governance of essential business processes [8].

2.1.1 BPMN 2.0 och exekverbara processer

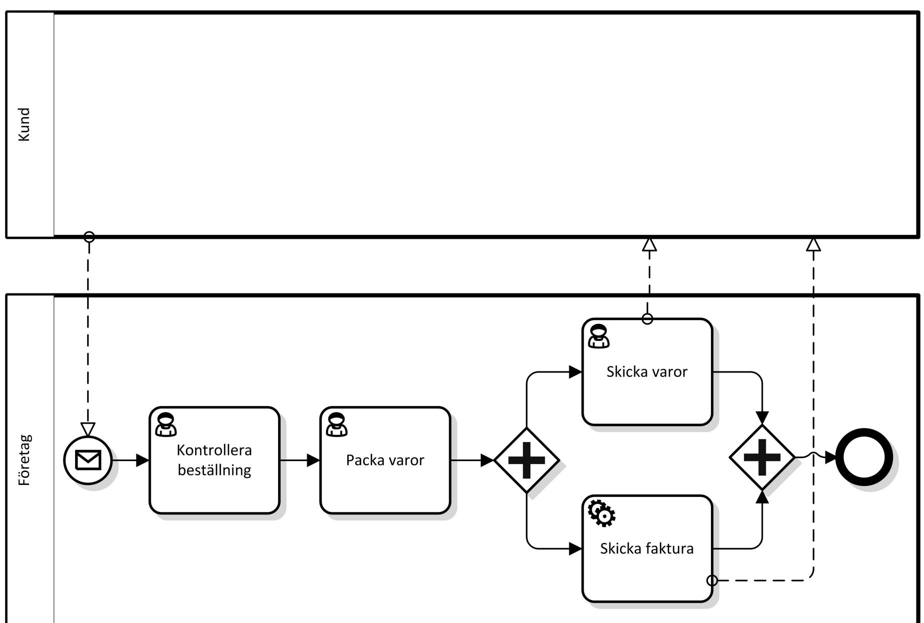
Historiskt har det funnits ett antal olika standarder för att beskriva verksamhetsprocesser, men sedan 2011 har *Business Process Management and Notation 2.0* (BPMN) [10] seglat upp som ett väletablerat och utbrett alternativ [11]. BPMN är ett

> expressive language, able to describe nuances of process behaviour compactly... At the same time, the meaning is precise enough to describe the technical details that control *process execution in an automation engine* [min kursivering]. Thus BPMN bridges the worlds of business and IT, a common process language that can be shared between them [11].

En viktig aspekt av BPMN 2.0 är att specifikationen håller en formaliserad XML-baserad syntax för att skapa *exekverbara* processer. Med exekverbar menas att den definierade processen kan importeras och exekvera i någon applikation, eller processmotor, som stödjer standarden. Uttryckt annorlunda kan en exekverbar BPMN 2.0 process liknas vid programkod som stöder ett givet arbetsflöde vilken dynamiskt kan instansieras. Denna förmåga ger verksamheter möjligheten att implementera processen i verksamhetssystem som stöttar verksamheten i att säkerställa att den överenskomna processen faktiskt följs. Den gör det även möjligt att analysera och förfina verksamhetens processer genom att processerna skapar konkret mätbar data under och efter exekvering.

I BPMN 2.0 är definitionen av vad en process är strikt: en process börjar och slutar och består av ett antal aktiviteter som utförs av någon aktör (människa eller maskin) [11]. Till det finns ett antal artefakter som specificerar hur proces-

sen är tänkt att flöda beroende på utfallet från aktiviteter eller händelser i processen. De mest grundläggande artefakterna i BPMN 2.0 är *händelser, aktiviteter* och *flöden*. Figur 1 nedan visar hur en mycket enkel beställningsprocess kan definieras enligt BPMN 2.0. Standarden innehåller betydligt fler konstruktioner än vad exemplet innehåller och den intresserade läsaren hänvisas till den för mer information.



Figur 1: Exempel på en BPMN process med pooler, star- och sluthändelser, aktiviteter och meddelandeflöden.

I figuren ovan finns två *pooler*, angivna som *kund* respektive *företag*, som illustrerar de två involverade entiteterna. En pool anger deltagarna i en process, vanligtvis organisationer eller organisatoriska enhet vars interna process är helt särskilt andra organisatoriska enheter. Kommunikationen mellan pooler utgörs av *meddelandeflöden*, illustrerade med de streckade linjerna, som innehåller specifika *meddelanden* vars innehåll resulterar i någon förändring hos den mottagande poolen. I figuren så skickar kunden en beställning till företaget som i sin tur skickar tillbaka de varor som beställts samt en faktura. Den beställning som skickas till företaget aktiverar företagets *starthändelse* (angiven som en rund ring omslutande ett kuvert) genom att ett meddelande. Beställningsprocessen hos företaget består av ett antal *aktiviteter* (rundade rektanglar med en symbol uppe till vänster), två *portar* eller *grindar* (romber med en plustecken) som är sammankopplade med *sekvensflöden*. De aktiviteter som har en människoliknande symbol uppe till vänster är *användaraktiviteter* och indikerar att uppgiften genomförs av en fysisk person. Den aktivitet som har kugghjul uppe till vänster är en *serviceaktivitet* och indikerar att aktiviteten genomförs av en maskin. Vanligtvis är det senare någon form av automatiserad uppgift som att kontrollera uppgifter mot en databas, anropa en extern webbtjänst eller liknande. Grindarna i processen ovan är av typen *parallella grindar* vilket innebär att båda aktiviteterna som följer efter den första grinden måste genomföras för att processen ska kunna fortlöpa och avslutas. Till sist avslutas processen med en *sluthändelse* (angiven som en cirkel med ett tjockt streck).

2.2 Tjänsteorientering

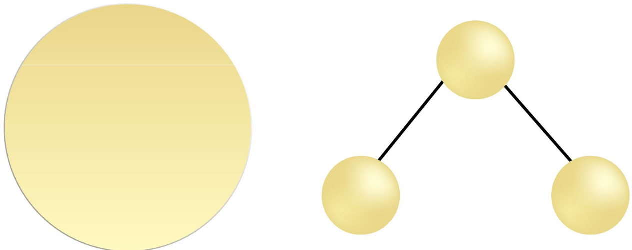
Över tid så har *informations- och kommunikationsteknologi* (IKT) blivit viktiga (för att inte säga oumbärliga) komponenter för att realisera nyttan med organisationens verksamhetsprocesser. Trots det är det vanligt att just IKT fastnar i teknologiska och organisatoriska stuprör som gör organisationen trögrörlig och därmed försvårar dess förmåga att förbättra och utveckla sina verksamhetsprocesser. Insikten om att IKT kan vara ett hinder för en organisations realisering av nytta gav under 2000-talets början upphov till begreppet *tjänsteorienterad arkitektur (SOA)* [12]. Målet med SOA var (och är) att beskriva och upprätthålla arkitektoniska principer för IKT som den följsam mot organisationens affärsmässiga mål – vilket självklart även inbegriper organisationens verksamhetsprocesser. SOA är ett stort område och kommer inte att redogöras för i helhet här varför följande stycken kommer att fokusera på det väsentliga för den här uppsatsen: tjänster.

För att uppnå en följsamhet mot organisationens strategier och mål är en viktig princip inom SOA att tala om *tjänster* och *förmågor* snarare än teknologier och specifika informationssystem. Inom SOA definieras en tjänst som ett

> independet software program with distinct design characteristics that support the attainment of the strategic goals... Each service is assigned its own distinct funtional context and is comprised of a set of capabilities related to this context. Those capabilities suitable for invocation by external consumer programs are commonly expressed via a published service contract [12].

Tjänster kan grovt delas in i *enkla* och *sammansatta* tjänster. *Enkla* tjänster är tydligt definierade, återanvändbara och oberoende av andra tjänster. *Sammansatta* tjänster är i sin enklaste form ett koordinerat aggregat av enkla tjänster, men kan även innehålla andra sammansatta tjänster [12][13]. Vanligtvis så kan man betrakta ett helt informationssystem som en sammansatt tjänst i det att informationssystemet syftar till att understödja eller automatisera en given verksamhetsprocess. En tjänsts *förmågor* är, enkelt uttryckt, vad tjänsten kan göra - ofta uttryckt i form av ett explicit kontrakt.

Ett konkret exempel på en enkel tjänst är en applikation som hanterar utskrifter av dokument. Förmågorna som denna tjänst exponerar kan vara att skriva ut på papper, som PDF eller som en bild. Figur 2 nedan visar symbolerna för enkla respektive sammansatta tjänster inom SOA.

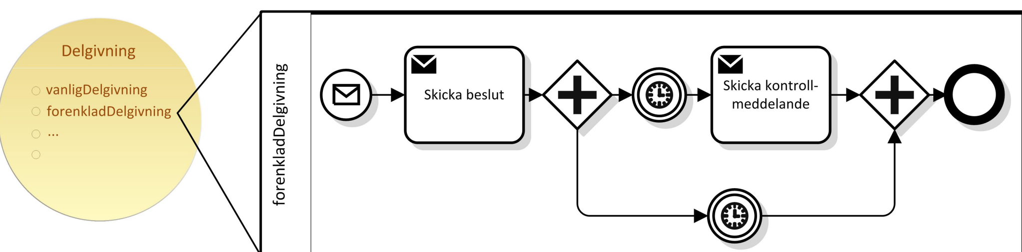


Figur 2: Enkel respektive sammansatt tjänst inom SOA.

2.3 Tjänsteorienterade verksamhetsprocesser

Inom SOA så är det vanligt att tala om tjänster som ingående komponenter i olika verksamhetsprocesser. Ur det perspektivet är verksamhetsprocessen något som *koordinerar* eller *orkestrerar* olika tjänster för att uppnå ett eller flera givna mål. Men, och det är en väsentlig poäng, en tjänst kan *i sig* vara en verksamhetsprocess [12]. I praktiken innebär det att en tjänst, enkel eller sammansatt, exponerar ett kontrakt där den bakomliggande implementationen är definierad enligt BPMN 2.0.

Ett konkret exempel på en sammansatt uppgiftstjänst där de bakomliggande implementationerna kan definieras enligt BPMN 2.0 är myndigheters delgivning av beslut[1]. Figur 3 nedan visar både definitionen av tjänsten *Delgivning* och implementationen av förmågan *förenklad delgivning* enligt BPMN 2.0.



Figur 3: Tjänsten Delgivning med förmågan forenkladDelgivning implementerad enligt BPMN 2.0

Man kan anta att tjänsten anropas med någon form av dokument som sedan flödar genom processen tills dess att den avslutas. De olika aktiviteterna, som exempelvis ”skicka beslut”, kan använda sig av andra enkla eller sammansatta tjänster för att utföra den uppgift som krävs.

2.4 Tjänsteorientering och distribuerade system

I majoriteten av alla större organisationer så är det inte vanligt för alla informationssystem att existera på en och samma fysiska server av resursmässiga skäl. Det i dag möjligt att fysiskt distribuera system över ett antal servrar och ändå betrakta dem som logiskt enhetliga [14]. Det finns ett antal olika paradigm för hur denna distribution av informationssystem ser ut i praktiken, men en gemensam definition för alla typer av distribuerade system kan uttryckas som följer:

> A distributed system is a collection of independent computers that appear to its users as a single coherent system [14].

Realiseringen av ett distribuerat system kan ske genom olika modeller eller paradigm och det kommer inte att ske någon fördjupning inom området här. Det som är av intresse för denna studie är att definitionen ovan är giltig även för tjänster som ingår i en tjänsteorienterad arkitektur. Faktum är att en av de grundläggande principerna är att tjänster, enkla eller sammansatta, kan vara fy-

1 Det finns flera olika former av delgivning och dessa kommer inte att redogöras för i detalj här. Den intresserade läsaren hittar mer på www.domstol.se/Delgivning/

siskt distribuerade över flera olika nätverk (interna såväl som externa) utan att tjänstekonsumenter är medvetna om det [15][16]. Detta möjliggör skapandet av virtuella organisationer som spänner över stora geografiska avstånd men som logiskt hör samman. En annan fördel är även att tjänsterna som sådana kan tillhandahållas av flera olika leverantörer utan att bakomliggande informationssystem lever rent fysiskt i den tjänstekonsumerande organisationen.

2.4.1 Webbtjänster och webbaserade API:er

Inom SOA litteraturen är den absolut mest vanligt förekommande metoden för att distribuera tjänster *webbtjänster* eller webbaserade programmeringsgränssnitt (*application programming interface,* API). Webbtjänster är, ungefär som det låter, tjänster där interaktionen sker genom webben. Mer specifikt är följande tre kriterier användbara i en definition av webbtjänster [16]:

1. Webbtjänster använder sig av protokoll ämnade för Internet som transportmekanism, vanligen HTTP.
2. Webbtjänster exponerar sig som en eller flera *representationer*, vanligtvis baserade på XML eller JSON.
3. Webbtjänster är adresserbara, vanligtvis genom en URL.

