XML-sprog til diagramgenerering via XSLT fra en webservice

doktor@dyregod.dk — **Ulf Holm Nielsen**
tnjr@ruc.dk — **Thomas Riisbjerg**
mads@danquah.dk — **Mads Danquah**

Vejleder:

hniss@itu.dk — **Henning Niss**

18. december 2003

IT Universitetet
Internet og Software Teknologi
4-ugers projekt

Abstract

Denne rapport omhandler udviklingen af en webservice, der tilbyder transformering af et domænespecifikt XML sprog til forskellige diagrammer, repræsenteret i SVG. Projektet er udarbejdet i samarbejde med IBM Zurich Research Lab med det formål, at repræsentere og visualisere resultater fra et passagersimuleringssystem udarbejdet hos IBM. Følgende diagramtyper er understøttet: lagkagediagram, søjlediagram samt punktdiagram. Transformeringen er foretaget med XSLT, hvor hver diagramtype har sin egen XSL stylesheet. Transformeringen fungerer efter hensigten.

Nøgleord: XML SVG PDF iText Stylesheet XSL XSLT Webservices SOAP.

Indhold

Forord

Denne rapport er en del af et 4-ugers projekt på Internet og Software Teknologi linien ved IT Universitetet i København. Projektet er blevet til i samarbejde med IBM Zürich Research Laboratory i Schweiz, hvor projektgruppen har arbejdet i fire uger på PaxFlow projektet. Rapporten er udarbejdet af Ulf Holm Nielsen, Mads Danquah og Thomas Riisbjerg.

En elektronisk udgave kan hentes på http://www.napalminthemorning.dk/svg/rapport.pdf

Tak til Richard Bödi, Markus Kunz, Christopher Giblin og Ulrich Schimpel for værdifuldt input i løbet af projektet. Tak til Henning Niss for god vejledning. Tak til Linda for korrektur og generel overbærenhed. Tak Lucas Heusler og Douglas Dykeman for at gøre opholdet ved IBM Research i Zürich muligt. Og ikke mindst tak til alle i kantinen.

Læsevejledning

Indledning giver en kort introduktion til hvad der skal laves og i hvilken sammenhæng. Dette munder ud i en indledende kravspecifikation.

Analyse introducerer de problemstillinger der skal behandles i forbindelse med projektet, og hvordan de tænkes grebet an. Afsnittet slutter med en udvidet kravspecifikation der danner basis for design og implementeringen.

Design afsnittet forklarer de overordnede ideer bag designet af systemet og hvordan det passer ind i PaxFlow.

Implementering gennemgår hvordan XML-transformationer, webservices og resten af koden er valgt implementeret.

Afprøvning vil gennemgå hvordan de enkelte dele af løsningen er valgt afprøvet og hvad resultatet af disse var.

Diskussion gennemgår løsningen og diskuterer den i forhold til kravspecifikationen.

Konklusion vil kort opsummere resultatet af projektet.

Perspektivering beskriver hvad der er de naturlige næste skridt.

1 Indledning

1.1 Introduktion til PaxFlow

PaxFlow er en simulator til at forudsige passagerbevægelser i lufthavne op til 7 dage i forvejen. Ved at benytte informationer fra flyselskabernes fælles bookingssystemer om alle fly til og fra en lufthavn simuleres det forventede antal passagerers bevægelse gennem lufthavnen. Undervejs omsamles data fra simuleringen omkring antal passagerer i hvert område på et givent tidspunkt, hvor lang tid der bliver ventet i kø, hvor lang tid der er til shopping m.m. Derved vil det være muligt for f.eks. lufthavnsmyndighederne at identificere mulige flaskehalse for passagerer op til 7 dage i forvejen. Selve data fra bookingssystemerne som indgår i simuleringerne er at betragte som fortrolige og må under ingen omstændigheder videregives til tredje part.

Simulatoren forestilles afviklet automatisk hver morgen, hvorefter brugerne kan få tilsendt rapporter eller selv se disse på en webside. En rapport består af lidt beskrivende tekst og grafer og tabeller til at præsentere resultaterne af simuleringen. Derved forventes det, at værktøjet kan indgå i den daglige planlægning af aktiviteter, bemanding etc.

I forbindelse med præsentation af en rapport om f.eks. hvor mange japanske mænd, der fløj med business class og tilfældigvis befandt sig i Terminal A, kræves det, at disse kan præsenteres på en fleksibel og flot måde. Og da ikke alle potentielle brugere af systemet har adgang til computere med internetforbindelse i arbejdssammenhæng, er det også nødvendigt at rapporterne kan udskrives på printer eller via fax.

Lufthavnsmyndighederne vil stå for den daglige drift med oprettelse af brugere, adgang til rapporter samt input af data til simuleringskørselen. Dette gøres fra en applikation, hvorfra det er muligt at styre hele systemet. Fra denne applikation er det også muligt at se rapporterne.

I forbindelse med PaxFlow ønsker IBM udviklet et fleksibelt system til præsentation af resultaterne af simuleringen til både GUI, webside og print.

1.2 Indledende kravspecifikation

Vi beskriver her den indledende kravspecifikation for vores implementering. Den indledende kravspecifikationen er udarbejdet i samarbejde med IBM Zürich Research Laboratory.

Der ønskes implementeret et system, der på forespørgsel kan generere rapporter indeholdende tekst og grafer. Forespørgslen vil indeholde beskrivelse af tekst, grafer, samt de data, der vil være nødvendig for at generere disse. Rapporterne skal ud fra den samme beskrivelse kunne præsenteres på tre måder.

- Som en web-side
- I en grafisk brugergrænseflade
- Som en pdf-fil

Rapportgenereringen ønskes implementeret som en webservice, der kan levere data til den grafiske brugergrænseflade, til webklienter, samt producere pdf-filer. Der stilles ingen krav til dataformatet, der specificerer rapporterne.

Grafer skal kunne genereres ud fra en kombineret data- og grafbeskrivelse. En beskrivelse kunne f.eks. være “generer et lagkagediagram ud fra følgende data, a=1, b=3, c=3, d=7”. Der stilles ingen krav til dataformatet.

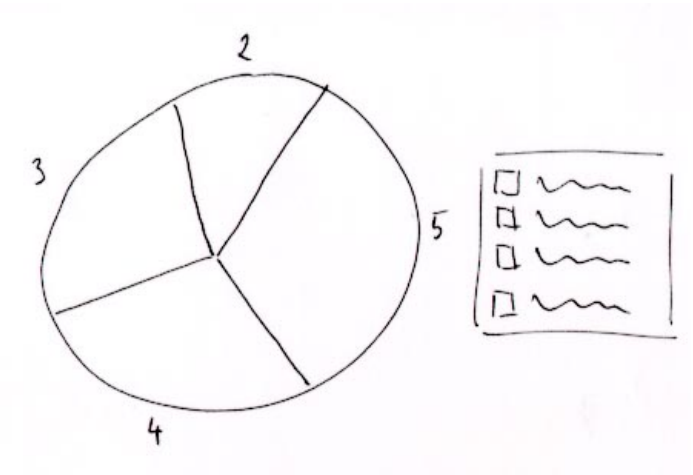
Der skal kunne genereres følgende grafer:

- Lagkagediagrammer
- Søjlediagrammer, enkelt såvel som stablet
- 2D plots (linje såvel som punkt) med understøttelse for sekundære X og Y-akser.

Diagrammerne skal kunne indeholde en signaturforklaring. Akserne skal kunne være logaritmiske.

Der stilles ingen krav til hvilket grafikformat, der anvendes, så længe det kan indgå i de tre former for rapportpræsentationer.

Da denne rapport er del af et 4-ugers projekt, og vi derfor har begrænset tid, vil vi kun beskrive en delmængde af den løsning, der bliver implementeret til IBM. Helt konkret beskriver vi overvejelser og implementering af SVG-genereringen samt implementeringen af denne som en webservice. Den videre analyse af kravspecifikationen vil derfor hovedsageligt koncentrere sig om de dele, der har med grafgenerering at gøre.



Figur 1: Mockup af et lagkagediagram

2 Analyse af problemstillinger

I de følgende afsnit vil problemstillingerne i kravspecifikationen blive analyseret, og der vil blive argumenteret for forskellige måder at løse dem på.

2.1 Grafikformatet

Graferne består af tekst samt geometriske elementer såsom linjer, cirkelbuer og punkter. Graferne bør derfor så vidt muligt repræsenteres i et vektorformat, der kan bibeholde denne information i modsætning til f.eks. et bitmapformat. Samtidigt er det også vigtigt, at formatet kan bruges i alle de præsentationsformer rapportgeneratoren understøttet, dvs. som en web-side med tilhørende grafik, i en fed klient og som en PDF-fil.

Den grafiske brugergrænseflade bliver implementeret i forbindelse med projektet. Man kan derfor som udgangspunkt antage, at den vil kunne fremvise ethvert format, der skulle blive valgt.

PDF er et vektorbaseret format, der kan indeholde tekst såvel som grafik. PDF understøtter inkludering af en række grafikformater, men ingen af disse er vektoriseret. Hvis man ønsker at inkludere vektoriseret grafik i et PDF dokument, skal grafikken optegnes med PDF-kommandoer. Det vil derfor være et krav til det grafikformat der vælges, at det kan omformes til PDF der efterfølgende kan inkluderes i den PDF baserede rapport.

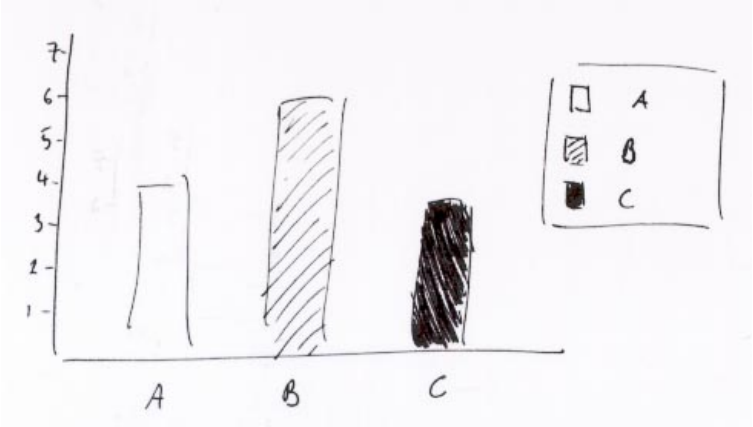
Repræsentationen som webside, kræver at klienten understøtter det format, der bliver valgt. Formatet skal derfor så vidt muligt være et anerkendt format der understøttes i hovedparten af web-browsere.

Det valgte grafikformat er Scalable Vector Graphic (SVG)[13], der er en anbefalet standard af World Wide Web Consortium (W3C). SVG er et vektorbaseret grafikformat, der er beskrevet i XML. Det understøtter alle tegneoperationer, der vil være nødvendige for at optegne graferne.

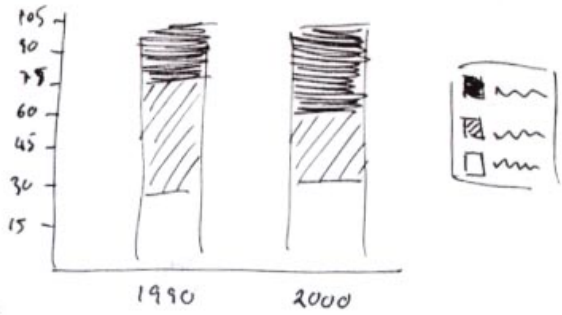
Til fremvisning af SVG i den fede klient bruges Apache projektets Batik SVG Toolkit[3]. Batik kan optegne SVG som PDF og kan derfor også bruges til at inkludere SVG-grafik i PDF-filer. Understøttelsen for SVG i de førende webbrowsere er acceptabel nok til, at det kan antages, at klienten kan fremvise SVG-filer.

2.2 Diagrammer

Ifølge kravspecifikationen skal diagrammerne kunne genereres ud fra en kombineret data



Figur 2: Mockup af et søjlediagram



Figur 3: Mockup af et stablet søjlediagram

og diagrambeskrivelse. De forskellige former for diagrammer har forskellige formål og præsenterer forskellig former for data. Der gives derfor en gennemgang af de forskellige diagramtyper, hvorefter kravene til beskrivelsen udformes. Jfr. [10] for nærmere beskrivelse af diagrammer.

Analysen og beskrivelsen af graferne tager udgangspunkt i anbefalingerne givet i [10].

Lagkagediagram

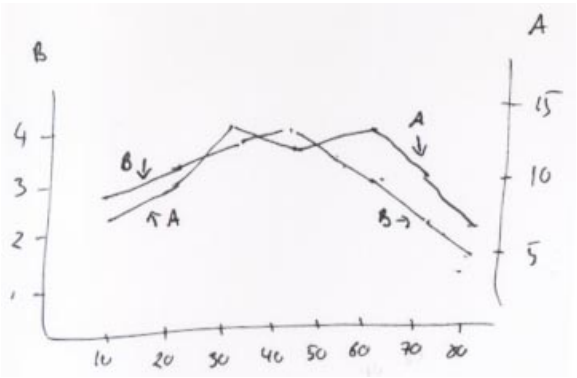
Et lagkagediagram bruges til at give et overblik over forholdet imellem få værdier. Da det er svært at se den præcise størrelse på et cirkeludsnit, bør et lagkagediagram ikke bruges til præcise sammenligninger. De er derimod gode til at give et hurtigt overblik over forhold. Et lagkagediagram repræsenterer endimensionelle data, og vil typisk være et billede af forholdet imellem en række variable på et givet tidspunkt.

De enkelte udsnit i lagkagen bør være farvet således, at de er let adskillige fra hinanden, farverne bør samtidig være "rolige" nok til, at de ikke giver falske illusioner af udsnittets størrelse. Eeks. kan en kraftig rød farve få et udsnit til at virke større end et tilsvarende udsnit med en neutral grå farve.

Ud over selve lagkagen bør de enkelte udsnit markeres med den værdi, de repræsenterer.

Søjlediagram

Et søjlediagram bruges til at sammenligne en række værdier. Søjlediagrammer er ligesom lagkagediagrammer velegnede til at give et overblik over få værdier i forhold til hinanden. Et søjlediagram repræsenterer endimensionelle data og vil typisk være et billede af forholdet imellem en række variable på et givet tidspunkt.



Figur 4: Mockup af et punktdiagram

Ved at stable søjler oven på hinanden kan søjlediagrammer være meget nyttige til at give et billede af, hvordan sæt af relaterede variabler forholder sig til hinanden. Det kunne f.eks. være en præsentation af forholdet imellem antallet af folk i et område der 1: venter i kø, 2: bliver betjent 3: laver ingenting til to forskellige tider. Se evt figur 3

Søjlediagrammet præsenteres med to akser, hvor de egentlige værdier ligger op ad y-aksen, og navnene på værdierne hen ad x-aksen. De enkelte søjler gives farver således, at de er let at kende fra hinanden. Hvis søjlerne er stablede, gives de farver efter deres vertikale position.

Punktdiagram

Et punktdiagram bruges til at give overblik over en eller flere større serier af todimensionelle data. Det vil som oftest være en variabels udvikling over tid. De enkelte punkter kan forbindes med linjer for at interpolere data imellem punkter. For at sammenligne data, der ikke direkte er sammenknyttet, kan der introduceres sekundære x- og y-akser.

Data præsenteres som en række af liner, der går igennem de enkelte datapunkter. Linjerne farves således, at linjer fra forskellige dataserier kan skelnes fra hinanden. Akserne er som regel linieære, men skal også kunne være logaritmiske.

Det er muligt at kombinere flere punktdiagrammer i ét diagram.

Diagrambeskrivelsen

Diagrammerne præsenterer overordnet set to former for data

- Lagkage- og søjlediagrammer præsenterer en enkelt serie værdier, der hver er tilknyttet et navn.
- Punktdiagrammer præsenterer en eller flere serier af todimensionelle data, hvor hver serie har tilknyttet et navn.

Punkt- og søjlediagrammer indeholder begge akser. Søjlediagrammerne indeholder en enkelt x og y-akse, mens 2D plots kan indeholde op til to x og y-akser. Da der derfor ikke umiddelbart er en implicit måde at afgøre, hvilken akse data skal tegnes ud fra, skal denne sammenhæng udtrykkes eksplicit i diagrambeskrivelsen.

Den kombinerede diagram- og databeskrivelse skal indeholde

- udtryk, der kan beskrive hvilken form for diagram, der skal tegnes, samt kunne tillade flere punktdiagrammer.
- udtryk, der kan beskrive serier af en- og todimensionelle data, samt navne til disse
- udtryk, der beskriver sammenhængen imellem data og en eller flere akser.
- udtryk, der konfigurerer den valgte diagramtype
- udtryk, der kan samle flere punktdiagrammer

Da SVG er XML-baseret vil det være fordelagtigt at lade diagrambeskrivelsen være XML-baseret, da dette åbner en række muligheder. Først og fremmest vil det være muligt at generere SVG dokumentet via en transformation, men det vil også lette transporten af diagrambeskrivelsen imellem klienten og web servicen.

De følgende afsnit vil uddybe disse muligheder.

2.3 Transformering

Diagrammerne skal genereres ud fra et udtryk i sproget, der indeholder en beskrivelse af diagrammet samt tilhørende data. Det resulterende SVG dokument beskriver ligeledes diagrammet og dennes data. Hvor diagrambeskrivelsen beskriver diagram og data generelt, beskriver SVG specifikt, hvordan diagrammet skal optegnes.

Genereringen af et givent diagram ud fra et udtryk i sproget kan ses som en *transformering* fra udtrykket i diagrambeskrivelsen til et tilsvarende udtryk i SVG.

Da transformeringen er et spørgsmål om at indlæse data, manipulere det og udskrive det igen, kan selve transformeringen implementeres i de fleste programmeringssprog. Da diagrambeskrivelsen er XML-baseret vil det være en fordel at benytte sig af et sprog, der kan drage nytte af dette.

Java har god understøttelse for manipulation af XML-data via blandt andet DOM. Da Java i forvejen bruges som implementerings sprog i de resten af løsningen, bør et valg af Java som implementerings sprog mindske interaktions-problemerne med det resterende system. Det umiddelbare problem ved at vælge Java er, at det på trods af sin XML-understøttelse er et generelt sprog. En implementering vil uundgåeligt komme til at skulle udtrykkes i meget generelle termer.

Alternativet til at udtrykke transformeringerne i et generelt sprog er at udtrykke dem i et sprog, der specifikt er udformet til at udtrykke transformeringer. Det XML-baserede sprog XSLT udformet af W3C[24] er et sådant sprog. XSLT er oprindelig udformet som en del af Extensible Stylesheet Language(XSL)[7] beregnet til transformeringer, fra data specificeret i et XML-format til XML Formatting Objects[7][24]. XSLT beskriver en række transformeringer der påføres på et *kildetræ* og derved danner et *resultattræ*. På trods af at XSLT ikke er ment som et generelt XML-transformerings sprog[24], kan det ikke desto mindre bruges til at transformere imellem alle former for XML-baserede dokumenter. Da XSLT er et sprog, der udelukkende beskriver transformationer, beskæftiger det sig ikke med, hvordan transformationerne i praksis udføres. En implementering i Java ville eksplicit traversere DOM-strukturen, og sideløbende opbygge, indsætte og fjerne elementer i SVG-dokumentet. XSLT ligger her mere op til, at traverseringen foregår implicit, og at resultatet, i dette tilfælde SVG-dokumentet, er et resultat af traverseringen. Til at udføre de egentlige transformationer bruges en XSLT-processor.

Da XML såvel som XSLT er defineret af W3C, og da XSLT er W3C's forslag til hvordan XML-transformeringer skal beskrives, vælges XSLT til at beskrive transformeringerne i. Selve transformationen udføres af XSLT-processoren Xalan [21].