Ett av skälen till att webbtjänster har vuxit fram som en *de facto* standard för distribuerade tjänster är den relativa enkelhet med vilken de exponeras och konsumeras. Enkelheten består, framförallt, i att webbtjänster inte ställer några specifika krav på vilket programmeringsspråk som används för att producera eller konsumera tjänsten. Detta skiljer webbtjänster från andra lösningar, som exempelvis Java RMI, där både producenter och konsumenter måste vara implementerade i samma språk (Java). Vidare så ställer webbtjänster låga krav på geografiskt spridda organisationer genom att all kommunikation sker över HTTP, vilket i sin tur ställer lägre krav på skalskydd som brandväggar och liknande. Ytterligare en fördel är att webbtjänster inte kräver att den egna organisationen äger eller helt kan hantera de komponenter som använder sig av tjänsten. Webbtjänster kan exponeras av olika leverantörer vars informationssystem befinner sig i eller utanför den konsumerande organisationens gränser och ägandeskap, utan att det förhindrar tjänstens användning [16].

I dag finns det i stort två dominerande modeller för hur webbtjänster implementeras: *Simple Object Access Protocol (SOAP)* och *Representational State Transfer (REST)*. Under 2000-talets början var SOAP den dominerande standarden för webbtjänster, medans REST i dag är den absolut mest vanligt förekommande [17].

2.4.2 Representational State Transfer

Det som utmärker REST som arkitektonisk princip för distribuerade tjänster är att den fullt ut nyttjar HTTP protokollet och hypermedia för att exponera och konsumera tjänster. Även om principerna för REST har existerat sedan HTTP protokollet standardiserades och hypermedia började användas för att skapa länkade dokument, så var det inte förrän Roy Fielding beskrev och satte princi-

perna i ett större sammanhang i sin klassiska avhandling *Architectural Styles and the Design of Network-based Software Architecture* som spridningen verkligen tog fart [20]. REST utgår från sex arkitektoniska avgränsningar som beskriver hur relationen mellan komponenter i ett distribuerat system ser ut [20] [21].

1. *Klient-server*. Denna avgränsning tvingar fram en uppdelning i problemområden mellan klienten och servern vilket låter båda utvecklas oberoende av varandra. I relation till SOA så innebär det att en tjänst tillhandahåller en eller flera förmågor och lyssnar på anrop till dessa. Konsumenten (klienten) nyttjar en tjänst genom att anropa den med ett meddelande och tjänsten skickar ett svarsmeddelande antingen efter att förfrågan har avslåts eller när aktiviteten är genomförd.
2. *Tillståndslöshet* (eng. s*tateless*). Varje förfrågan från klienten måste innehålla tillräckligt mycket information för att servern ska kunna förstå och besvara frågan utan att *klientens* tillstånd hålls på servern. Det innebär att varje förfrågan som tjänstekonsumenten skickar kan betraktas som helt oberoende från tidigare förfrågningar. Detta sker i praktiken genom *självbeskrivande meddelanden* som behandlas i avsnitt 2.5.3 nedan.
3. *Mellanlagring* (eng. *cache*). Varje svar som servern ger till en fråga från klienten måste markeras som lagringsbart (eng. *cacheable*) eller ickelagringsbart (eng. *Non-cacheable*). Fördelen med markeringen är att en klient vet om det är möjligt att återanvända ett tidigare svar vilket i sin tur leder till färre antal frågor och en ökad effektivitet, skalbarhet och användarupplevd prestanda.
4. *Enhetligt gränssnitt*. Ett utmärkande drag för REST är att gränssnitten mot komponenter (dvs de tjänster som ett informationssystem exponerar) är uniforma i sin utformning. Det medför att interaktionen med tjänster förenklas och att tjänstens implementation blir löst kopplad gentemot tjänstkontraktet. Denna avgräsning behandlas i avsnitt 2.5 nedan.
5. *Skiktat system* (eng. *layered system*). Ett skiktat system innebär att en komponent på ett givet skikt inte kan se, eller behöver känna till, vad som är på andra skikt. Detta bidrar till att minska komplexiteten i systemet som helhet och ökar den lösa kopplingen mellan olika skikt.
6. *Kod-på-begäran* (eng. *Code-on-demand*). Syftet med kod-på-begäran är att utvidga klientens funktionalitet i form av applets eller skript som kan exekveras på begäran. Vinsten är därmed att det minskar antalet features som klienten måste ha på förhand.

Av dessa är det uniforma gränssnittet den avgränsning som primärt särskiljer REST från andra typer av arkitekturer och behandlas i nästa avsnitt.

2.5 Distribuerade system och hypermedia

Distribuerade system som bygger på principerna för REST måste använda sig av ett enhetligt gränssnitt vid kommunikation mellan komponenter inom samma skikt. Styrkan med ett enhetligt gränssnitt är att det bygger på principen om generaliserbarhet, det vill säga att de är likartade till sin natur. Ur ett tjänsteorienterat perspektiv innebär detta att det finns en fundamental grund för alla tjänstekontrakt, vilket gör både kontrakten och konsumtionen av tjänsterna blir både enklare och mer förutsägbara [21]. Ett enhetligt gränssnitt måste enligt Fielding följa fyra principer: *förändring av resurser genom representationer*, *identifiering av resurser*, *självbeskrivande meddelanden* samt *hypermedia som "motor" för applikationens tillstånd* [20]. Var och en av dessa principer redogörs för i mer detalj nedan.

2.5.1 Resurser och representationer

Ett grundläggande koncept i en REST-baserad arkitektur är *resurser* och deras *representationer* [17][20][21][22]. En resurs är, enligt Fielding

> [a]ny information that can be named *can be be a resource* [min kursivering]: a document or image, a temporal service (e.g. "today's weather in Los Angeles, a collection of other resources, a non-virtual object (e.g. a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entitiy that corresponds to the mapping at any particular point in time[20].

Enligt definitionen ovan består en resurs av två komponenter: ett *namn* eller *beteckning* samt de *entiteter* som namnet (beteckningen) är kopplad till. Ett konkret exempel är examensarbetet "Hypermediaorienterade verksamhetsprocesser" som är det *namn* som beskriver den informationsmängd som arbetet innehåller. Det finns ingen väl definierad begräsning för vad som kan vara och inte vara resurser, men vanligt perspektiv är att se dem *subjekt*, det vill säga något som *är* [17]. Fieldings definition ger med sig att även att även verksamhetsprocesser kan betraktas som resurser. De webbtjänster som exponeras och möjliggör interaktionen med verksamhetsprocesserna måste då följa principerna för ett enhetligt gränssnitt.

För att möjliggöra interaktion med resurser så krävs det att tjänstekonsumenten kan nå kunskap om resursens nuvarande tillstånd samt förändra detta tillstånd i någon önskvärd riktning [20]. Detta sker genom interaktion med *representationer* av resursen och aldrig mot resursen i sig. En representation kan enklast liknas vid en vy av en resurs tillstånd vid ett givet tillfälle [22]. En sådan vy innehåller både data och metadata för att beskriva datamängen [20]. I praktiken är representationer exempelvis XHTML, XML, PDF, textfiler eller liknande. Som ett konkret exempel så finns informationsmängden i föreliggande examensarbete (resursen) representerat som både PDF och ODF (representationer). Det dataformat som en representation använder sig av kallas för representationens *media typ*. Media typen anger hur dataformatet och datamängden ska hanteras av tjänstekonsumenten och utgör därmed en väsentlig del av webbtjänsten. Avsnitt 2.5.3 fördjupar den diskussionen.

Möjlighet att representera resurser på olika vis kallas för *content negotiation* och bidrar till en lös koppling mellan konsumenter och producenter genom att en resurs kan exponeras olika beroende på konsumentens önskemål och förmågor [22]. Till exempel så kan uppgifter om kunder och deras faktueringsadresser som lagras i en databas skickas som en XHTML representation om tjänstekonsumenten är en webbläsare, som en XML-fil om konsumenten är ett annat informationssystem eller som ett textdokument om konsumenten är en äldre informationssystem. Det finns alltså ett en-till-många relation mellan resurser och dess representationer och en webbtjänst kan, i bästa fall, stödja flera olika varianter [20].

2.5.2 Identifiering av resurser

För att tjänstekonsumenter ska kunna interagera med resurser (genom representationer) så krävs det kunskap om var resursen existerar. En *resursidentifikator* representerar själva resursen som en tjänst exponerar. Det vanligaste sättet att beskriva resursidentifikatorer för webbaserade tjänster är genom *Uniform Resource Identifier (URI)* [21][22]. I dagligt tal använder vi begrepp som adress, webbadress eller *Uniform Resource Locator (URL)* när vi talar om URI:er.

En URI följer en gemensam, internationellt tillämpad standard som anger syntax och giltiga tecken. Syntaxen för en URI är alltid `{schema}:{auktoritet}{sökväg}?{fråga}#{fragment}`. För webbaserade tjänster är schemat vanligtvis HTTP, auktoriteten är den domän inom vilken tjänsten lever och sökvägen anger vilken resurs inom domänen som åsyftas. Ett exempel på hur en URI för en tjänst kan se ut är `http://exempel.com/delgivning/forenklad`. `HTTP` anger att URI:n ska tolkas enligt schemat för HTTP. Under detta schema så pekar `exempel.com` på den värddator som har *exempel.com* registrerat i en DNS. `delgivning/forenklad` anger sökvägen till den specifika resursen som hanterar förenklad delgivning av dokument. Precis som för representationer gäller en en-till-många relation mellan resurser och URI:er. Det innebär att `http://exempel.com/delgivning/forenklad` och `http://organisation.com/a/b` kan peka till exakt samma resurs.

2.5.3 Självbeskrivande meddelanden

Vid en given interaktion mellan producent och konsument så måste de meddelanden som skickas varar *självbeskrivande*. Som det låter så innehåller ett självbeskrivande meddelande all den information (data och semantik för data) som krävs för att processa frågan eller svaret; data såväl som metadata [20]. Självbeskrivande meddelanden är en konsekvens av att principerna för REST anger att all interaktion mellan en tjänsteproducent och tjänstekonsument måste vara tillståndslös [17][20][21][22]. Det innebär i praktiken att interaktionen inte kan förutsätta att det finns någon kontextuell förståelse som klienten och servern delar utan *meddelandet i sig måste bära all nödvändig information för den förändring som önskas*.

För webbtjänster innebär det i praktiken att en fråga (eller svar) innehåller vilken *HTTP-metod* som används, vilken *media typ* som frågan (svaret) innehåller

samt den *data* som håller själva meddelandets innehåll. Figur 4 nedan visar ett exempel på hur ett sådant meddelande kan se ut i form av ett HTTP anrop.

```
POST /customer   HTTP 1.1
Host: exempel.com
Content-Type: application/vnd.exempel+xml
Content-Length:...

<customer xmlns="http:://schema.exempel.com/customer>
  <name>Företaget</name>
  <location>
    …
  </location>
  …
</customer>
```

Figur 4: Exempel på självbeskrivande meddelande (HTTP anrop)

Figuren ovan illustrerar hur en klient kan skapa en ny kund genom att skicka en förfrågan mot http://exempel.com/customer. De fetade partierna visar de viktigaste delarna och förtydligas nedan.

2.5.3.1 *HTTP-metoder*

Vid interaktion med identifierade resurser följer HTTP-metoderna en vedertagen semantik som kort är följande [23]:

- GET hämtar den representation som är identifierad av en URI.
- POST indikerar att klienten vil skapa en ny underliggande resurs till den resurs som är identifierade i URI:n.
- PUT innebär att klienten önskar förändra en befintlig resurs med det innehåll som finns i anropet.
- DELETE innebär att klienten önskar ta bort den resurs som är identifierad i URI:n.

Sammantaget så kan ovanstående metoder användas för att skapa, läsa, uppdatera och radera/ta bort en resurs.

2.5.3.2 *Metadata och media typer*

Den metadata som finns angiven i ett meddelande införlivas i ett antal nyckel värde par i ”huvudet” under HTTP-metoden och tillhörande URI. HTTP protokollet definierar ett antal sådana fält, men två av de viktigare är `accept` (för anrop) och `content-type` (för anrop och svar) vilka anger den *media typ* som meddelandekroppen innehåller. Media typen anger hur servern (eller klienten) ska tolka och bearbeta datamängden som överförs och utgör därmed en viktig del av tjänstekontraktet. Utan en media typ finns det ingen explicit överrenskommelse över hur datamängden ska bearbetas vilket bryter mot principen om självbeskrivande meddelanden. I figuren ovan är det dataformat som skickas av typen `application/vnd.exempel+xml` vilket indikerar att media typen är applikationsspecifik (*application*), en organisationsspecifik standard (*vnd.exempel*) och att data representeras som XML (*+xml*).