2.4 Validering af sproget

Følgende er et kort overblik over forskellige valideringssprog med deres svagheder og styrker i forhold til vores behov for diagram- og rapportbeskrivelsen. Dette er ikke ment som en udtømmende gennemgang af de forskellige muligheder, men blot en kort opsummering der forklarer hvilke faktorer, der lå til grund for vores valg af valideringssprog.

For at sikre, at diagram- og rapportbeskrivelserne indeholder gyldige beskrivelser er det nødvendigt at validere dem. Der eksisterer en række metoder til at validere XML-dokumenter, hvor nogle af de mest populære er DTD[6], XML Schema[23], Relax NG[12] samt DSD 2.0[5]. DTD og XML Schema er anbefalet fra W3C, hvor Relax NG og DSD 2.0 er selvstændige teknologier. Relax NG er udviklet af OASIS[9], hvor DSD 2.0 er udviklet af BRICS[2].

DTD

DTD er det ældste valideringssprog, der blev defineret samtidigt med XML 1.0. For at en parser skal opfylde den fulde XML specifikation, er det er krav at den skal implementere DTD. I praksis understøtter alle større XML parsre altså DTD. Dog har DTD har en lang række mangler som senere valideringssprog har løst. En af de større mangler ved DTD er, at det har et begrænset datatypesystem, hvor det kun er muligt at definere simple datatyper for attributter. Elementers indhold har ikke nogen defineret datatype, men behandles blot som tekst eller underelementer.

Desuden har DTD ringe understøttelse for at validere referentiel integritet mellem elementer i et dokument. Det er muligt at kræve, at hvis ét element har en `IDREF` attribut med en given værdi, så skal der eksistere et andet element med en `ID` attribut med samme værdi i dokumentet. Dog er det ikke muligt at begrænse, hvilken type af element en `IDREF`-attribut skal pege på. En `IDREF` attribut kan altså referere til et vilkårligt element, hvis elementet blot har en `ID` attribut med den samme værdi. Sidst, men ikke mindst, er DTD ikke selv defineret i XML. Det har ikke nogen direkte betydning, men vidner om, at DTD ikke formår at validere sig selv.

XML Schema

Efterhånden som XML 1.0 fik større anvendelse blev det klart, at der var brug for en afløser til DTD. Efter mange forskellige forslag blev XML Schema vedtaget som den næste anbefaling til valideringssprog fra W3C. I modsætning til DTD er XML Schema selv defineret i XML og har som målsætning at være i stand til at validere sig selv. Eftersom XML Schema er anbefalet af W3C, har det efterhånden fået en bred understøttelse i de fleste XML parsere.

I XML Schema er det muligt at definere komplekse datatyper for såvel elementer som attributter. Det er også muligt at definere unikke nøgler i elementer, som andre elementer kan referere til. Således sikres, at en given `IDREF` attribut altid peger på et element af den rette type.

Desværre har XML Schema stadig nogle begrænsninger for, hvad det er muligt at definere i en XML grammatik. Eksempelvis er det ikke muligt at definere, at hvis en instans af et element har en attribut A, så må den ikke have en attribut B. Ligeledes er det ikke muligt at definere, at hvis en attribut A er defineret, så må element X ikke optræde som barn.

Relax NG

Relax NG er et af de mere populære alternativer til XML Schema. Det er en sammensætning af to tidligere valideringssprog, TREX[16] og RELAX[11]. Ligesom XML Schema er Relax NG selv defineret i XML, dog har det også en kompakt form, som kan oversættes til XML repræsentationen. Relax NG har som mål bl.a. at være mere overskueligt for mennesker at læse og skrive samt at kunne udtrykke nogle af de begrænsninger og regler, som XML Schema ikke er i stand til at håndtere. Relax NG har ikke noget typesystem, men kan bruge andre valideringssprog til at håndtere typer, eksempelvis XML Schema.

Desværre har Relax NG heller ikke understøttelse for nøgler af en bestemt type og referencer til disse, på lige fod med DTD. En anden ulempe ved Relax NG er, at det ikke er særlig udbredt. Relax NG er implementeret i en eller anden form i adskillige XML parsere, men det er endnu ikke så velunderstøttet som XML Schema.

DSD 2.0

DSD 2.0 er et forholdsvist nyt valideringssprog fra BRICS, der, ligesom Relax NG, har som mål at være nemt at håndtere for mennesker og samtidigt være mere udtryksfuldt i forhold til XML Schema. DSD 2.0 understøtter typer for elementer og begrænsninger på værdier

ligesom XML Schema. Desuden understøtter DSD 2.0 også nøgler og referencer på lige fod med XML Schema.

Den største ulempe ved DSD 2.0 er, at det stadig er så nyt at det ikke er understøttet af ret mange XML parsere. Desuden er dokumentationen for DSD 2.0 stadig mangelfuld, men det er sandsynligvis blot et tegn på hvor ungt sproget er.

Valg af valideringssprog

Til de ovennævnte valideringssprog stiller vi følgende krav:

- Vi har brug for at knytte nogle elementer til andre på tværs af elementheirarkiet. Flere diagrammer kan i nogle tilfælde have brug for at referere til samme akse eller data.
- Det skal være muligt at begrænse indholdet af elementer til en given datatype, eksempelvis reelle tal eller gyldige referencer til andre elementer. Ved at begrænse datatyperne sikres at diagrammerne opbygges fra meningsfyldt data.
- Vi vil så vidt muligt undgå at binde os til én bestemt XML parser, derfor vægtes bred understøttelse frem for et fleksibelt sprog.

Ud fra de overstående krav, står valget mellem XML Schema og DSD 2.0. DTD er ikke i stand til at definere typer for elementer eller begrænse de elementer en reference må pege på. Relax NG er i stand til at begrænse typen på elementer, ved at bruge eksempelvis XML Schema, men understøtter heller ikke begrænsninger på referencer. DSD 2.0 er i stand til at udtrykke det samme som XML Schema og mere til, med en simplere syntaks. Havde DSD 2.0 været mere udbredt og haft lidt mere dokumentation havde det været et oplagt valg som validerignssprog. Dog er XML Schema i stand til at opfylde de krav vi stiller til at valideringssprog, samtidigt med at det nyder stor udbredelse. Eksempelvis er XML Schema det eneste valideringssprog som SOAP kan bruge, hvilket er dækket nærmere i afsnit 2.5. Derfor vælges XML Schema som valideringssprog til diagram- og rapportbeskrivelserne.

2.5 Introduktion til Service-Orienteret Arkitektur og webservices

Følgende afsnit er ment som et kort overblik over Service-Orienteret Arkitektur (SOA) og webservices, ikke som en udtømmende diskussion af hvad der gør en arkitektur til service orienteret eller i detaljer med hvad en webservice er. Bølgerne går stadig højt i W3C's WSA gruppe (Web Services Architecture) om definitionen på en webservice, og denne debat ønsker vi ikke at redegøre for her.

De seneste par år har trenden i it-industrien gået mod mere integration af forskellige applikationer internt i virksomheden eller endog eksternt med leverandører og kunder. Blandt andet på grund af dette skift er der kommet mere fokus på at få disse seperate systemer til at arbejde sammen på tværs af sprog og platforme. En række store virksomheder og organisationer er gået sammen om at udforme standarder for hvordan man kan få alle disse systemer til at snakke sammen; disse standarder kan opfattes generelt som byggeklodserne for webservices. Men bag al snakken om teknikker til udveksling af data ligger et lidt andet syn på applikationers arkitektur, dette er beskrevet som SOA.

SOA er en måde at bygge softwaresystemer ved at konstruere selvstændige services, der benyttes af enten andre services eller en slutbruger, og tilsammen udgør en applikation. SOA er ingen ny opfindelse, faktisk har det eksisteret i mange år, dog under andre navne; f.eks kan det følges tilbage starten af 80'erne med introduktionen af RPC[25] der senere har udviklet sig til distribuerede protokoller og komponent modeller som CORBA, RMI, DCOM og Enterprise Java Beans (EJB). Et eksempel på SOA er f.eks den distribuerede platform Jini.

Komponentbaserede systemer

I et komponentbaseret system grupperes en samling objekter som en blackbox i en komponent med et veldefineret interface, som derefter kan benyttes af andre komponenter eller andre applikationers objekter. Store komplekse applikationer til virksomheder udvikles ofte ved integration af mange forskellige komponenter der sættes sammen til en komplet applikation. Applikationer med samme kompleksitet, men uden tanke på at benytte komponenter, kan hurtigt ende i et uoverskueligt netværk af afhængigheder mellem objekterne. Det skulle gerne afløses af et mindre komplekst netværk mellem komponenterne.

Der findes idag mange teknologier til at hjælpe udviklerne med at udvikle software på en komponent orienteret måde, f.eks Microsoft .NET, Java 2 Enterprise Edition mfl. Her benyttes Enterprise Java Beans og lignende teknologier til at bygge (genbrugelige) komponenter, der senere kan sammensættes til de endelige applikationer.

Forskellen mellem services og komponenter

Ved komponentbaserede systemer skal man kende komponentets interface og dens placering for at benytte komponentet. Derved opnår man afhængigheder til de enkelte komponenter, hvilket også er mere overskueligt i forhold til afhængigheder på tværs af komponenterne. Men disse afhængigheder er i høj grad statiske.

I SOA forsøges denne afhængighed fjernet ved indførelsen af et servicekatalog. Her kan services publicere deres interface hvorefter klienter og andre services kan søge efter services med et interface der passer dem.

Derudover er komponentbaserede systemer og SOA meget nært beslægtede og et komponentbaseret system kan opfattes som en SOA uden servicekataloget.

SOA kan opsummeres som bestående af:

Serviceudbyder der tilbyder en service til omverden. Interfacet til servicen har en offentligt tilgængelig beskrivelse.

Servicekatalog indeholder et register over serviceinterfaces. Service udbydere offentliggør deres services her.

Servicebruger der bruger services vha. et offentligt interface og finder disse i service kataloget.

Værktøjer til understøttelse af SOA

Virksomheder som IBM, Microsoft, BEA har gennem W3C udarbejdet en række specifikationer der skal definere hvordan services kørende på forskellige platforme kan samarbejde i en SOA. Resultatet af dette arbejde er hvad der i medier og mange bøger oftest omtales som webservices.

Webservices er reelt set en SOA med nogle ekstra begrænsninger og er defineret af W3C som[18]: *En softwareapplikation der kan identificeres med en Uniform Resource Identifier (URI) og hvis interfaces er defineret i XML og understøtter direkte interaktion med andes softwareapplikationer vha. XML beskeder til disse interfaces over internet baserede protokoller*. Denne rapport vil gå ud fra denne definition.

Bemærk dog at det også er muligt at bygge SOA baserede applikationer der baserer sig på W3C standarderne ved at benytte Jini og TSpaces. Faktisk kan man bygge sådanne applikationer udfra stort set alle platforme og API'er, hvilket er helt i tråd med ideerne bag. Eksempelvis hvis en virksomhed ønsker at integrere deres lønsystem med et HR intranet. Ved at tilbyde lønsystemet som en webservice vil applikationer på intranettet kunne tilgå denne på samme måde som alle andre webservices. Selv hvis lønsystemet er skrevet i COBOL. Med webservices opnås en fælles måde at tilgå services på tværs af platforme over internettet.

Webservices

W3C har udover at definere begrebet webservices gennem Web Services Architecture (WSA) [18] også foreslået standarder for hvordan XML beskederne og interface beskrivelsessproget ser ud. Derudover har OASIS gruppen specificeret en standard for service katalog. Disse er beskrevet som henholdsvis SOAP (Simple Object Access Protocol)[14], WSDL (Web Service Description Language)[19] og UDDI (Universal Description, Discovery, and Integration)[17].

WSDL er et sprog der abstrakt beskriver services og de operationer, disse tilbyder, samt hvordan man konkret tilgår disse gennem forskellige bindinger. En WSDL fil består af en abstrakt del, hvor servicen defineres sammen med operationer og de beskeder, der benyttes. Beskedernes individuelle dele beskrives inklusiv deres type. Derudover indeholder den en konkret del, der specificerer hvordan disse services kan tilgås vha. af kendte protokoller. En protokol kan f.eks være HTTP eller SOAP

SOAP protokollen beskriver, hvordan man sender beskeder mellem services. I øjeblikket er webservices mere eller mindre lig SOAP services da det stort set er den eneste protokol der for alvor er implementeret. SOAP er en XML-baseret protokol der håndterer udveksling af beskeder, der kan transporteres vha. stort set hvilken som helst anden protokol der bruges på internettet, f.eks XML over HTTP eller SMTP.

SOAP kan bruges på to forskellige måder; Med RPC- eller dokumentbaserede beskeder. Dokument-baserede services specificeres i WSDL hvor beskederne beskrives i XML Schema og det er op til implementeringen eller brugeren af lave beskederne der sendes frem og tilbage. Desværre understøtter WSDL specifikationen ikke andre valideringssprog end XML Schema. RPC-baseret behøver ingen WSDL fil, idet er op til implementationen at serialisere data til og fra XML og minder derved mere om RMI. Det fungerer dog ikke helt så godt som man kunne ønske, da der opstår for mange problemer i forbindelse med samarbejde mellem forskellige implementationer af SOAP. Microsoft understøtter slet ikke denne type SOAP, kun dokumentbaserede beskeder.

UDDI beskriver hvordan centrale servicekataloger beskrives og kommunikerer. UDDI har en centraliseret tilgang til at finde services, de skal findes i servicekataloget. UDDI er ofte selv en webservice. Der findes også en anden måde at opdage services på, nemlig WSIL (Web Services Inspection Language), der er et XML baseret sprog der beskriver hvor en given service kan findes. WSIL filer publiceres ofte for et helt område på en enkelt URI hvor en række services så er beskrevet. WSIL er en decentraliseret måde at publicere services på. Man kan med lidt god vilje sammenligne UDDI med en søgemaskine i traditionel "webforstand" og WSIL som et "sitemap". Et WSIL dokument kan også henvise til forskellige UDDI services.

WSDL og WSIF

W3C's forslag til WSDL indeholder mulighed for at binde operationer til SOAP eller HTTP. Dette kan udvides hvis man mener man har behov for dette. F.eks findes der udvidelser til at håndtere transmission af binære filer sammen med et SOAP request/response[20] i enten MIME eller DIME[4] format. MIME understøttes bredt af mange implementationer, undtagen Microsofts .NET der kun understøtter DIME.

De fleste API'er til at arbejde med webservices understøtter kun SOAP som protokol til beskeder, i enkelte tilfælde understøttes HTTP dog også. Derfor har IBM lavet Web Services Invocation Framework (WSIF)[26], der nu er doneret til Apache projektet. WSIF tilføjer endnu et lag der sigter mod at gøre det transperant for brugeren hvilken binding der benyttes til at kalde webservicen. Derved er det også muligt at skifte binding dynamisk mens programmet kører. Dette kan f.eks være relavant hvis en service flytter fra at være lokalt på maskinen, og derfor kaldes direkte med Java binding, til en anden fysisk maskine og derfor skal kalde med en anden binding. Der findes en mængde bindinger allerede og det er nemt at udvide med egne, p.t. findes der Java, JCA, EJB og JMS.

Ved at udvide WSDL med en Java binding er det f.eks muligt for en service der kører på samme maskine at tilgå denne istedet for f.eks gennem SOAP, hvilket betyder at der bruges langt mindre tid på at kalde en operation defineret af denne service.

Rapport og diagram generering som webservices

PaxFlow systemet er et stort projekt, der udover diagram- og rapportdelene også indeholder et simuleringsmodul implementeret i Arena vha. Siman sproget, moduler til aggregering af simulations resultater og meget mere. Arena er et Microsoft Windows program og kan bl.a. styres vha. et Visual Basic API. Med Microsofts fokus på at overholde standarderne indenfor webservices som SOAP og UDDI ligger det lige for at publicere en webservice til styring af Arena fra Java programmerne.

Diagram- og rapportgenerering er centrale dele i PaxFlow. De benyttes både fra administrations GUI, server, f.eks til at sende rapporter via email, og fra web klienten og bør derfor være tilpasset, så de nemt kan benyttes af disse forskelligartede brugere. For web klientens vedkommende vil en servlet være et oplagt valg, for administrations GUI vil dette ligeledes være en acceptabel løsning, selvom en løsning baseret på RMI sandsynligvis ville være mere naturlig, da RMI frigør udvikleren for at tage stilling til selve kommunikationen. For serverens vedkommende ville det være mest sandsynligt blot at kalde komponenten direkte, hvis det altså kører på samme maskine.

Dette har dog den ulempe, at der til hver bruger skal laves et nyt interface hvad enten om det er en servlet eller en RMI stub. Med webservices og WSIF bliver det muligt at kombinere disse ved at beskrive interfacet i WSDL og tilbyde henholdsvis SOAP og Java bindings til operationerne.

Vi vælger derfor at benytte webservices til PaxFlow. Derved opnås en mere ensartet arkitektur på tværs af maskiner og platforme og forhåbentlig også en løsning der er forberedt på fremtidige udvidelser, i form af styring fra eksempelvis mobiltelefoner eller email.

Alle services bliver specificeret i WSDL med en SOAP og Java binding. Vi benytter derfor også WSIF til at kalde disse services fra alle klienter, der kører på Java.

Vi vælger desuden at benytte WSIL til serviceopdagelse, da WSIL er meget nemmere at håndtere end UDDI, specielt sålænge, vi har alt koden under vores kontrol.

2.6 Udvidet kravspecifikation

Der skal specificeres et sprog der kan beskrive følgende

- Diagram-typer:
  - Lagkage diagrammer.
  - Søjlediagrammer.
  - 2D Plots, herunder linje og punkt-plots.
- Data:
  - Serier af enkelte værdier med mærkater tilknyttet hver værdi.
  - Serier af 2-dimensionale punkter med et mærkat tilknyttet selve serien.
- Akser:
  - Liniære akser.
  - Logaritmiske akser.
  - Inddeling af akserne.
  - Primære akser og sekundære akser.
- Relationen imellem diagrammers akser og data.

I den forbindelse skal der udarbejdes et Schema der definere det lille domænespicifikke sprog der benyttes til at udtrykke diagrammerne. Schemaet kan desuden benyttes til validering af input. Desværre findes der intet Schema for SVG, og det er klart udenfor dette

projekts rammer at udarbejde et. Men det betyder at vi er afskåret fra at benytte Schema til validering af output.

Derefter skal der udarbejdes XSL-stylesheets der udtrykker transformationen fra diagramsproget til den enkelte diagram type udtrykt i SVG. Disse stylesheets forventes at genbruge kode så vidt dette er muligt fra diagramtype til diagramtype. XSL-stylesheets

Der skal defineres XSL-stylesheets beskriver transformationen fra udtryk i sproget til de enkelte typer af diagrammer.

Det ovenstående skal tilbydes til omverdenen som en webservice i sammenhæng med rapportgenerering på PaxFlow serveren. Denne webservices forestilles udviklet baseret på et SOAP API. Derved kan Schema filen der beskriver input sproget benyttes til definere input til webservicen. Webservicen skal kaldes fra både klienter og serveren selv.