2.5.4 Hypermedia som motor för applikationens tillstånd

Den sista, men helt avgörande, principen för REST-baserade webbtjänster är användandet av *hypermedia* [25][26], sammanlänkandet av resurser genom explicita *länkar*, för att ändra *applikationens* tillstånd (eng. *Hypermedia as the engine of application state, HATEOAS*). Denna princip är föremål för en stor mängd diskussioner och missförstånd som ofta faller tillbaka på begreppet *applikation* [17][21][22]. Av detta skälet så är det viktigt att klargöra att *applikationens tillstånd* syftar till *klientens* tillstånd och inte serverns (eller resursens) tillstånd [17][27].

Hypermedia driver tjänstekonsumentens (klientens) tillstånd genom att tjänsteproducenten (servern) kontinuerligt visa vilka handlingar som är möjliga att genomföra i varje enskilt ögonblick [17]. Det exempel som brukar anges för att illustrera principen för HATEOAS är en vanlig webbläsare som interagerar med någon webbapplikation. I en sådan interaktion kommer den (X)HTML kod som webbläsaren renderar när användaren besöker en resurs (webbsida) med överväldigande sannolikhet att innehålla avsnitt som påminner om Figur 5 nedan:

```
<a href=”http://exempel.com/erbjudanden” rel=”offers”>Erbjudanden!</a>
```

Figur 5: Exempel på hypermedia i XHTML

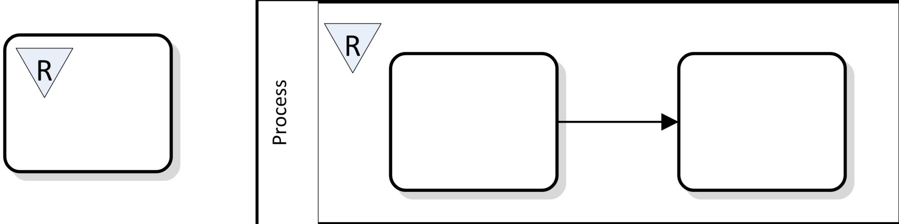
I XHTML innebär elementet *a* att den inneslutna texten ska renderas som en *länk*. När användaren (klienten) klickar på länken så kommer webbläsaren att ta användaren vidare till den resurs (webbsida) som är angiven med attributet *href*. Attributet *rel* är en semantisk markör som anger vilken relation den innevarande resursen (webbsidan) har till den länkande resursen. Elementet är därmed en *hypermediakontroll* eftersom den ”guidar” eller talar om var klienten kan gå vidare (*href*) samt vad klienten kan förvänta sig när den besöker resursen (*rel*). En webbapplikation kan innehålla mängder av liknande kontroller som guidar klienen genom hela verksamhetsprocesser (t ex en on-line butik). Skälet till att webbläsaren ”vet” att (X)HTML elementet *a* ska rendera en länk är att den media typ (*text/html*, eller *application/xhtml+xml*) som anges i svaret från servern är känd för klienten.

Ovanstående beskrivning är i grund och botten vad principen HATEOAS innebär. Det som inte är uppenbart vid första anblicken är att HATEOAS gör det möjligt för tjänstekonsumenter att interagera med en tjänst utan att på förhand veta hur den fullständiga interaktionen kommer att se ut [17][21][22]. Enligt Fielding är HATEOAS ett krav webbtjänster som vill följa principerna för REST:

> A REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and a set of standardized media types that are appropriate for the intended audience (i.e., expected to be undersood by and client that might use the API). From that point on, all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user's manipulation of those representations. The transitions may be determined (or limited) by the client's knowledge of media types and resource communication mechanisms, both of which may be improved on the fly... *Failure here implies that out-of-band information is driving interaction instead of hypertext...* [27]

2.6 Hypermedia och verksamhetsprocesser

Ovanstående avsnitt har visat hur *exekverbara verksamhetsprocesser*, definierade enligt standarden BPMN, kan vara använda sig av, eller vara byggstenar i, *tjänster* i en *tjänsteorienterad arkitektur*. En tjänsteorienterad arkitektur bygger på distribuerade system som exponerar sig själva som tjänster med ett *exponerat* och *väldefinierat* gränssnitt för interaktion. Principerna för REST kan användas som vägledning vid framtagandet av sådana gränssnitt för att skapa tjänster som är väl anpassade till webbens arkitektur. Cesare Pautasso vid Luganos Universitet i Schweiz har undersökt hur principerna för REST kan integreras med BPMN 2.0 för att möjliggöra komposition och orkestrering av processer exponerade som tjänster [28]. Pautasso tar sin utgångspunkt i att både processer och de aktiviteter som ingår i processer bör kunna definieras som resurser med vilka klienter kan interagera. Dessa resurser (processer och/eller aktiviter) kan sedan manipuleras genom det uniforma gränssnitt som REST förespråkar. För att möjliggöra detta förespråkar han en utvidgning av BPMN 2.0 standarden med en specifik notation som anger vilka processer och aktiviteter som ska vara exponerade resurser. Figur 6 nedan visar notationen.



Figur 6: Exempel på en BPMN aktivitet respektive process som REST resurser.

Varje process som är exponerad som en resurs kan få då en URI lik `/process/{process-id}` där {process-id} är den specifika processen som ska manipuleras. Motsvarande URI för en aktivitet kan vara `/process/{process-id}/{aktivitets-id}`. Enligt Pautasso kan sedan en klient interagera med de exponerade resurserna genom det uniforma gränssnitt som REST förespråkar.

Det Pautasso inte diskuterar närmare är att det i BPMN 2.0 standarden redan finns konstruktioner för att exponera processer och delar av processer gentemot en klient. Dessa är de *meddelandeflöden*, *meddelandeaktiviteter* samt *meddelandehändelser* som redogörs för under avsnitt Fel: Det gick inte att hitta referenskällan ovan. Hur detta kan se ut i praktiken återkommer vi till i kapitel 4, resultatet presenteras i kapitel 5 och kapitel 6 utvärderar resultatet.

3 Metod

Detta kapitel inleds med att klargöra den metodologiska ram inom vilken studien har genomförts för att gå vidare med att redogöra för hur studien har genomförts.

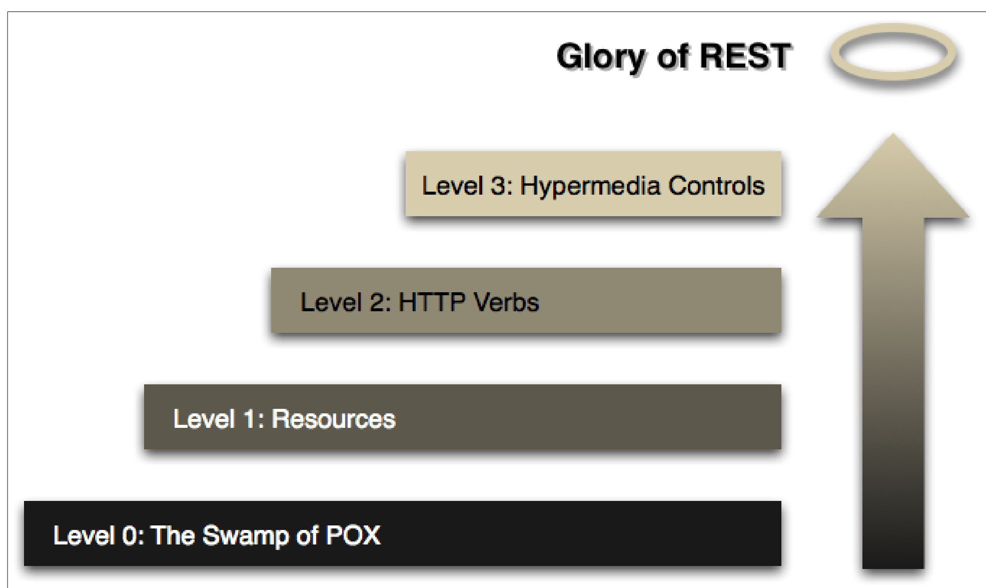
3.1 Arbetsmetodik

Lösningsförslaget har arbetats fram i ett agilt förfarande med en iterativ och inkrementell utvecklingsprocess. Gemensamt för alla agila metoder är att de syftar till verifiera att rätt behov tillgodoses genom att kontinuerligt involvera den part för vilken lösningen tas fram, bygga lösningen interrativt och inkrementellt samt visualisera arbetet som ska utföras [36]. Den metod som har legat till grund för detta arbete är *kanban*. Kanban, till skillnad från Scrum eller XP, beskriver inga specifika lösningar för utvecklingsprocessen ska gå till i termer av arbetsmetoder, tidsavgränsningar och roller [37]. Det gör metoden mycket anpassningsbar till den kontext inom vilket arbetet ska utföras. Som en jämförelse så anger Scrum att utvecklingsarbete ska innehålla rollerna produktägare, scrum-master och resterande delen av utvecklingsteamet [36]. För detta examensarbete har inga formaliserade roller använts, men i likhet med Scrum så har s k *sprintar* använts för själva lösningsförslaget. En sprint är en tidssatt period, i det här fallet en vecka, inom vilken ett antal delar av lösningsförslaget förväntas vara klara. Varje enskilt sprint levererade fungerande del av lösningsförslaget och/eller innevarande rapport. Den veckovisa avstämningen redogjorde för vad som är tänkt att genomföras under kommande vecka, samt vad som har genomförts under den gångna perioden. Arbetet har visualiserats med hjälp av det webbaserade verktyget Trello.

3.2 Utvärderingskriterier

3.2.1 Richardsons mognadsmodell

För att utvärdera hur väl lösningsförslaget följer principerna för REST har Leonard Richardsons mognadsmodell använts. Modellen delar in efterlevnaden av principerna för REST fyra nivåer där den lägsta nivån innebär ingen efterlevnad, medan den högsta nivån innebär full efterlevnad [29]. Figur 7 nedan illustrerar modellen.



Figur 7: Richardsons mognadsmodell. Källa [34].

- Nivå 0 innebär att HTTP enbart används som transportmekanism för interaktion mellan tjänstekonsumenter och producenter. Denna nivå använder sig oftast av HTTP-metoden POST för att skicka data till *en* given URI som agerar kontaktyta för *all* interaktion med tjänsten [17][22]. Denna nivå påminner om vanlig procedurbaserad programmering där dokumentet (vanligen XML-baserat) som skickas innehåller den metod som ska anropas med tillhörande parametrar. SOAP befinner sig på nivå 0.
- Nivå 1 delar upp tjänsten i interaktion med olika resurser (med distinkta URI) men följer annars samma mekanismer för RPC som nivå 0.
- Nivå 2 använder sig av ett enhetligt gränssnitt baserat på HTTP-metoder för att interagera med resurser. I denna nivå utgår en tjänsteproducent från den överenskomna semantiska innebörden av olika HTTP-metoder. Exempelvis så innebär HTTP-metoden GET att konsumenten (klienten) enbart är intresserad av att få en representation av resursen, POST att klienten vill skapa en ny resurs, PUT att klienten vill förändra resursen och DELETE att klienten vill radera resursen.
- Nivå 3 inkluderar hypermedia för synliggöra och möjliggöra klientens olika valmöjligheter i fråga om att läsa och förändra resurser. Exempelvis så kan en given representation redogöra för var ytterligare data kring ett givet innehåll kan finnas. Webbtjänster som *helt* följer principerna för HATEOAS (se avsnitt 2.5.4 ovan) befinner sig *alltid* på nivå 3.

3.2.2 Verktyg

För att verifiera lösningsförslaget så sattes en testmiljö bestånde av två olika Motrice instanser upp. Den första instansen exekverade på en Ubuntu 14.04 server, medan den andra befann sig på en virtualiserad Lubuntu 14.04 server. Den virtualiseringsprogramvara som användes var Oracles VirtualBox[31] där

nätverket för den virtualiserade servern sattes till *bridged mode.* Det innebar att all nätverkskommunikation skedde genom enhetsdrivrutiner på virtualiseringshosten med följden att virtualiseringhosten ser gästsystemet som fysiskt kopplad till nätverkskortet, vilket är det är det närmaste man kan komma för att emulera två olika fysiska maskiner [32].

Inledningsvis testades varje steg av lösningsförslaget genom att process A och process B befann sig på en och samma Motriceinstans. Meddelandena mellan process A och B skedde över serverns virtuella loopback gränssnitt [33]. Loggen för Motrice instansen avlästes för att säkerställa att kommunikationen skedde med synkrona anrop och svar för varje meddelande som skickads mellan processinstanserna. När hela lösningsförslaget var färdigutvecklat instansierades process A och process B på var sin server med den konfiguration som är angiven ovan. Loggarna för var Motriceinstans avlästses för att säkerställa att kommunikationen följde samma mönster som tidigare. Bilaga B går har anvisningar för hur en testmiljö kan sättas upp.