3 Design

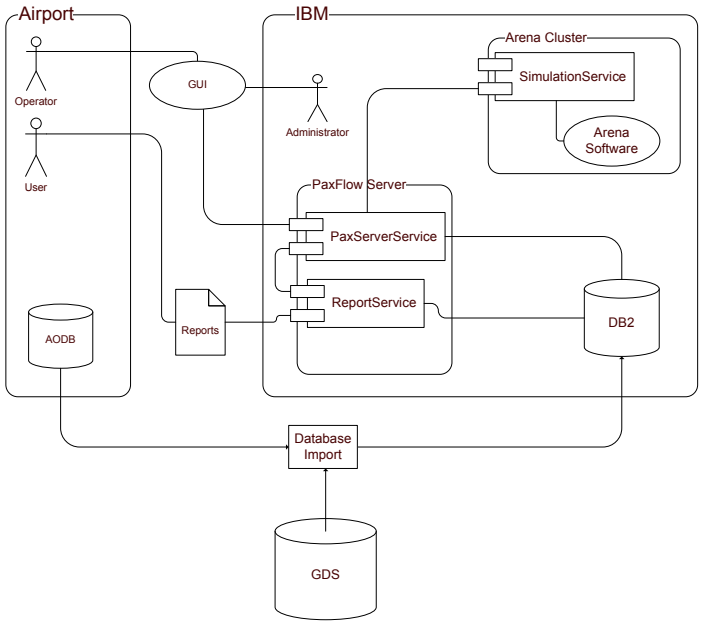
3.1 Generelle designovervejelser

Diagram- og rapportgenerering skal som tidligere nævnt benyttes i PaxFlow applikationen, derfor gives her et kort overblik over, hvor disse moduler passer ind i den overordnede arkitektur.

Opbygning af PaxFlow

PaxFlow består overordnet set af følgende dele (illustreret i figur 5):

- En PaxFlow server
- En installation af Arena simuleringssoftware
- En database
- PaxFlow administrations GUI
- Websider til at hente rapporter



Figur 5: PaxFlow systemet

PaxFlow serveren sørger sørger for adgangskontrol, import/eksport fra simulering og bookingsystemer, rapport generering samt meget mere. Arena softwaren tager sig af at afvikle selve simuleringen og skrive resultaterne tilbage til serveren. Databasen indeholder stort set al data, der benyttes i PaxFlow, lige fra bookingdata til rapport templates.

Administrations GUI benyttes til opsætning af simuleringsscenarier, f.eks. lufthavnens grundplan, skemaer for bemanding af kiosker, gates etc. Desuden kan der foretages administration af brugere samt oprette nye rapporter. Websiderne giver adgang til rapporter fra simuleringskørslerne.

PaxFlow serveren samt databasen kører på hver sin egen maskine. Arena installationen er en samling af Windows maskiner med hver sin Arena installation. En af maskinerne med Arena sørger for kommunikation mellem Arena maskinerne og serveren, f.eks. delegerer den simuleringsjob til den enkelte maskine.

Administrations GUI benyttes både af interne IBM administratorer samt lufthavnens ansvarlige. Websiderne benyttes udelukkende af butiksejere, grænsepoliti etc. i den enkelte lufthavn.

Design i en Service-Orienteret Arkitektur

At designe en applikation efter en service-orienteret arkitektur betyder, at der skal tages stilling til en række problemstillinger, der følger af særligt webservices' arkitekturen.

Da webservices standarderne hovedsageligt er bygget på upålidelige protokoller som HTTP og SOAP, skal der overvejes, hvilken betydning en tabt besked har.

En webservice specificerer ofte et interface, der aggregerer og derefter udstiller en række komponenter og klassers funktionalitet samlet. Derved opnås en høj grad af uafhængighed mellem selve interfacet og implementation af de enkelte klasser. Samtidig skal beskedformaterne specificeres på en effektiv måde.

Med hensyn til effektivitet tænkes her på størrelsen af den enkelte besked/antallet af beskeder, der sendes vs f.eks. SOAP protokollens relativt store overhead. Ofte vil det være ønskværdigt, at webservicen laver så meget arbejde som muligt ved hvert kald, derved holdes omkostningerne ved kommunikationen nede. Hvis der eksempelvis skal sendes information tilbage fra et kunderegister, kunne man forestille sig en metode, der givet et kundenummer returnerer navnet, samt en metode der givet kundenummer returnerer kundes adresse. Dette kunne nemt være metoderne i en kundeklasse. Hvis webservicen udstiller disse metoder, som de er implementeret i klassen, ville det føre til 2 kald over SOAP for at kunne få kundens navn og adresse.

Webservicen kan tilbyde en metode, der givet et kunde id sender navn og adresse i en samlet besked, hvorved der elimeneres et dyrt kald over SOAP. Det må dog komme an på den enkelte situation, hvorvidt det er mest fordelagtigt at sende en stor besked eller flere små. Ofte afhænger dette af, hvordan man forestiller sig webservicen brugt, hvilket kan være svært, hvis det er en offentlig service, men her må det være op til udviklerens skøn (erfaring) at vælge den ene eller anden.

På den måde minder det meget om at designe til et komponentbaseret arkitektur, men samtidig bør man også overveje hvilke bindinger, man stiller sine operationer til rådighed igennem. Hvis det er nødvendigt, at beskeden når frem pålideligt, kan det være en dårlig ide at bruge en SOAP binding. Her ville en binding via Java Message Service (JMS)[8] være et bedre valg, da denne protokol netop er lavet til dette. Evt. kunne SOAP beskederne også sendes over JMS istedet for over HTTP.

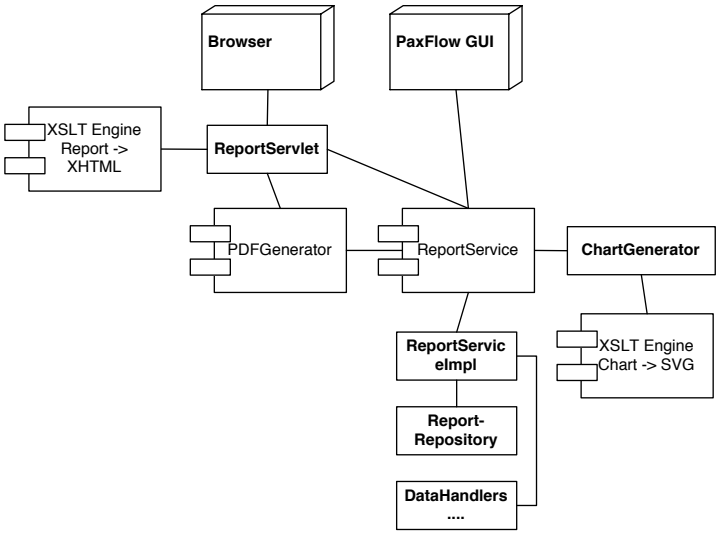
Til sidst bør det også overvejes, hvorvidt kompatibilitet med andre implementeringer af standarderne er et ubetinget krav. På trods af meget arbejde med at få de enkelte implementeringer til at arbejde sammen, så er der endnu mange områder, særligt når det kommer til specifikationernes mere eksotiske dele, hvor der stadig er problemer med at integrere de forskellige implementeringer. Dette kan have store konsekvenser for designet. F.eks. er den absolut nemmeste måde at lave webservices på i Java verden via SOAP med RPC-stilen, men da denne ikke er understøttet af Microsoft, må kravene til den enkelte applikation bestemme, hvorvidt en sådan løsning vælges.

3.2 Overordnet design

I PaxFlow systemet skal der skal laves diagrammer til brug for følgende:

- Indlejring i HTML sider som derefter vises i en webbrowser.
- Indlejring i en rapport XML-fil som derefter transformeres til PDF-fil. PDF-filen kan derefter enten sendes pr. email eller hentes fra en webbrowser.
- Indlejring i en rapport XML-fil som derefter kan vises i et GUI.

Dette forestilles opbygget som vist på figur 6. Argumentationen for dette præsenteres i de følgende afsnit.



Figur 6: PaxFlow ReportService design

Centralt er ReportService. ReportService består af en række klasser, der sørger for at holde styr på alle de rapporter, en bruger kan tilgå samt, hvordan disse kan fyldes med data. Alle rapporter er specificeret i et rapportsprog defineret på samme måde som diagramsproget. Dette sprog vil dog ikke blive beskrevet nærmere. Inddata til ReportService er et ID, der indikerer hvilken rapport, der skal hentes fra databasen. En klasse på serveren indeholder det samme ID. I selve rapportsproget er indeholdt diagramelementer, der indikerer hvilken klasse på serveren, der skal benyttes hente data. Derefter hentes de data, der passer til de enkelte diagrammer fra databasen.

Resultatet er en rapport XML-fil, der indeholder indlejrede diagramspecifikationer. Diagramspecifikationerne sendes videre til ChartGenerator klassen, der sørger for at kalde XSLT-processoren, således at alle diagramelementerne transformeres til SVG, hvorefter disse erstatter diagramelementerne i rapport XML-filen.

Hvis ReportService har modtaget kaldet fra PaxFlow GUI, returnerer den rapport XML-filen til GUI'et, hvorefter den kan vises. Hvis kaldet oprindeligt kommer fra en browser, er det nødvendigt at sætte en Servlet imellem. Denne skal sørge for at pille alle SVG-elementer ud af rapport XML-filen, gemme disse på et session objekt. Derefter sendes dokumentet uden indlejrede SVG-diagrammer og transformeret til HTML til browseren, der så beder Servlet'en om de enkelte SVG-diagrammer.

Grunden til denne lille omvej skyldes, at ingen browsere på nuværende tidspunkt understøtter indlejring af SVG-filer i XHTML. Havde de understøttet dette kunne en rapport fil transformeret til XHTML med de indlejrede SVG-diagrammer istedet være sendt direkte til browseren.

Hvis webbrowseren bad om en rapport i PDF-format, vil rapport XML-filen inklusive de indlejrede SVG-diagrammer sendes til PDFGenerator, der sørger for at konvertere denne til PDF. Den returnerer den færdige PDF-fil til ReportServlet, der sender den til webbrowseren. Udover ovennævnte funktioner kan ReportService også levere en liste over alle tilgængelige rapporter.

ReportService implementeres som en webservice med to funktioner: getReport, der tager en ID streng som argument og returnerer et rapport XML-dokument samt getReportList, der ingen argumenter tager og returnerer ID strenge for alle tilgængelige rapporter. Man kunne have haft flere metoder, f.eks. en til at hente rapporten, en til at lave SVG-diagrammerne etc. Men det vurderes, at det ekstra arbejde, ved at klienten specifikt skal bede om disse ting, overstiger det, der i givet fald kunne vindes ved, at serveren skulle lave mindre fra

starten. Ved at implementere ReportService som en webservice opnås desuden en ensartet arkitektur for PaxFlow, da kommunikationen med serveren og Arena maskinerne også tænkes implementeret som webservices.

Webservicen kan kaldes både fra GUI'et via SOAP-bindinger og fra ReportServlet via Java bindinger.

4 Implementering

4.1 Implementering af Diagramsproget

Dette afsnit gennemgår sproget med et lagkagediagram som eksempel. Yderligere præsenteres et diagram over samtlige typer i sproget, der inddrages i lagkageeksemplet.

Lagkagediagrammer

Lagkagediagrammer er repræsenteret i sproget med `piechart` elementer. Et lagkagediagram har en serie af endimensional data tilknyttet samt en x-akse. Lagkagediagrammet bruger dataen som grundlag for at tegne sig selv. Aksen definerer minimums- og maksimumsværdier, der bruges til at normalisere værdierne under tegneprocessen.

Relationen til dataen og akserne sker igennem en association, frem for en komposition. I praksis har *PieChartType* en `@dataref` attribut, som peger på et `@id` attribut på et *Point1Collection*-element. I XML Schema filen for sproget er `@id`-attributten defineret som nøgle for *Point1Collection*-elementer. Ligeledes peger `@axisref`-attributten på et *AxisType*-element under *AxisContainerType*. Dette ses i figur 7 ved at *PieChartType*-typen har en associationsrelation til *AxisType* så vel som *Point1CollectionType*, begge annoteret med navnet på den attribut der indeholder referencen.

PieChartType er indeholdt af *PlotContainerType* igennem en *Choice*-relation. Dette vil sige, at *PieChartType* er en af de mulige typer som *PlotContainerType* kan indeholde. Enten indeholder *PlotContainerType* et *PieChartType*-element, et *LinePlotContainerType*-element eller et *BarChartType*-element. Igen er relationen annoteret med navnene på de elementer der indeholder referencen.

PlotContainerType er indeholdt af *ChartType*, som er typen på rodelementet i sproget. *ChartType* er defineret som en sekvens af elementer, hvor dækkefølgen skal overholdes i et gyldigt dokument. Før *PlotContainerType* kommer *AxisContainerType* og efter kommer *PointDataCollectionType*.

PointDataContainer indeholder et vilkårligt antal og blanding af *Point1Collection*-elementer og *Point2Collection*-elementer, dog skal der mindst være enten et element af typen *Point1Collection* eller *Point2Collection*.

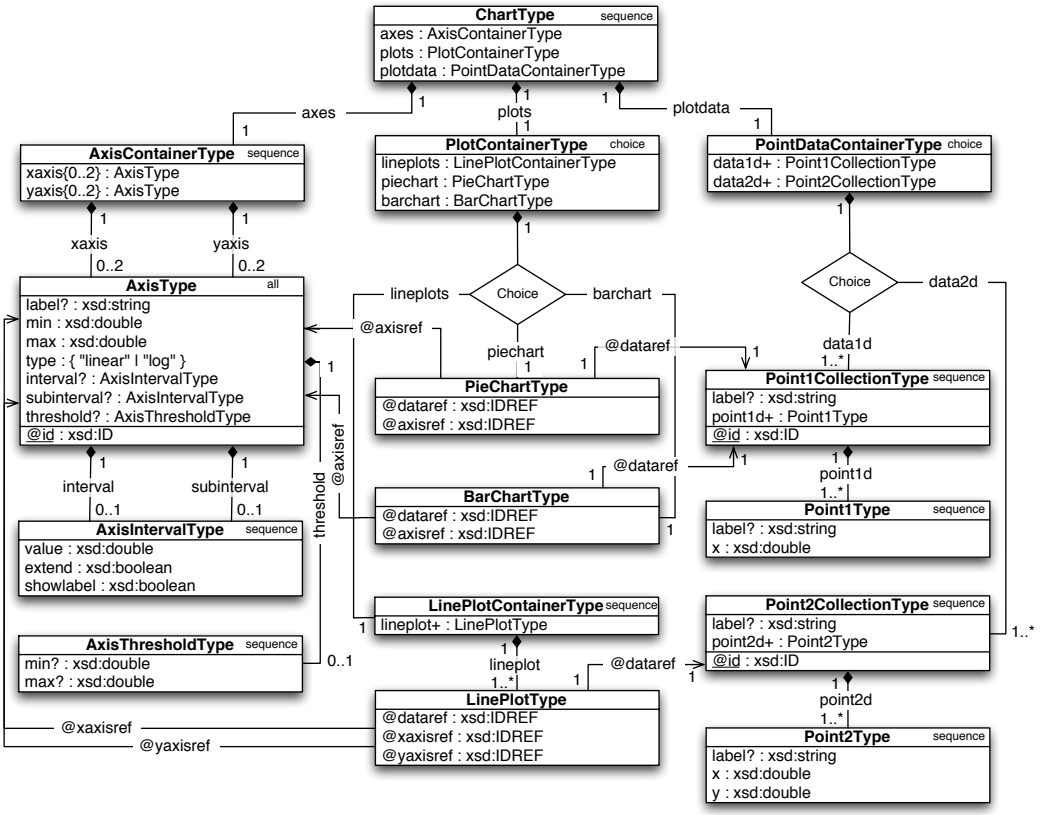
Resten af sproget

Resten afsproget er illustreret i figur 7, baseret på det tilhørende XML Schema. Notationen er konstrueret til lejligheden og er ikke beregnet til at kunne beskrive vilkårlige XML Schemas, kun dette enkelte tilfælde. Hovedinspirationen kommer fra UML samt [27].

Komplekse typer er modelleret som kasser, inddelt i op til tre felter. I øverste felt står typens navn, i det midterste felt står de elementer der indgår i typen og nederste felt står attributterne. Hvis en type ikke indeholder elemter eller attributter undlades det relevante felt. Hvis et barn er en kompleks type fra samme namespace, angives dette med en UML kompositionsrelation (udfyldt diamant). Nøglereferencer angives med en UML associationsrelation (pil) Hvis et barn eller en attribut er en standardtype fra XML Schema, angives dette med `xsd:` namespacet i typenavnet.

Attributter markeres med præfikset '@', ligesom i XPath. Kardinaliteten mellem elementer angives med kvantorer kendt fra regulære udtryk bag på elementnavnene. Eksempelvis udtrykkes at et element er valgfrit med `?`. Én eller flere forekomster udtrykkes med `+`, mellem nul og to forekomster med {`0..2`} og så videre.

I sproget er nøgler altid defineret som attributter ved navn *id* af typen `xsd:ID`. Analogt med E/R diagrammer er nøglerne understreget i diagrammet. Nøgler skal være unikke inden for det XPath udtryk de er defineret i. Eksempelvis har `data1d` elementet en nøgle



Figur 7: Diagram over de forskellige typer i sproget

i dens `id` attribut. Dette er markeret i Schema ved at angive, at inden for nodesættet som udtrykket `chart/plotdata/data1d` resulterer i, skal `@id` være unik.

Hvis et element eller en attribut indgår i en relation annoteres relationen med navnet på elementet eller attributten. Ofte undlades elementer og attributter inde i selve typedeklarationen hvis de indgår i en relation[27], men her vises de både i deklarationen og på relationen. Derved er det nemmere at få et overblik over samtlige elementer og attributter der indgår i en type såvel som deres rækkefølge i de tilfælde hvor det er relevant.

Elementgrupperingen er angivet i øverste højre hjørne, med mulighederne *all*, *choice* eller *sequence*. I tilfælde af *sequence*-gruppering gælder samme rækkefølge i diagrammet som definitionsrækkefølgen i Schemaet. Ved *choice*-gruppering deler kompositionsrelationen sig i *n* dele, markeret med en choice-diamant.

4.2 Transformering

Implementeringen af transformeringerne vil her blive beskrevet. Da der ikke forudsættes dybere kendskab til XSLT elle SVG, gives der først en kort introduktion til de begreber der benyttes i beskrivelsen af implementeringen.

XSLT

I en XSL-transformering indgår der tre elementer. Kildetræet, et stylesheet og resultattræet. Kildetræet kan i dette tilfælde være et udtryk i diagrambeskrivelsen og resultattræet SVG-filen.

En transformeringsbeskrivelse er bygget op af en række transformeringsregler. Hver regel består af et mønster og en skabelon. En tranformeringsregel siges at *matche* et element,

hvis elementet opfylder mønsteret tilknyttet transformeringsregelen. Når et kildetræ skal transformeres, holdes rod-elementet op imod samtlige mønstre. Hvis elementet matcher et mønster, påføres den tilhørende skabelon elementet, og resultatet indsættes i resultattræet. XSLT definerer elementet `xsl:apply-templates`, der bevirker, at alle børn af det nuværende element i kildetræet, også kaldet *kontekst-elementet*, bliver påført alle definerede transformerings-regler. `xsl:apply-templates` tillader også at man selv definere kontekst-elementet, skabeloner kan derved påføre andre skabeloner på kildetræet, og derved styre traverseringen. Hvis XSLT møder et element, der ikke bliver matchet af nogen brugerdefinerede regler, vil det bliver påført en *default template*, der resultere i, at alle børn af elementet bliver traverseret. XSLT vil derved implicit sørge for, at alle elementer i kildetræet bliver traverseret. En samling af skabeloner der transformere et kildetræ til et resultattræ bliver kaldt et stylesheet.

En skabelon kan indeholde et hvert validt udtryk i XML. Ethvert udtryk der ikke tilhøre XSL-namespacet bliver en del af resultatet-træet. Følgende skabelon vil f.eks. omslutte indholdet alle `chart`-elementer med et `svg`-rodelement.

Listing 1: En skabelon der omslutter børn af det element den matcher med et `svg` element

```
<xsl:template match="chart">
   <svg height="400" width="400"
      <xsl:apply-template />
   </svg>
</xsl:template>
```

Ud over at lade en transformering ske, når dets mønster passer til et element, kan en skabelon navngives og påføres eksplicit. Ud over et mønster eller navn kan en skabelon specificere i hvilken tilstand (mode) den gælder.