4 Lösningsförslag

Detta kapitel beskriver ett lösningsförslag för hur kommunikation mellan verksamhetsprocesser exponerade som tjänster kan ske genom hypermedia. Förslaget baserar sig på den teoretiska grunden i kapitel 2 ovan och hur den kan appliceras för att lösa problemet med kommunikation mellan verksamhetsprocesser.

4.1 Kravspecifikation

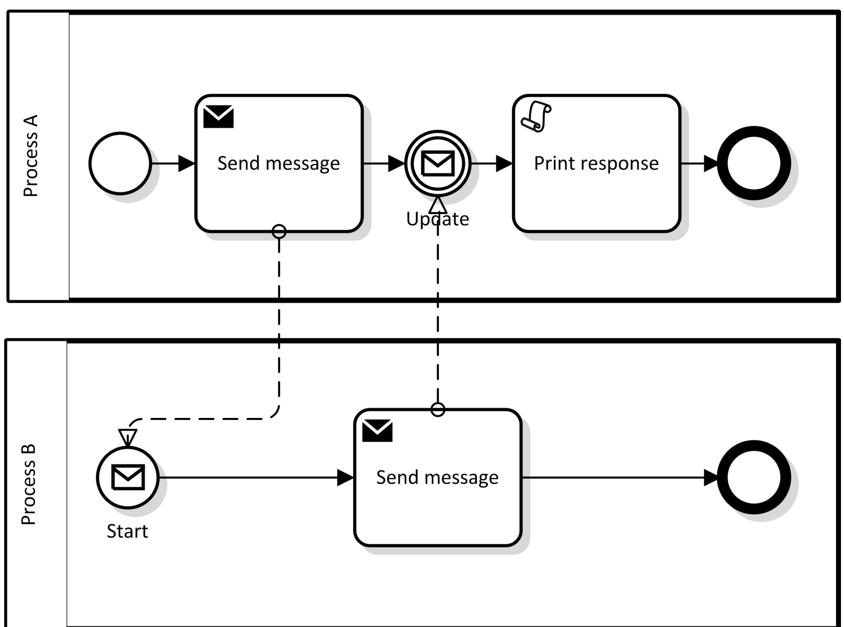
4.1.1 Övergripande krav

Kraven på lösningen följande:

- Verksamhetsprocesser, exponerade som tjänster, ska vara unikt addresserbara genom en URL. Detta medför att processerna bli identifierbara/adresserbara resurser i enlighet med principerna för ett enhetligt gränssnitt (se avsnitt 2.5.2 ovan).
- Tjänsterna ska exponera verksamhetsprocesser som representationer baserade på en definierad media typ som både tjänsteproducent och tjänstekonsument har kunskap om. Media typen är en grundläggande mekanism för att båda parter ska kunna interagera med varandra (se avsnitt 2.5.1).
- Meddelanden som flödar mellan verksamhetsprocesserna ska vara självbärande. Det vill säga att de ska använda sig av HTTP-protokollets semantik för interaktion med resurser, metadata som anger meddelandets innehåll samt representationen i form av en definierad media typ. (se avsnitt 2.5.3).
- Hypermedia ska användas för att förändra verksamhetsprocessernas tillstånd. Det medför att vetskapen om *vad* som ska göras, *var* det ska göras och *hur* det ska göras utbyts i själva interaktionen mellan tjänsteproducet och tjänstekonsument (se avsnitt 2.5.4).

4.1.2 Processmodell

Den processmodell och scenario för interaktion mellan verksamhetsprocesser som har använts som utgångspunkt presenteras i figur 8 nedan. Modellen bygger på hur kommunikation sker mellan olika processer enligt BPMN 2.0 standarden. Det innebär konkret att kommunikationen mellan processer måste bygga på meddelandeflöden [10]. Specifikationen tillåter meddelandeflöden mellan olika typer av aktiviteter, men för enkelhetens och tydlighetens skull så sker all kommunikation i modellen mellan meddelandebaserade aktiviteter och händelser. Ett annat viktigt skäl är att för meddelandebaserade aktiviteter och händelser, till skillnad från icke-meddelandebaserade dito, så är ett namngivet meddelande *nödvändigt* för att processen ska påbörjas / fortsätta. Detta medför att process A och process B kräver att det är en extern part så aktiverar respektive process, och inte någon annan intern mekanism.



Figur 8: Processmodell med meddelandeflöden

För att leva upp till principerna för REST så måste även följande kriterier för interaktionen mellan process A och B vara uppfyllda:

- Process A har den enbart kunskap om den URL som process B befinner sig på samt det namngivna meddelande som B väntar på (”start”).
- Process B påbörjar sin exekvering först när rätt namngivna meddelande mottas.
- Process B får inte ha någon kunskap om A på förhand utan all interaktion ska ske genom att A och B utbyter representationer innehållande hypermedia.

4.2 Komponenter

Kraven i avsnitt 4.1 ovan mynnar ut i ett antal övergripande komponenter som behöver utvecklas:

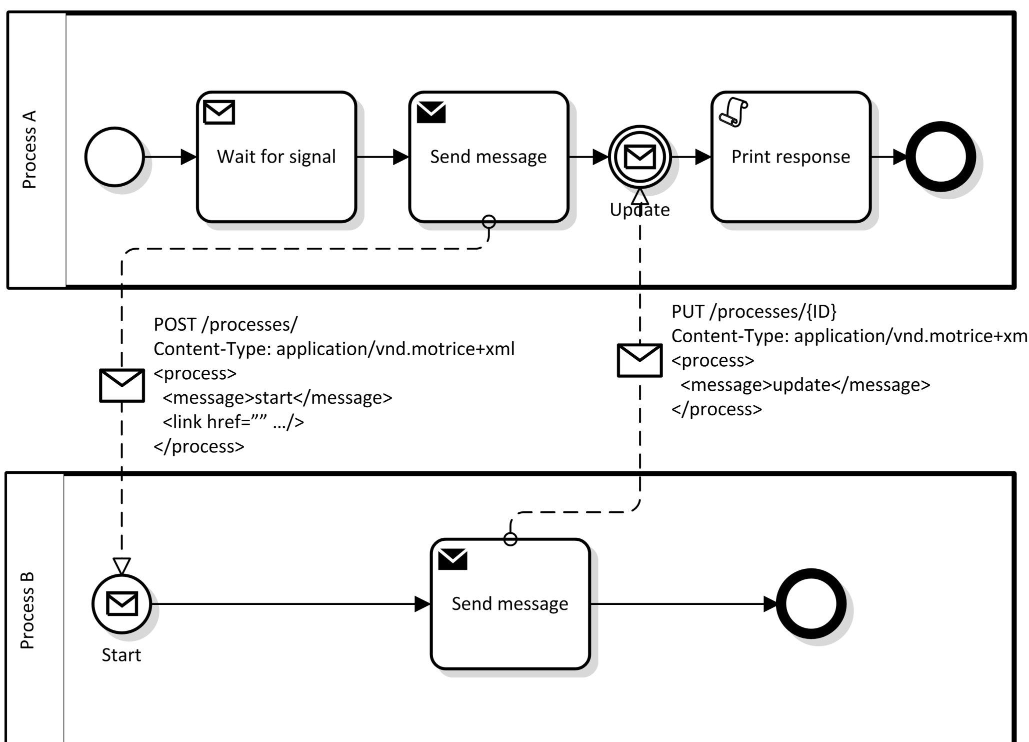
- Två konkreta verksamhetsprocesser, definierade enligt BPMN 2.0, som följer processmodellen under avsnitt 4.1.2 ovan. Verksamhetsprocesserna används för att illustrera hur BPMN 2.0-processer kan exponeras som tjänster där interaktionen mellan sker genom meddelandeflöden i enlighet med principerna för REST.
  - Eftersom Activiti inte har stöd för aktiviteter som skicka meddelanden enligt BPMN 2.0 standarden behöver en sådan utvecklas. De meddelande som skickas ska leva upp till principerna för självbeskrivande meddeladen.
- Ett webbaserat tjänstelager som exponerar verksamhetsprocesser som addreserbara tjänster (resurser) där interaktionen sker genom HTTP-metoder och exponering/konsumtion av representationer baserade på en media typ. Tjänstelagret har som uppgift skapa representationer utifrån de underliggande BPMN 2.0 processerna, vidarebefordra anrop till rätt

specifika resurs och tolka innehållet i de representationer som för över i anropen.

- En media typ som kan användas som underlag för de meddelanden (representationer) som ska flöda mellan verksamhetsprocesserna. Media typen bär semantiken för innehållet i representationen och gör det möjligt för båda parter att agera på de representationer som utbyts. Utan en media typ så finns det ingen explicit överenskommelse av hur datamängderna i representationerna ska tolkas vilket bryter mot principen om självbeskrivande meddelanden.

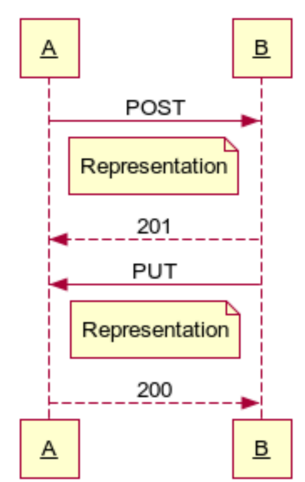
4.3 Implementation

De konkreta processimplementationerna inklusive meddelandeflöden enligt principerna för REST visas i figur 9 nedan. Process A och process B befinner sig i olika instanser av Motrice och interagerar helt genom HTTP metoder, representationer och hypermedia.



Figur 9: Processimplementationer och meddelandeflöden

Som figuren ovan visar så sker all interaktion mellan processerna genom meddelandeflöden. I BPMN 2.0 standarden finns det tre tillämpbara konstruktioner för att hantera meddelandeflöden mellan två processer: *avsändaraktiviter*, *meddelandebaserade starthändelser* samt *meddelandebaserade fångande händelser*. De förstnämnda innebär att processen skickar ett givet meddelande till en given mottagare innan den fortsätter sin exekvering. *Starthändelser* respektive *fångande händelser* baserade på meddelanden innebär att en process inväntar ett namngivet meddelande innan den påbörjar respektive fortsätter sin exekvering. Det innebär att dessa händelser kan användas för att exponera BPMN 2.0 processer som resurser likt Pautassos förslag (se avsnitt 2.6 ovan).



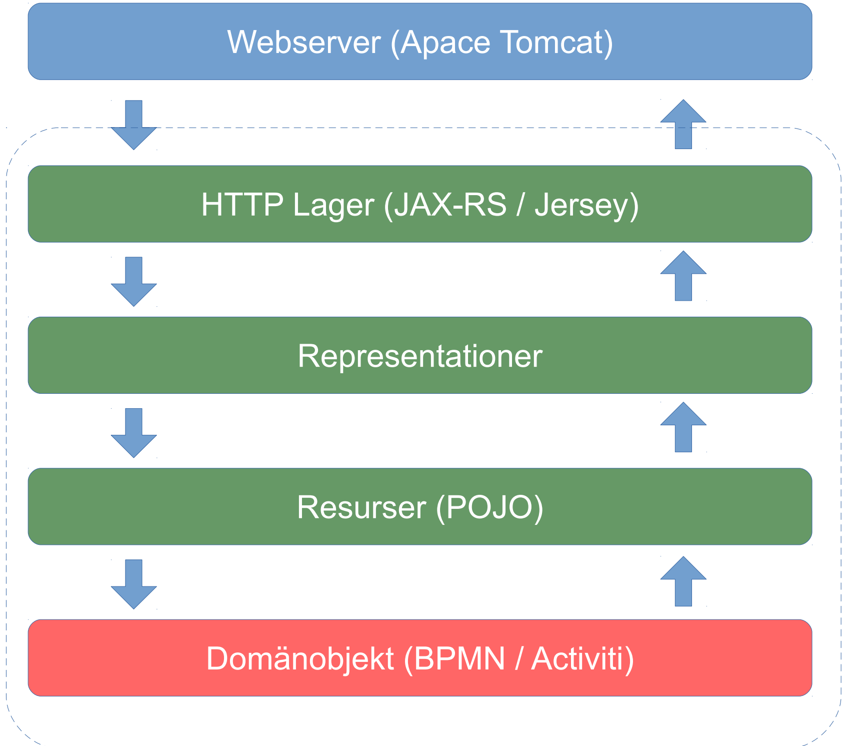
Figur 10: Meddelandeflöden som HTTP-interaktioner

Figur 10 till vänster illustrerar hur kommunikationen mellan processerna ser ut i form av HTTP-interaktioner med tillhörande statusmeddelanden (se avsnitt 4.3.2 nedan för en utförligare diskussion). Interaktionen sker i följande steg:

1. A anropar B genom HTTP metoden POST till en känd URI där B är exponerad. Anropet innehåller en en representation på en överenskommen media typ som innehåller *all* den information B behöver för att påbörja sin exekvering samt skicka ett meddelande till A vid ett senare tillfälle. I praktiken måste representationen innehålla ett namngivet meddelande som B ”lyssnar” på samt en *callback* länk som anger till var B ska skicka ett meddelande. Avsnitt 4.3.2 behandlar hur A och B kan exponeras som adresserbara resurser medan avsnitt 4.3.3 behandlar media typen och representationer närmare.
2. Givet att inget fel uppstår svarar B med HTTP-statusmeddelandet 201 för att klargöra att en förändring av resursen har skett och både A och B fortsätter sin exekvering. Avsnitt 4.3.2 diskuterar HTTP-interaktioner och statusmeddelanden närmare.
3. Vid ett senare tillfälle skickar B ett meddelande genom HTTP metoden PUT till den URI som angavs i den representation A skickade i steg 1. Om inget fel uppstår svarar A med HTTP-statusmeddelandet 200 för att signalera att meddelandet är mottaget och exekveringen fortlöper.