Listing 2: En navngivet skabelon

```
<xsl:template name="drawAxes"> ... </xsl:template>
```

Sproget, der bruges til at beskrive XSL-transformeringer er deklarativt, har statiske scopes, og tillader ikke opdateringer af variabler. Det medfører, at resultatet af den følgende skabelon er "5", da den betingede variabeldeklaration ikke opdatere `x`, men definerer et nyt x, der ikke er synligt uden for betingelsens scope.

```
<xsl:template name="test">
   <xsl:variable select="5" name="x" />
   <xsl:if test="$x &lt; 5" />
      <xsl:variable select="10" />
   </xsl:if>
   <xsl:value-of select="$x" />
</xsl:template>
```

Bemærk at "<" skrives som "<" da "<" ikke er tilladt som attributværdi ifølge [6].

En variabel kan tildeles resultatet af ethvert gyldig XPath-udtryk [22]. Variablen vil altid have en af følgende typer:

- String
- Number
- Node-set
- Boolean

XSLT indfører yderligt typen *result tree fragment*. Et node-set er en del af kildetræet og kan derfor bruges i XPath-udtryk. De enkelte elementer i et node-set kan adresseres, og hele sættet kan traverseres. Et result tree fragment er et fragment af resultattræet og bliver mere eller mindre behandlet som en streng. Et result tree fragment er dog et særligt node-set, og alle node-set operationer, der er tilladt på strenge er derfor også tilladte på et result tree fragment.

SVG

SVG er et vektoriseret grafikformat. Når diagrammet skal udtrykkes i SVG sker det via en blanding af tegne og manipulations operationer. SVG understøtter en række operationer lige fra simple tegneoperationer til avancerede transformationer og kompositioner af de grafiske elementer.

Følgende er en kort beskrivelse af de SVG-elementer der bliver benyttet mest i diagrambeskrivelsen.

`text` Indsætter en tekst-streg i dokumentet.

`path` tegner rette og krumme linjer imellem en række koordinater. Kan også bruges til at tegne former der kan udfyldes med farver.

`polyline` en simplificeret version af `path` der tegner rette linjer imellem et række punkter.

`rect` regner et rektangel.

SVG understøtter transformeringer af det koordinatsystem de ovenstående elementer operere i. Følgende bliver benyttet under diagram-genereringen.

Translatering flytter koordinatsystemets origo (0,0). Origo ligger som udgangspunkt i øverste venstre hjørne af tegneområdet.

Skalering Skalerer koordinat systemet. Der kan også angives negative værdier, hvilket vil medfører at akserne bliver vendt. Som udgangspunkt vender y-aksen "nedad".

Rotering Rotere koordinat-systemet, kan f.eks. anvendes til at rotere tekst.

For en nærmere beskrivelse af SVG se [13].

Overordnet beskrivelse af tranformeringen

De forskellige diagramtyper produceres på forskellig vis. Transformeringen til de tre forskellige typer er derfor implementeret i tre separate diagramstylesheets.

Diagrammerne har dog visse ting til fælles, og disse funktioner er derfor implementeret i separate stylesheets, der bliver inkluderet ind i de stylesheets, der har brug for dem.

Det drejer sig om

- Optegninger af akser, deles af søjle- og punkt-diagrammer
- Farveangivning
- Signaturforklaring
- Placing af diagrammet (layout)

Farveangivningen, signaturforklaring og placeringsskabelonerne er samlet i det fælles stylesheet `common.xsl`, mens akseoptegningen, der ikke bruges af lagkagediagrammet, er implementeret i stylesheetet `axis.xsl`.

Fælles funktionalitet

Det fælles stylesheet indeholder først og fremmest den skabelon, der matcher `chart`-elementet. Skabelonen sørger for at indsætte SVG-kode, der placerer signaturforklaring og diagrammet korrekt i forhold til hinanden. Det sker ved at inddele tegneområdet i områder, ét der indeholder signaturforklaringen, og ét der indeholder det egentlige diagram. Skabelonerne der optegner de forskellige diagramtyper tager deres højde og bredde som parametre. Placeringen af den konkrete diagramtype sker ved at angive højden og bredden af den tilhørende kasse, hvorefter skabelonen vil sørge for at diagrammet passer ind. Diagramskabelonerne påføres ved at kalde `apply-templates` med `plots`-elementet som kontekstelement.

Layout

Diagrammet og signaturforklaringen skal placeres korrekt i forhold til hinanden således at de ikke overlapper hinanden. Skabelonen, der matcher det yderste element i diagrambeskrivelsen (`chart`), er implementeret i det fælles stylesheet. Skabelonen indsætter SVG-rodelementet, der angiver størrelsen af det endelige SVG-dokument. Dette område bliver opdelt i to tegneområder til henholdsvis diagrammet og signaturforklaringen. Alle skabeloner der optegner diagrammer, tager deres højde og bredde som parameter. Når et diagram skal optegnes, translateres koordinat-systemet til det øverste venstre hjørne af diagrammets tegneområde, og diagrammets skabelon kaldes med højden og bredden af dets tegneområde som parameter.

Listing 3: Skabelonen der placere signaturforklaring og diagram

```
1   <xsl:variable name="CANVASWIDTH" select="600"/>
2   <xsl:variable name="CANVASHEIGHT" select="400"/>
3   <xsl:variable name="LABELWIDTH" select="75"/>
4   <xsl:variable name="LABELMARGIN" select="5"/>
5   <xsl:template match="pc:chart">
6      <xsl:variable name="plotwidth" select="$CANVASWIDTH - 3 * $MARGIN
           - $LABELWIDTH"/>
7      <svg width="{$CANVASWIDTH}" height="{$CANVASHEIGHT}" xmlns="http:
           //www.w3.org/2000/svg">
8         <g transform="translate({$MARGIN},{$MARGIN})">
9            <xsl:if test="$DEBUG">
10              <rect x="0" y="0" width="{$plotwidth}" height="{$
                    CANVASHEIGHT - 2 * $MARGIN}" stroke="red" fill="none"
                    />
11           </xsl:if>
12           <xsl:apply-templates>
13              <xsl:with-param name="width" select="$plotwidth"/>
14              <xsl:with-param name="height" select="$CANVASHEIGHT
                     - 2 * $MARGIN"/>
15           </xsl:apply-templates>
16        </g>
17        <g transform="translate({2 * $MARGIN + $plotwidth},{$MARGIN})">
18           <xsl:if test="$DEBUG">
19
20              <rect x="0" y="0" width="{$LABELWIDTH}" height="{$
                    CANVASHEIGHT - 2 * $MARGIN}" stroke="red" fill="none"
                    />
21           </xsl:if>
22           <xsl:apply-templates mode="label">
23              <xsl:with-param name="width" select="$LABELWIDTH"/>
24              <xsl:with-param name="height" select="$CANVASHEIGHT
                     - 2 * $MARGIN"/>
25           </xsl:apply-templates>
26        </g>
27     </svg>
28  </xsl:template>
```

Referencer

Den generiske metode til at referere til elementer i kildetræet er via XPath-udtryk. Disse kan dog hurtigt blive uoverskuelige, hvis man bevæger sig for langt væk fra kontekstnoden. Følgende er et eksempel fra en tidligere version af implementeringen.

```
1  ../../pc:axes/pc:xaxis/pc:max[../../@id = $xref]
```

Diagrambeskrivelsen indeholder interne referencer, blandet andet imellem et plotelement og dets akse. Det ovenstående er netop et eksempel på et XPath udtryk, der finder max-værdien for en x-akse tilknyttet et lineplot. Da dette ikke er et unormalt problem, indeholder XSLT funktioner til at afhjælpe dette. XSLT elementet `xsl:key` definerer en nøgle, der senere kan benyttes til at slå elementer op. Det fælles stylesheet definerer nøgler, der kan bruges til opslag af de mest benyttede referencer

```
<!-- Define dataxd/@id as a key -->
<xsl:key name="data1dkey" match="pc:data1d" use="@id" />
<xsl:key name="data2dkey" match="pc:data2d" use="@id" />
<xsl:key name="data3dkey" match="pc:data3d" use="@id" />
<!-- Define data1d/@id as a key -->
<xsl:key name="xaxiskey" match="pc:xaxis" use="@id" />
<xsl:key name="yaxiskey" match="pc:yaxis" use="@id" />
```

Følgende udtryk finder max-værdien for `<xis`-elementet med id “xakse”.

```
key('xaxiskey', @xaxisref)/pc:max
```

Farver

Ifølge [10] bør forskellige diagramtyper farvelægges efter forskellige paletter. De enkelte elementer i et diagram (f.eks. udsnit af et lagkagediagram) farvelægges efter den rækkefølge, de er defineret i. Givet en diagramtype samt et elements position returnerer `colorByPosition`-skabelonen den korrekte farve. Farveværdien findes ved at slå positionen op i farvepaletten anbefalet af [10]. Denne palette er defineret i filen `colors.xml`(se kodeliste 4), der bliver indlæst som en variabel i common.xsl. Skulle en instans af et diagram kræve flere farver end der er defineret i paletten vil `colorByPosition` lade farveserien gentage sig. Er der således fem farver defineret, og en skabelon forespørger om farven for element nr syv, vil farve nummer to bliver returneret.

Listing 4: Et udsnit af farvepaletten

```
<chartcolors>
   <colors plottype="default">
      <color name="red" value="#ff0000" />
      <color name="green" value="#00ff00" />
      <color name="blue" value="#000000" />
   </colors>
   <colors plottype="lineplot">
      <color name="12" value="rgb(240,180,0)" />
      <color name="20" value="rgb(30,108,11)" />
      <color name="28" value="rgb(0,72,140)" />
      .
      .
   </color>
   .
   .
</chartcolors>
```

Signaturforklaring

Signaturforklaringen indeholder en liste af alle mærkater, der indgår i diagrammet, samt de tilknyttede farver. For at kunne generere denne liste kræves blot information om positionen af de forskellige mærkater, da `colorByPosition`-skabelonen efterfølgende kan bruges til at finde mærkatets farve. Skabelonen der genererer signaturforklaringen er generel. Den skal derfor også have information om hvilken type, diagram den skal beskrive, da farvernepaletten som før beskrevet er forskellig for forskellige diagramtyper.