4.3.1 Övergripande arkitektur

Implementationen av lösningen följer den arkitektur som illustreras i figur 11 nedan och är starkt inspirerad av Webber *et al* [22].



Figur 11: Övergripande arkitektur för lösningsförslaget

Syftet med arkitekturen är att skapa löst kopplade lager där varje lager fokuserar på att sköta en uppgift och säkeställa att varje enskilt lager är så isolerat från omgivande lager som möjligt. De delar som lösningsförslaget berör i huvudsak har en grön färg i figuren ovan och består av komponenter som agerar som en brygga mellan webbservern (blå färg i figuren ovan) och underliggande domänlager (röd färg i figuren ovan). Lösningsförslaget gör enbart ingrepp i befintlig kodbas vid ett tillfälle (se avsnitt 4.3.4.1 nedan).

4.3.2 HTTP lager och resurser

Det API som har utvecklats för lösningen har som huvudsaklig uppgift att exponera definierade BPMN 2.0 processer som resurser med vilka klienter kan interagera genom representationer på en överenskommen media typ (se avsnitt 4.3.3 nedan). Grunden för API:et är referensimplementationen av standarden JSR-339 (JAX-RS), Jersey (version 1.18.1) [38][39]. Det finns för- och nackdelar med att använda sig av referensimplementationer och skälet till valet är att Jersey redan används av ett antal andra komponenter i Motrice, att den är helt öppen (både källkodsmässigt och dokumentationsmässigt) och att den är en etablerad och spridd standard.

JAX-RS använder sig av annotationer på vanliga Java klasser för att exponera dessa som resurser mot vilka klienter kan interagera. Ett exempel på hur annotationerna ser ut i praktiken finns i figur 12 nedan.

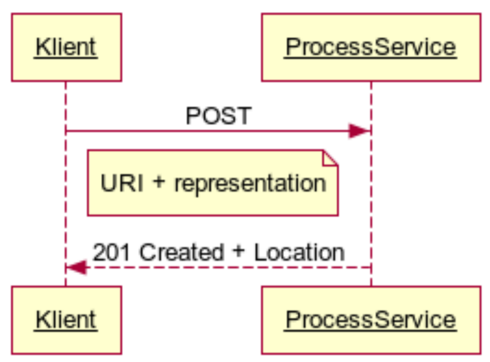
```
@PATH (”processes”)
Class ProcessSerivce
{
  @PATH(”instances/{id}”)
  @PUT
  @Consumes(”application/vnd.motrice+xml”)
  @Produces(”application/vnd.motrice+xml”)
  public Response updateProcessInstance(@PathParam(”id”) String id,
                                        String content)
  {
    …
  }
  …
}
```

Figur 12: Annotationer enligt JAX-RS

Annotationen `@PATH` används för att exponera en vanlig Java klass eller metod som en resurs på en angiven relativ URL. Utifrån figuren ovan så kommer URL:en `http://www.exempel.com/processes` att vara knuten till resursen `Process-Service.` `@PATH` kan även användas för att deklarera URL-mallar där delar av URL:en kan extraheras som variabler och användas vid runtime. Annotationen av metoden `updateProcessInstance()` ovan anger exempelvis att den del av URL:en som följer efter `/processes/instances` ska knytas till variablen `id`. Annotationerna `@GET`, `@POST`, `@PUT` m fl används som annotationer på specifika metoder föra att ange vilken HTTP-metod de ska svara mot. `@Consumes` och `@Produces` anger vilken media typ som HTTP-anropet respektive svaret hanterar. Om en klient anger en felaktig media typ så kommer Jersey att svara med ett meddelande av typen `406 not acceptable` eller `415 unsupported media type` för att klargöra att resursen existerar och accepterar HTTP-metoden, men kan inte hanterar den efterfrågade media typen.

4.3.2.1 Resursen ProcessService

Den resurs som har skapats till lösningen, `ProcessService`, har som uppgift att dirigera HTTP-anrop till rätt metod, extrahera och skicka vidare information från representationerna samt att omvandla resultat och undandtag till HTTP-svar. `ProcessService` har en enkel logik och använder sig av lösningsförslagets representationer (se avsnitt 4.3.3) vid extrahering av information samt Jerseys `Response` klass för att bygga de svar som returneras till en klient.

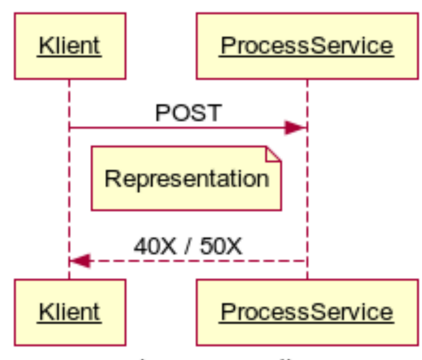


Figur 13: Sekvensdiagram för lyckosamma anrop och svar

Vid ett lyckosamt anrop från klienten, det vill säga att ProcessService lyckades extrahera information från representationen och använda informationen för att anropa ett underliggande domänobjekt, liknar interaktionen det sekvensdiagram som visas i figur 13 ovan. I figuren ämnar klienten skapa en ny resurs och anropar därmed `ProcessService` med HTTP-metoden `POST` där meddelandekroppen

innehåller en representation av media typen `application/vnd.motrice+xml`. Från representationen extraherar ProcessService det namngivna meddelandet som en underliggande process svarar på samt den eventuella URI som den instansierade processen ska anropa. Givet att en ny resurs kan skapas svarar `ProcessService` klienten med statusmeddelandet `201 Created` där HTTP-headern `Location` håller en URL till den nyskapade resursen.

Om anropet från klienten inte lyckas av något skäl, till exempel att den efterfrågade resursen inte existerar eller att `ProcessService` inte lyckades anropa något underliggande domänobjekt utifrån den information som extraherats, så kommer interaktionen att följa det generella mönster som sekvensdiagramet i *figur 14* till höger visar. Om felet består i att klienten efterfrågar en resurs som inte existerar, anropat en resurs med en felaktig HTTP-metod, har angivit en felaktig media typ eller liknande svarar `ProcessService` med statusmeddelanden av 40X-typen för att tala om att anropet måste justeras. Om felet består i något internt fel, exempelvis att domänobjekten inte är nåbara av något skäl, returneras ett statusmeddelande av 50X-typen för att signalera att klienten kan försöka vid ett senare tillfälle.



Figur 14: Sekvensdiagram för anrop och svar vid fel

En mer utförlig dokumentation av ProcessService och de metoder som exponerar resurser finns i bilaga A.

4.3.3 Representationer

Representationer är en nyckel till att skapa distribuerade tjänster som bygger på principerna för REST. En representation baseras på en given media typ som både klienten och servern kan konsumera och producera (se avsnitt 2.5.1 och 2.5.3 ovan).

4.3.3.1 Media typen `application/vnd.motrice+xml`

Den media typ som lösningsförslaget använder sig av är `application/vnd.motrice+xml` och bygger i sin enklaste form på ett antal XML-element som tillsammans bygger upp ett meddelande som indikerar en önskvärd förändring av en resurs. En XML-representation av media typen ser ut som följer:

```
<process>
  <message></message>
  <link href=”...” rel=”...” type=”...” title=”..” data-method=”...”
data-message=”...”/>
</process>
```

Figur 15: XML-representation av media typen `application/vnd.motrice+xml`

Rootelementet <process> *måste* innehålla elementet `<message>` och *kan* innehålla ett eller flera element av typen `<link>`. Elementet `<message>` innehåller det namnginva meddelande som en processdefinition eller en specifik processintans väntar på. `<link>` elementet bygger på motsvarande element i HTML5 standar-

den tillsammans med en utökning av applikationsspecifika attribut som standarden medger (angivna som `data-*`) [40]. Syftet med `<link>` är att skapa ett element som är helt kapslar in all den information som behövs för att skicka en representation till en angiven URL. Attributet `href` håller den URL som ska anropas och kommer i praktiken att ange adressen till den anropande process instansen. `rel` håller den semantiska relationen mellan den innevarande representationen och den resurs som den anginva URI pekar på. Det vill säga att värdet på rel attributet anger hur URL:en ska tolkas och används tillsamans med resterande attribut för att tala om vad en anropande process ska göra. I lösningsförslaget innebär värdet ”update/message” att URL:en pekar på en resurs som förväntar sig ett namngivet meddelande. Attributet `type` anger vilken media typ som resursen kan hantera, `title` kan användas för en mer utförlig förklaring av URL:en. De sista attributen, `data-method` och `data-message`, anger vilken HTTP-metod respektive vilken meddelande som ska användas i anropet.

Attributen för link gör det möjligt att skapa meddelanden som helt lever upp till principerna för ett enhetligt gränssnitt (se avsnitt 2.5 ovan) i och med att en representation av en given media typ kan skapas och skickas till en identifierad resurs där all information blir känd först vid runtime.

4.3.3.2 JAXB och representationer

De representationer som överförs mellan en klient och `ProcessService` byggs av ett antal vanliga Java-klasser som använder sig av JSR 222, *Java Architecture for XML Binding (JAXB) 2.0*, för att enkelt kunna skapa XML-element [41]. JAXB bygger, precis som JAX-RS, på att annotationer för att deklarera hur en Java klass ska representeras som XML. Ett konkret exempel på hur det ser ut finns i figur 16 nedan.

```
@XmlRootElement(namespace = Representation.MOTRICE_NAMESPACE)
public class Link {
  @XmlAttribute(name = "rel")
  private String rel;
  @XmlAttribute(name = "uri")
  private String uri;
  @XmlAttribute(name = "mediaType")
  private String mediaType;
  …
}
```

Figur 16: Exempel på JAXB annotationer

Annotationen `@XmlRootElement` anger att klassen som sådan ska betraktas som ett root-element. `@XmlAttribute` anger att det element som skapas har ett attribut med ett givet namn till vilket ett värde kan knytas. Den resulterande representationen från koden ser ut som elementet `<link>` i figur 15 ovan. De klasser som bygger upp representationen redovisas kort nedan.

4.3.3.3 Link

Klassen `Link` är en grundsten i att bygga representationer som uppfyller kraven för REST. Enkelt uttryckt bör en länk innehålla följande information: En URI, en beskrivning av relationen mellan den nuvarande resursen och resursen bakom den angivna URI:n samt vilken media typ som resursen bakom den anginva

URI:n förväntar sig. Denna information hålls i XML-representationen av klassen `Link` av attributen `uri`, `rel` respektive `mediaType`. I Java implementationen motsvaras attributen av de privata medlemmarna som syns i figur 16 ovan.

4.3.3.4 Representation

Den abstrakt basklassen `Representation` agerar utgångspunkt för att skapa olika typer av representationer och är inte mycket mer än en behållare för en samling länkar (objekt av klassen Link). Klassen håller även vanligt förekommande parametrar som används av övriga klasser i lösningsförslaget.

4.3.3.5 ProcessRepresentation

Klass som ärver `Representation` och är den kompletta implementationen av den media typ som lösningsförslaget använder sig av.

4.3.4 Domänobjekt

Activiti har befintliga implementationer av meddelandebaserade starthändelser respektive meddelandebaserade fångande händelser som används i processdefinitionerna. Det Activiti saknar är en implementation av avsändaraktiviteter för att skicka meddelanden till en given mottagare. Denna har skapats som en del i lösningsförslaget och redogörs för i mer detalj under avsnitt 4.3.4.1 nedan.

En viktig begränsning med meddelanden och meddelandehändelser är att det namngivna meddelandet måste vara helt unikt inom en Activiti runtime [42]. Det innebär att två olika processer som har meddelandebaserade starthändelser inte kan vänta på meddelanden med samma namn. Exempelvis så kan meddelandet ”start” enbart kopplas till en starthändelse.

4.3.4.1 Avsändaraktivitet

Den meddelandeaktivitet av avsändartyp som har utvecklats, `MessageTaskDelegate`, bygger på principen om självbeskrivande meddelanden (se avsnitt 2.5.3 ovan) och har som uppgift att bygga ett enkelt men komplett HTTP meddelande utifrån ett antal variabler. Dessa variabler är den URL som ska anropas, vilken HTTP-metod som ska användas, den media typ som gäller för meddelandet samt själva representationen som skickas.

I lösningsförslaget så tillhandahålls variablerna som processvariabler vid en instansiering av processen. För att underlätta utvecklingen av specialiserade klasser som behöver åtkomst till processvariabler tillhandahåller Activiti den abstrakta basklassen `JavaDelegate`. `MessageTaskDelegate` ärver `JavaDelegate` och kan genom att implementera den abstrakta metoden `execute(DelegateExecution execution)` inspektera `execution`-objektet för att få information om den aktiva processinstansen, inklusive processvariabler.

5 Resultat

5.1 Utfall av test i testmiljö

Vid upprepade exekveringar i testmiljön så har resultaten varit entydiga. Tabell 1 nedan visar en summering av loggarna från de två Motriceinstanserna A och B (röd respektive grön färg). Notera att process A exekverar i instans A och process B exekverar i instans B.

Händelselogg	Beskrivning

`MessageTaskDelegate:`
`POST`
`http://192.168.1.89:8080/restrice/jersey/runtime/processes`
	`MessageTaskDelegate` i process A anropar process B med en representation innehållande ett namngivet meddelande samt en callback som innehåller URL:en till process A.

`ProcessService:` `REQUEST to start process by message=[start]`	`ProcessService` hos B exraherar det namngivna meddelandet, callback länken och påbörjar sin exekvering.
`ProcessService:` `RESPONSE is 201 with Location=[http://192.168.1.89:8080/restrice/jersey/runtime/processes/6412]`	`ProcessSerivce` hos B svarar A med `Location` headern satt till URL:en för process B

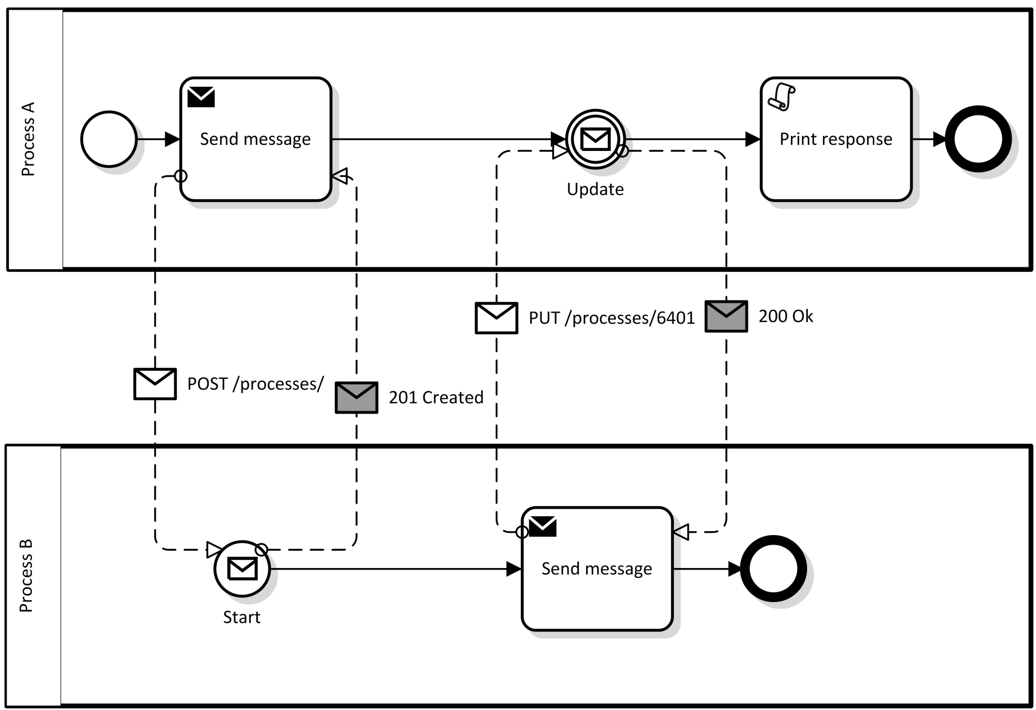
`MessageTaskDelegate:`
`RESPONSE`
`http://192.168.1.89:8080/restrice/jersey/runtime/processes/6412`
	`MessageTaskDelegate` i process A får ett svar från process B innehållande URL:en till process B.

`MessageTaskDelegate:`
`PUT`
`http://192.168.1.89:8080/restrice/jersey/runtime/processes/6401`
	`MessageTaskDelegate` i process B anropar process A med den metod, URL samt det namngivna meddelande som har extraherats från process A:s tidigare anrop.

`ProcessService:` `REQUEST to update process instance with ID=[6401]`	`ProcessService` hos A extraherar ID och det namngivna meddelandet från B och uppdaterar process A asynkront.
`ProcessService:` `RESPONSE is 200 Ok`	`ProcessService` hos A svarar B med att anropet har lett till en förändring.
`MessageTaskDelegate:` `RESPONSE`	B får svar från A
`proc A`	Process A skriver ut meddelandet i sin sista aktivitet.

Tabell 1: Händelselogg

Figur 17 nedan illustrerar interaktionena mellan process A och B utifrån en förenklad version av processimplementationen.



Figur 17: HTTP-interaktioner mellan process A och B.

De vita/tomma ”kuverten” visar anropen medan de gråa/ifyllda visar svaren. Notera att flödena mellan processerna i figuren ovan *inte* ska tolkas som meddelandeflöden enligt BPMN 2.0 standarden, utan enbart syftar till att visa på de HTTP-interaktioner som sker. Som figuren visar sker varje meddelandeutbyte mellan process A och process B med synkrona HTTP-interaktioner. När A anropar B med en representation innehållande ett namngivet meddelande och en hypermedia länk så svarar B utan att den egna processen har exekverat klart. Detta beror på att resurslagret använder sig av icke-blockerande anrop till det underliggande domänlagret. Om lösningsförslaget använt sig av blockerande anrop så hade ett möjligt utfall varit att process A fått svaret för sitt första anrop efter det att B har anropat och fått svar från A.

6 Analys

6.1 Efterlevnad mot Richardsons mognadsmodell

För att uppnå nivå 3 i Richardsons mognadsmodell i avsnitt 3.2.1 ovan krävs att lösningsförslaget uppfyller följande kriterier:

- Processer är adresserbara resurser.
- HTTP-metoder och statusmeddelanden med deras semantiska betydelse används vid interaktion med resurser.
- Förändringen av resurser bygger på utbyte av representationer.
- Hypermedia används för att vägleda interaktionen vidare.

Det första och andra kriteriet uppfylls med komponenten `ProcessService` som bygger ett tjänstelager ovanför domänobjekten där processer exponeras med hjälp av URL:er och där interaktionen sker med HTTP protokollet som ett *applikationsprotokoll*. Specifikt så tar lösningsförslaget sin utgångspunkt i att meddelandehändelser är de noder där processer exponeras som en resurser medan meddelandeflöden implementeras som synkrona anrop och svar bestående av självbeskrivande meddelanden. Den semantiska betydelsen för HTTP anrop och svar används även för att särskilja anrop som skapar en resurs (POST) från de som syftar till att uppdatera en resurs (PUT) samt att informera klienten om utfallet genom statuskoder.

Det andra och tredje kriteriet uppfylls genom att de representationer som skapas med hjälp av Java-klasserna `Representation`, `ProcessRepresentation` och `Link` innehåller all nödvändig information för att påbörja och uppdatera de exponerade processerna. Detta synliggörs tydligast i den interaktion där A skickar ett meddelande tillbaka till B. B har inte någon *a priori* kunskap om var A finns, vilket namngivet meddelande som förväntas eller ens vilken HTTP metod som ska användas. All sådan vetskap byggs *a posteriori* efter det att A har överfört en representation på en överenskommen media typ till B. Media typen, som båda parter förutsätts ha kunskap om, håller den begreppsliga ramen som gör att B kan tolka innehållet och konfigurera de parametrar som krävs för att skicka ett självbeskrivande meddelande tillbaka till A. Det självbeskrivande meddelandet byggs i sin tur helt utifrån den hypermedia kontroll som är angiven i media typen genom elementet `<link>`.

6.2 Efterlevnad mot BPMN 2.0 och processmodell

Vid en jämförelse av processmodellen i avsnitt 4.1.2 och processimplementationen i lösningsförslaget under avsnitt 4.3.4 finns det en liten men avgörande skillnad: processimplementationen har en extra aktivitet i form av ”wait for signal”. Denna aktivitet är en konsekvens av Actitivs blockerande metodanrop vid exekvering av processer och krävs för att extrahera ett ID (se bilaga A för en utförligare diskussion). Bortsett från den aktiviteten så använders sig både pro-

cessmodellen och processimplementationerna av meddelandehändelser och meddelandeflöden för sin interaktion, vilket är helt enligt BPMN 2.0 standarden. Det som saknas i implementationen är möjligheten för processerna att inspektera sig själva och utifrån introspektionen generera vilka meddelanden som processen väntar på. Detta medför att resurslagret har kunskap om domänobjektens interna logik.

7 Diskussion

7.1 Resurser, kommunikation och REST

De frågor som detta examensarbete har sökt svar på är om det är möjligt att exponera en processmotor och dess processer som tjänster enligt principerna för REST och om, i fall det är möjligt, även kommunikation mellan processer och processmotorer kan sker enligt samma principer. Det lösningsförslag som har tagit fram inom ramen för detta arbete visar att detta är fullt möjligt. I praktiken innebär det att en process som finns i en processmotor måste exponeras som en adresserbar *resurs* där *interaktionen med andra processer sker genom utbyte av representationer innehållande hypermedia*. Lösningsförslaget tar sin utgångspunkt i BPMN 2.0 standarden där *meddelandehändelser* och *meddelandeflöden* är de mekanismer som möjliggör kommunikation mellan processer. Meddelandehändelser är de noder i en process som anger var extern interaktion med en process är möjlig varför dessa är en gångbar utgångspunkt för att exponera processer som resurser. Exponeringen som sådan sker genom ett tjänstelager baserat på JAX-RS där HTTP anrop inspekteras och dirigeras till rätt process. Meddelandeflöden sker genom överföringar av representationer baserade på hypermedia där representationen innehåller all nödvändig information (vad, hur och vad) för att båda den avsändande prcessen och den mottagande processen ska kunna fortlöpa. Dessa egenskaper i lösningsförslaget påvisar att principerna för REST är förenliga med BPMN 2.0 avseende kommunikation mellan processer och processmotorer.

Lösningsförslaget påvisar även att Pautassos idè om att utvidga BPMN 2.0 standarden med nya notationer för att ange processer eller delar av processer som adresserbara resurser har bärighet som koncept, men att det inte krävs någon utveckling av BPMN 2.0 standarden för att uppnå det resultat som önskas. Genom att införa en ny specifik notation för att exponera processer och/eller aktiviteter som resurser så sammanblandar Pautasso *vad* som ska göras med *hur* det görs. Utifrån Pautassos förslag så hade exempelvis interaktionen mellan processer skett genom att specifika aktiviteter i processerna var angivna som resurser i själva modellen. Detta innebär att modelleringen av en process (vad som ska göras) behöver ta hänsyn till vilken implementation som används för den konkreta interaktionen (hur det ska göras). Som diskuterats ovan så är meddelandehändelser och meddelandeflöden implementerade enligt principerna för REST tillräckliga för att uppnå det Pautasso strävar efter.

7.2 Begränsningar och förbättringsmöjligheter

Det finns två större brister i lösningsförslaget som kan härledas till implementationen av Activiti. Den första är att det krävs en aktivitet i process A som placerar processen i ett väntetillstånd för att kunna extrahera processens runtime ID. Den andra är att resurslagret, i form av klassen `ProcessService`, har kunskap om vilket meddelande som process A väntar på i sin fångande meddelandeaktivitet.

Det sistnämnda innebär att kunskap om domänlagret läcker till överliggande lager vilket bryter principen om väl avgränsade och löst kopplade lager. För att komma till rätta med bristerna hade det varit önskvärt om Activiti stödde följande tre fall:

- Möjligheten att skapa en processinstans utan att den påbörjar sin exekvering.
- Möjligheten att asynkront påbörja exekveringen av en processinstans. Specifikt är det önskvärt om `startProcessByMessage()` eller motsvarande är icke-blockerande eller har icke-blockerande alternativ.
- Möjligheten att inspektera en pågående processinstans för att se vilka meddelandeaktiviteter eller händelser som befinner sig senare exekveringskedjan.

Ovanstående punkter hade gjort det möjligt för ett resurslager att instansiera en process, generera en (eller flera) callback länkar, skicka svaret till den anropande klienten. Informationen om vilket namngivet meddelande som skulle skickas till vilken resurs hade då varit möjlig att generera vid runtime och helt frikopplat resurslagret från domänlagret.

7.3 Vidareutveckling och framtida forskning

I gränslandet och korsbefruktningen mellan BPMN 2.0 och REST finns ett i stort outforskat område som kan bidra till utvecklingen av distribuerade tjänster med avancerade processflöden. Utifrån det lösningsförslag som har tagit fram inom ramen för detta arbete så kan ett antal olika fall för vidareutveckling tänkas. Bland andra så är en mer komplex interaktion mellan två processer där anrop till en meddelandehändelse påverkar hur processen fortlöper och inte enbart *att* den fortlöper. I en sådan interaktion måste den process vars exekveringsbana påverkas meddela vilka nästa möjliga interaktioner är (om de finns) inklusive vilken förändring anropet innebär. Eftersom den semantiska förståelsen för vad dessa förändringar innebär kan gå utöver vad som ryms inom en och samma media typ, så är ett intressant alternativ att tillämpa s k mikroformat [43]. Mikroformat innebär att definierade standarder med tillhörande semantik för olika områden införlivas i representationer för att möjliggöra en maskinell behandling.

Även om REST har visat sig vara ett gångbart alternativ för kommunikation mellan processer och processmotorer så finns det andra vägar som är värda att undersöka. Ett sådant förslag är att undersöka tillämpningen av publish-subscribe/Observer mönstret [44] där meddelandeflöden inte baseras på synkron klient-server kommunikation, utan på att tjänstelagret publicerar information om processer (vilka som finns, var de finns och vad de behöver) till en central nod som sedan vidarebefordrar informationen till alla som är intresserade av informationen. En potentiell fördel med publish-subscribe är att klienten inte måste ha någon *á priori* kunskap om var tjänsten är eller förväntar sig, utan denna information kan förmedlas via det dataflöde som tjänstelagret för tjänsten producerar.

7.4 Samhälleliga effekter

Överlag så visar lösningsförslaget att kombinationen av BPMN 2.0 och REST bidrar till en effektivare utveckling av processorienterade distribuerade tjänster, vilket i sin tur leder till effektivare distribuerade system. Tack vare att båda är vedertagna standarder (den senare en *de facto* ”standard” för distribuerade system) så bör lösningar som bygger på det koncept som presenteras i lösningsförslaget vara återanvändbara, vilket innebär en lägre instegskostnad för de organisationer som väljer att arbeta med processorienterade applikationer.

Källförteckning

[1] Activiti, ”Activiti BPMN Platform”, http://activiti.org. Hämtad 2014-07-06.

[2] Orbeon, ”Orbeon Forms”, http://www.orbeon.com/. Hämtad 2014-07-06,

[3] Hippo, ”Hippo”, http://www.onehippo.com/. Hämtad 2014-07-06.

[4] PostgreSQL, ”PostgreSQL”, http://www.postgresql.org/. Hämtad 2014-08-22.

[5] Spring, ”Spring IO”, http://spring.io/. Hämtad 2014-08-22

[6] The Apache Software Foundation, ”Apache Tomcat”, http://tomcat.apache.org/. Hämtad 2014-07-06.

[7] A. Ljungberg & E. Larsson, *Processbaserad verksamhetsutveckling.* 2 uppl. Lund: Studentlitteratur, 2012.

[8] J. Jeston & J. Nelis, *Business Process Management*. 3 uppl. New York: Routledge, 2014.

[9] Association of Business Process Management Professionals, *Guide to the Business Process Management Common Body of Knowledge.* 2 uppl. www.abpmp.org

[10] Object Management Group, *Business Process Model and Notation (BPMN), version 2.0.* www.omg.org

[11] B. Silver, *BPMN Method & Style: With BPMN Implementers Guide*. 2 uppl. Aptos: Cody-Cassidy Press, 2011.

[12] T. Erl, *SOA Principles of Service Design*. Upper Saddle River, New Jersey: Prentice Hall, 2007.

[13] The Open Group, ”Service Oriented Architecture: SOA Features and Benefits”, http://www.opengroup.org/soa/source-book/soa/soa_features.htm. Hämtad 2014-05-29.

[14] A. Tanenbaum & M. Van Steen. *Distributed Systems: Principles and Paradigms.* 2 uppl. Upper Saddle River, New Jersey: Prentice Hall, 2007.

[15] N. Josuttis. *SOA in Practice*. Sebastopol, Kalifornien: O'Reilly Media, 2007.

[16] W3C, ”Web Services Architecture”, http://www.w3.org/TR/ws-arch/. Publicerad 2004-02-11. Hämtad 2014-06-19.

[17] L. Richardson & M. Amundsen. *RESTful Web APIs*. Sebastopol, Kalifornien: O'Reilly Media, 2013.

[18] Wikipedia, ”SOAP”, http://en.wikipedia.org/wiki/SOAP. Hämtad 2014-06-19.

[19] M. Angstadt, ”WSDL SOAP bindings confusion – RPC vs document”. http://mangstacular.blogspot.se/2011/05/wsdl-soap-bindings-confusion-rpc-vs.html. Publicerad 2011-05-08. Hämtad 2014-06-21.

[20] R. Fielding, *Architectural Styles and the Design of Network-based Software Architecture*. Irvine: University of California, 2000.

[21] T. Erl *et al*, *SOA with REST*. Upper Saddle River, New Jersey: Prentice Hall, 2011.

[22] J. Webber, S. Parastatidis & I. Robinson, *REST in Practice: Hypermedia and Systems Architecture*. Sebastopol, Kalifornien: O'Reilly Media, 2010.

[23] W3C, ”Method definitions”, http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9. Publicerad 2004-09-01. Hämtad 2014-08-12.

[24] W3C, ”Hypertext Transfer Protocol”, http://www.w3.org/Protocols/rfc2616/rfc2616.html. Publicerad 2004-09-1. Hämtad 2014-08-12.

[25] Oxford Dictionaries, ”Hypermedia”, http://www.oxforddictionaries.com/definition/english/hypermedia. Pulicerad 2014. Hämtad 2014-06-25.

[26] Oxford Dictionaries, ”Hypertext”, http://www.oxforddictionaries.com/definition/english/hypertext. Publicerad 2014. Hämtad 2014-06-25.

[27] R. Fielding, ”REST APIs must be hypertext driven”, http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. Publicerad 2008-10-20. Hämtad 2014-06-25.

[28] C. Pautasso, *BPMN for REST*. Lugano: Luganos Universitet, 2011.

[29] L. Richardson, ”Justice will take us millions of intricate moves”, http://www.crummy.com/writing/speaking/2008-Qcon/act3.html. Publicerad 2009-01-16. Hämtad 2014-06-30.

[30] OMG Group, 2011, ”

[31] Oracle, ”VirtualBox”, https://www.virtualbox.org/. Hämtad 2014-08-06.

[32] Oracle, ”Bridged networking”, *Oracle VM Virtual Box, User Manual*. https://www.virtualbox.org/manual/ch06.html#network_bridged. Hämtad 2014-08-07.

[33] Wikipedia, ”Virtual loopback interface”, http://en.wikipedia.org/wiki/Loopback#Virtual_loopback_interface. Hämtad 2014-08-07.

[34] M. Fowler, ”Richardson Maturity Model – steps towards the glory of REST”. http://martinfowler.com/articles/richardsonMaturityModel.html. Publicerad 2010-03-18. Hämtad 2014-06-30.

[35] K. Beck m fl. ”Principles behind the Agile Manifesto”, http://agilemanifesto.org/principles.html. Publicerad 2007-08-07. Hämtad 2014-06-30.

[36] K.S. Rubin. *Essential Scrum – a practical guide to the most popular agile process.* Upper Saddle River, New Jersey: Addison-Wesley.

[37] M. Hammarberg & J. Sundèn. *Kanban in Action.* Shelter Island, New York: Manning Publications Co.

[38] Oracle, ”JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services”, https://jcp.org/en/jsr/detail?id=339. Hämtad 2014-07-03.

[39] Oracle. ”Jersey: RESTful Web Services in Java”, https://jersey.java.net. Hämtad 2014-07-04.

[40] W3C, ”The link element”, http://www.w3.org/TR/html5/document-metadata.html#the-link-element. Publicerad 2014-07-31. Hämtad 2014-08-04.

[41] Oracle, ”JSR 222: Java Architecture for XML Binding (JAXB) 2.0”, https://jcp.org/en/jsr/detail?id=222. Hämtad 2014-07-10.

[42] Activiti, ”Message Start Event”, http://activiti.org/userguide/#bpmn-MessageStartEvent. Hämtad 2014-08-05.

[43] Wikipedia, ”Microformat”, http://en.wikipedia.org/wiki/Microformat. Publicerad 2014-08-27. Hämtad 2014-09-09.

[44] Wikipedia, ”Publish-subscribe pattern”, http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern. Publicerad 2014-09-06. Hämtad 2014-09-09.

Bilaga A: Dokumentation av egenutvecklad programkod

ProcessService

ProcessService håller tre metoder som är av större intresse för lösningsförslaget: `startProcessByMessage()`, `updateProcessByMessage()` samt `startProcessByKey()`. Den första exponeras på URL:en `/processes` och är den resurs till vilken en klient skickar en representation för att skapa en instans av en process som lyssnar på ett namgivet meddelande ("start" i fallet Process B). Den andra exponeras på `/processes/{id}` och hanterar meddelanden för pågående processinstanser där `{id}` anger ID:t för en given processinstans. Den tredje befinner sig på /processes/key/{key} där {key} anger nyckeln[2] på den process som ska instansiers. Resursen är enbart till för att skapa en processintstans av Process A och skapa en representation av media typen `application/vnd.motrice+xml` innehållande det meddelande som ska skickas samt en callback i form av ett `<link>` element.

För alla tre metoderna gäller att den enda giltiga media typen vid anrop är med att den enda tillåtna media typen är `application/vnd.motrice+xml`. Detta begränsning är en del av tjänstekontraktet och medför att `ProcessService` förväntar sig att alla representationer kan deserialiseras till en objekt av klassen `ProcessRepresentation`. Om det av någon anledning inte skulle gå hanterar ProcessService tre olika undantag som redogörs för i tabellen nedan.

Undantag	HTTP kod	Beskrivning
ActivitiException	400	Det finns ingen processdefinition/instans som lyssnar på det namngivna meddelandet.
ActivitiObjectNotFound Exception	404	Det finns ingen processinstans med det angivna ID:t.
MediaTypeException	422	Representationen innehåller okända XML-element.
Exception	500	Något annat fel uppstod vid runtime

Observera att även Jersey i sig kan svara med felmeddelanden om klienten anger en felaktig media typ i sitt anrop, använder sig av en ogiltig metod för en specifik URL etc.

2 I Activiti är en *nyckel* ett unikt namn för en processdefinition som alltid pekar på den senaste versionen av processen.

`startProcessByMessage()` förväntar sig att representationen innehåller ett namngivet meddelande och undersöker även representationen efter ett eventuellt `<link>` element som håller den URI till vilket ett meddelande önskas vid ett senare tillfälle. I lösningsförslaget skickas alltid en sådan länk med, men koden förutsätter inte att elementet finns. Om `<link>` elementet finns så skapas ett nytt objekt av klassen ProcessRepresentation där värdet på attributet `data-message` blir meddelandet som representationen håller. Resterande attribut för `<link>` elementet används som processvariabler för att skapa ett HTTP anrop genom klassen `MessageTaskDelegate` vid ett senare tillfälle.

`updateProcessByMessage()` innehåller kod för att korrelera en given processinstans med en pågående exekvering. Varje processinstans som skapas av Activiti skapar i sin tur en eller flera exekveringar som visar var processen befinner sig. Eftersom Activiti kräver att ett meddelande som ska tas emot av en pågående process skickas till en specifik exekvering så är det inte möjligt att enbart använda sig av processinstans ID:t för att leverera meddelandet. Koden anropar därför Activitis runtime och frågar efter den exekvering som är barn till den processinstans med det angivna ID:t *och* som lyssnar efter det angivna meddelandet.

`startProcessByKey()` är en hjälpfunktion för att skapa en instans av Process A och utifrån skapa den information som extraheras från processinstansen skapa en representation innehållande ett namngivet meddelande och en callback. För att skapa en callback krävs en lösning som arbetar runt Activitis implementation för att exekvera processer: anrop som instansierar processer är blockerande och kommer omedelbart att påbörja exekveringen av processen. Exekveringen överförs till klienten först när processinstansen når en punkt som kräver interaktion av klienten. Det innebär att processer som enbart innehåller automatiska aktiviteter, det vill säga de som inte kräver någon interaktion från klienten (som t ex anrop av webbtjänster), kommer att blockera klienten tills att hela processen har exekverat klart. En annan betydande konsekvens är att det inte är möjligt att få fram det ID för processinstansens som krävs för att kunna skapa en callback. Den lösning som tillämpas i lösningsförslaget är att `Process A` har en `RecieveTask` aktivitet som sin första aktivitet. Syftet med `ReceiveTask` är egentligen att invänta ett meddelande, men är i sin nuvarande form enbart delvis implementerad. Den egenskap som lösningsförslaget utnyttjar är att `ReceiveTask` gör att processinstansen hamnar i ett väntetillstånd och därmed återför kontrollen till klienten. Med processinstansen i ett väntetillstånd är det möjligt att skapa en representation innehållande en callback utifrån instans ID:t och skicka in representationen som en processvariabel tillsammans med de variabler som konfigurerar det senarekommande HTTP anropet. För att låta processinstansen åter exekvera anropas `signal()` i Activitis `RuntimeService` med ID:t som argument.

Bilaga B: Testmiljö

Följande instruktioner redogör för hur läsaren kan konfigurera en testmiljö lik den som har använts för att verifiera lösningsförslaget.

Hård- och mjukvarukrav

Den hårdvara som används bör bestå av minst en tvåkärnig processor med 2 Gb minne. Operativsystemet bör vara av Ubuntu 12.04 eller senare utrustad med Java JDK 1.7, Apache Maven 3.0.5, Git 1.9.1 samt SQL databasen PostgreSQL 9.2 eller 9.3. Någon applikation som kan generera HTTP anrop är även nödvändig. För den som är van vid kommandoraden är Curl (curl.haxx.se) ett utmärkt alternativ, för den som är van vid grafiska användargränssnitt är SoapUI en bra, men något mer komplext, alternativ.

Installation av Motrice

Klona det utvecklingsträd av Motrice som har används för examensarbetet och placera `motrice.properties` så att Motrice kan läsa den vid kompilering:

```
git clone https://bjho0801@bitbucket.org/bjho0801/exarbete-motrice.git
sudo mkdir /usr/local/etc/motrice
sudo Cp exarbete-motrice/motrice.properties /usr/local/etc/motrice
```

Skapa databasanvändare, själva databasen och fyll den med data:

```
cd exarbete-motrice/database/bin
sudo sh -c "./create_user.sh"
sudo sh -c "./create_database.sh"
sudo sh -c "./create_tables.sh"
sudo sh -c "./execute_file.sh motricedb
../create/motrice.postgres.basedata.sql"
```

Kompilera Motrice och placera lösningsförslaget (restrice.war) så att det deployas när Motrice startas:

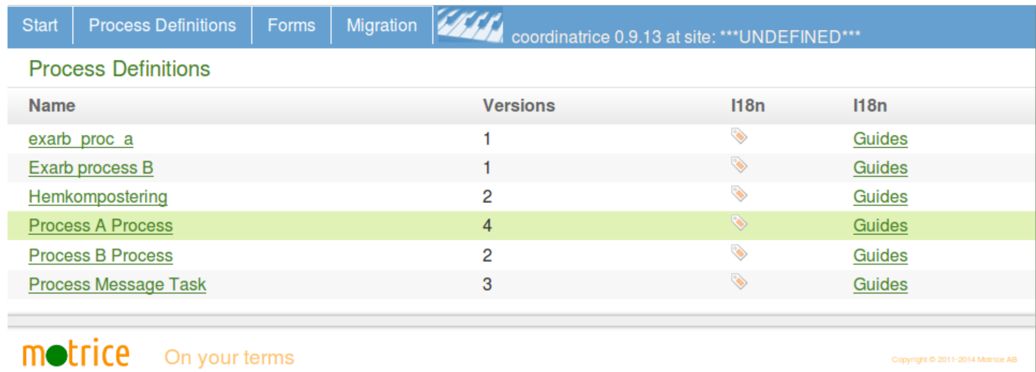
```
cd exarbete-motrice/
mvn clean install
cp exarbete-motrice/inherit-service/inherit-service-rest-
server/target/restrice.war exarbete-motrice/inherit-
portal/target/tomcat7x/webapps/
```

Starta Motrice:

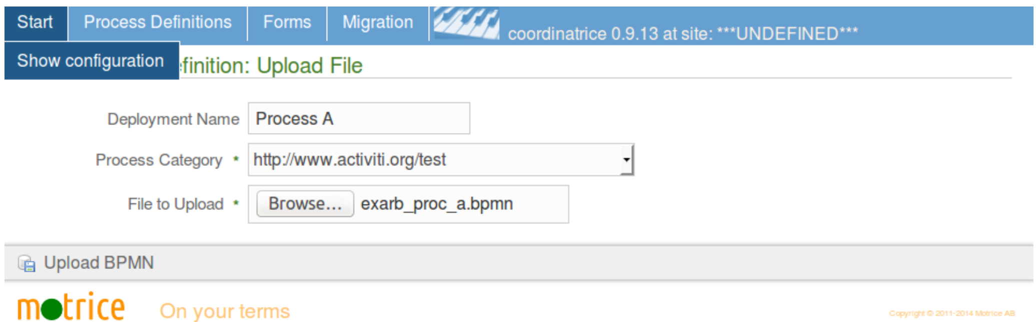
```
cd exarbete-motrice/inherit-portal
mvn -P cargo.run
```

Administration och installation av processdefinitioner

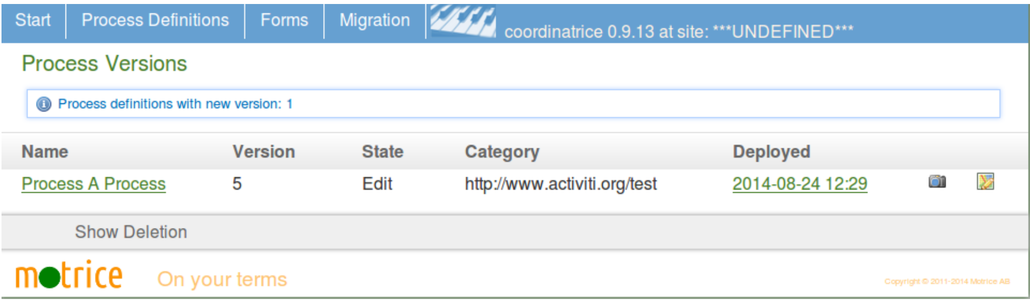
Administrationsgränssnittet för Motrice, Coordinatrice, finns på `localhost:8080/coordinatrice` och ser ut ungefär som i bilden nedan (notera att vid en helt ny installation finns det inga processdefinitioner listade):



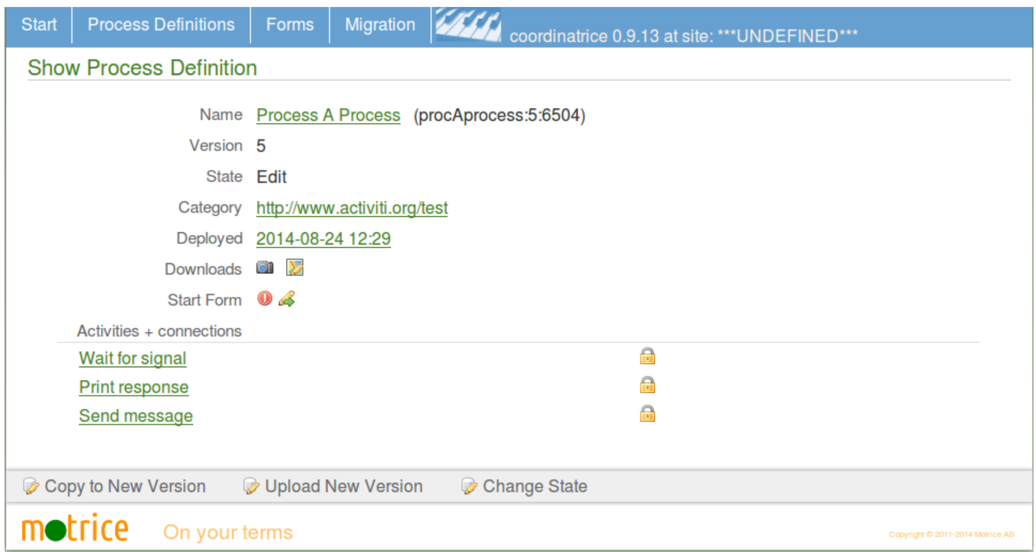
För att ladda upp de processdefinitioner som används i arbetet går man till `process definitions → upload BPMN` och väljer `file to upload`. BPMN definitionerna för arbetet heter `exarb_proc_a.bpmn` och `exarb_proc_b.bpmn` och finns under `/exarbete-motrice/bpm-processes`.



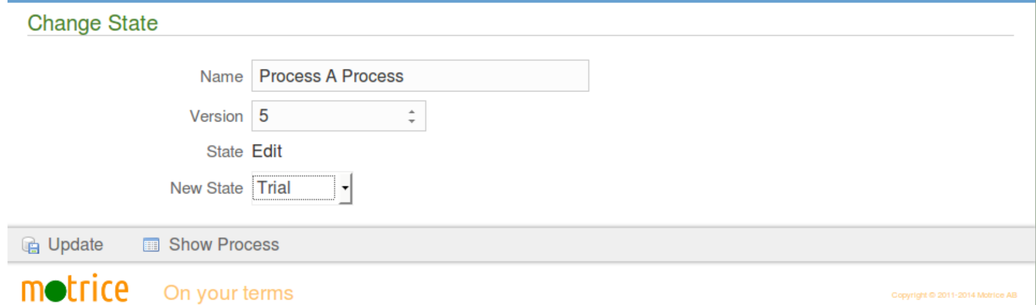
Välj Upload BPMN för att ladda upp processdefinitionen. Följande sida visar sig:



Tryck på namnet för processdefinitionen (exempelvis Process A Process i bilden ovan) för att gå till detaljvyn för processen. Vyn är likartad den i bilden nedan:



För att gör en processdefinition exekverbar behöver definitiones tillstånd ändras till publicerad. Gör detta genom att välja `change state`.



Ändra processens tillstånd genom den drop-down meny som finns till höger om rubriken new state. Välj `approved` och klicka på `update`. När processend tillstånd har uppdateras hamnar man åter i listan över processdefinitioner. Gå tillbaka till detaljvyn och navigera framåt tills att vyn `change state` syns igen. Välj `published` och klicka på `update`. Processen är nu exekverbar och åtkomlig genom REST gränssnittet.

Test av lösningsförslag

Förutsatt att båda processdefinitionerna finns i samma Motriceinstans så kan följande kommando med Curl användas för att påbörja exekveringen av process A.

```
curl -X POST
http://localhost:8080/restrice/jersey/runtime/processes/key/procAprocess/
```

Anropet till `../processes/key/procAprocess` routas till en specifik funktion i ProcessService, `startProcessByKey()`, som innehåller kod vilken konfigurerar och påbörjar exekveringen av process A. Koden är relativt enkel att ändra för de fall där process B befinner sig i en annan instans av Motrice. Följande kodparti visar den relevanta delen av funktionen:

```
Pi = engine.getActivitiEngineService()
.getEngine().getRuntimeService()
.startProcessInstanceByKey(key);

ProcessRepresentation pr =
       new ProcessRepresentation("start",
              new Link.Builder("message",
                     uriInfo.getBaseUriBuilder()
                     .path(RuntimeService.class)
                     .path("processes")
                     .path(pi.getId())
                     .build())
              .mediaType(Representation.MOTRICE_MEDIA_TYPE)
              .method("put")
              .message("update")
              .build());

Map<String, Object> vars = new HashMap<String, Object>();

/*
URI uri = new URI("http://loclhost:8080/restrice/
                   jersey/runtime/processes/");
*/
URI uri = uriInfo.getBaseUriBuilder()
                          .path(RuntimeService.class)
                          .path("processes")
                          .build());

vars.put("targetURI", uri);
vars.put("httpMethod", "POST");
vars.put("mediaType", Representation.MOTRICE_MEDIA_TYPE);
vars.put("representation", pr.toString());


engine.getActivitiEngineService().getEngine().getRuntimeService()
.setVariables(pi.getId(), vars);

engine.getActivitiEngineService().getEngine().getRuntimeService()
.signal(pi.getId());
```

Processvariabeln targetURI anger den URI där process B befinner sig. Om process B befinner sig i en virtualiserad Motrice instans är går det att ändra localhost i det utkommenterade kodavsnittet, ange URI:n för den virtualiserade instansen och använda . Notera att

För att kontrollera att anropen och svaren går som förväntat kan loggfilen `/ex-arbete-motrice/inherit-portal/target/logs-cargo/container.log` avläsas.