Da data i de forskellige diagrammer er struktureret forskelligt, er der som udgangspunkt ikke nogen generel måde signaturforklaringsskabelonen kan afgøre hvilke mærkater, der skal indgå i signaturforklaringen. Dette problem er løst ved lade den enkelte diagramskabelon om at angive hvilke mærkater, der indgår i den, og videregive denne information til signaturforklaringsskabelonen.

Når diagram-skabelonen bliver påført i tilstanden *label* forventes det, at den udvælger et node-set indeholdende alle dens mærkater i den korrekte rækkefølge og påføre *drawLegend*-skabelonen med det udvalgte node-set som parameter. Kodeliste 5 viser et eksempel på hvordan mærkater for et lagkagediagram opsamles og sendes videre til signaturforklaringensskabelonen.

Listing 5: Påførsel af signatur-forklarings skabelonen

```
<!-- Extract our labels and pass them on to the legend template -->
   <xsl:template match="pc:piechart" mode="label">
      <xsl:param name="width" select="$LABELWIDTH" />
      <xsl:param name="height" select="$CANVASHEIGHT - 2 * $MARGIN" />

      <xsl:call-template name="drawLegend">
         <xsl:with-param name="labels" select="key('data1dkey',@dataref)
             /pc:point1d/pc:label" />
         <xsl:with-param name="width" select="$width" />
         <xsl:with-param name="height" select="$height" />
      </xsl:call-template>
   </xsl:template>
```

Akser

Akserne bliver genereret ved en transformering af `axes` elementet. `axes` kan indeholde en eller flere instanser af `axis`-elementet, der repræsenterer en enkelt akse. Diagramstylesheets, der har behov for akser, inkluderer det fælles akse-stylesheet, der indeholder en skabelon, der matcher på `axes`-elementet og efterfølgende tegner akserne. Implementeringen af transformeringerne introducerer et problem. Diagramskabelonen modtager som før nævnt diagrammets højde og bredde som parameter, hvis diagrammet overskrider disse begrænsninger vil det enten tegne uden for billedet eller oven i signatur-forklaringen. Implementeringen af den fælles akse-skabelon, tegner sig uden på den kasse, som den får opgivet som tegneområde. Det betyder, at akserne vil komme til at overskride tegneområdet.

Implementeringen løser dette problem ved at definere et nyt tegneområde inde i området diagramskabelonen får tildelt. Dette område placeres således, at margen imellem det og det oprindelige tegneområde, er stor nok til at indeholde akserne og de tilhørende mærkater. Da akseskabelonen derfor skal have oplyst den "nye" højde og bredde, indsættes der en skabelon, der matcher `axes` i diagramskabelonen. I det fælles akse-stylesheet implementeres ligeledes en skabelon der matcher `axes`, men denne gælder kun i tilstanden "drawaxes". Skabelonen i diagramskabelonen modtager højde og bredde på det oprindelige tegneområdet, udregner størrelsen på det nye tegne område, og kalder apply-templates i tilstanden "drawaxes". Følgende kodeliste viser et eksempel på, hvordan skabelonen i et diagram-stylesheet er implementeret.

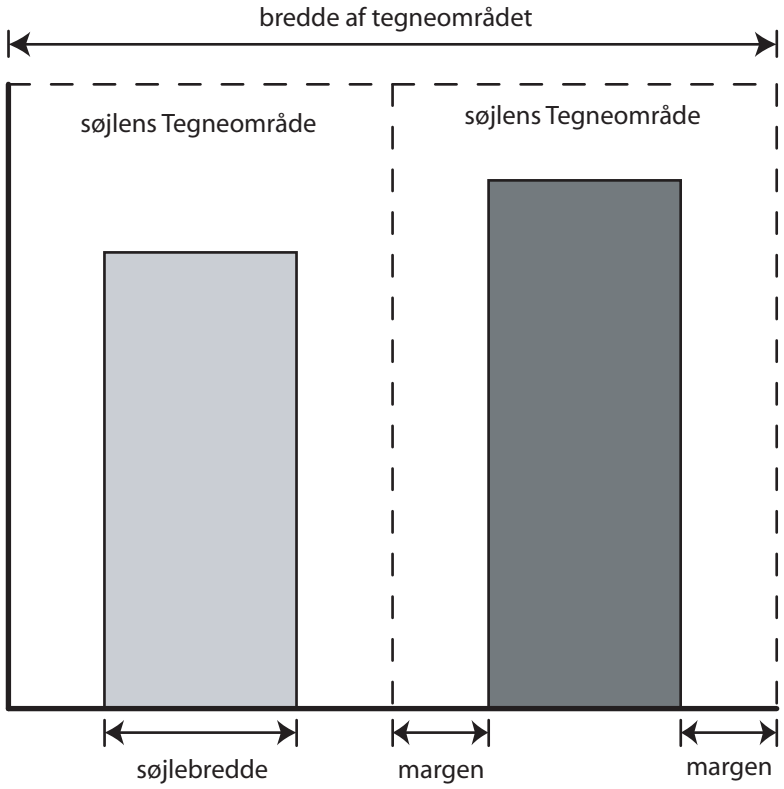
Listing 6: Skabelonen, der udregner det nye tegneområde og påfører den fælles akseskabelon

```
   <xsl:template match="pc:axes">
      <xsl:param name="width"/>
      <xsl:param name="height"/>
      <xsl:variable name="margin">50</xsl:variable>
      <xsl:variable name="width" select="$width - (2 * $margin)"/>
      <xsl:variable name="height" select="$height - (2 * $margin)"/>
      <g transform="translate({$margin},{$margin})">
```

```
8           <xsl:apply-templates select="." mode="drawaxes">
9              <xsl:with-param name="canvasWidth" select="$width"/>
10             <xsl:with-param name="canvasHeight" select="$height"/>
11          </xsl:apply-templates>
12       </g>
13    </xsl:template>
```

For at lette implementeringen skal brugeren selv angive maximum- og minimumværdier for akserne. Dermed undgår akse-stylesheetet at skulle gennemsøge al data der tegnes op imod en given akse, for at udregne de korrekte værdier.

Søjlediagram



Figur 8: En oversigt over de variable størrelser i et søjlediagram

Et søjlediagram bruges til at sammenligne en række værdier. Værdierne er i diagrambeskrivelsen repræsenteret som et enkelt `data1d` element indeholdende en række `point1d` elementer. Hvert enkelt `point1d` element indeholder et mærkat der angiver navnet på den værdi elementet repræsentere. Et søjlediagram indeholder en enkelt y-akse. Før værdierne kan bruges direkte i diagrammet, skal de skaleres således at de passer til aksen. Skaleringen udregnes efter følgende formel.

$$skalering = \frac{tegneomraade_{hoejde}}{akse_{max}} \tag{1}$$

Før tegningen af søjlerne påbegyndes, translateres origo til nederste venstre hjørne af koordinatsystemet. Derefter skaleres koordinatsystemet med (1,-1). De enkelte søjler optegnes med et `rect`-element. `rect` tager 4 parametre, x og y koordinatet for nederste venstre hjørne af rektanglet, samt højde og bredde. Da y-aksen på grund af skaleringen nu peger "opad" kan de skalerede værdier indgå direkte som højden af rektanglet.

Før søjlen kan tegnes skal der dog beregnes 2 konstanter, søjlens bredde, og dens margen. Først beregnes det tegneområde hver enkelt søjle har til rådighed.

$$tegneomraade_{soejle} = \frac{tegneomraade_{bredde}}{antalsoejler} \tag{2}$$

Margen rundt om en enkelt søjle findes ved at dividere søjlens med en konstant. Denne konstant kan senere bruges til at justere margen.

$$soejle_{margen} = \frac{tegneomraade_{soejle}}{margenfaktor} \tag{3}$$

Bredden af den enkelte søjle vil nu være den plads der er tilbage i tegneområdet.

$$tegneomraade_{bar} - (2 * soejle_{margen}) \tag{4}$$

Startkoordinatet for rektanglet vil nu være

$$x = (soejle_{position} - 1) * \frac{tegneomraade_{bredde}}{antalsoejler} + soejle_{margen} y = 0 \tag{5}$$

SVG-elementet der tegner det enkelte rektangel vil nu se ud som følger:

```
1   <rect x="{$x}" y="{0}" width="{$bar_width}" height="{$value * $scale}
        " fill="{$color}" stroke="#000000"/>
```

Punktdiagram

Et punktdiagram bruges til at give overblik over en eller flere større serier af 2-dimensional data. Punktdiagrammet tegnes som en eller flere linjer, hvor hver linje forbinder en række 2-dimensionelle punkter. Dette er repræsenteret i diagrambeskrivelsen som et `lineplots` element der indeholder en eller flere `lineplot` elementer. Hvert `lineplot` element har en reference til et `data2d` element, samt den henholdsvis x- og y-akse dataen skal tegnes ud fra. `data2d` elementet indeholder en serie af `point2d` elementer der angiver de punkter der skal tegnes. Mærkatet til de enkelte linje er angivet i `data2d` elementerne.

Når diagrammet skal optegnes sker det ved at påføre `lineplots` elementet, en skabelon der efterfølgende påføre skabeloner til de underliggende `lineplot` elementer. Da punktdiagrammet indeholder akser, skal dets tegneområde påføres en margen før de enkelte linjer kan tegnes. Denne påføres efter samme fremgangsmåde som beskrevet i afnittet om søjlediagrammer. Da akserne kan have to forskellige orienteringer, bruges højde og bredde af tegneområdet til at skalere henholdsvis y- og x-akserne. F.eks. udregnes skaleringen af y-aksen på følgende måde:

$$skalering = \frac{tegneomraade_{hoejde}}{yakse_{max} - yakse_{min}} \tag{6}$$

Når det enkelte lineplot skal tegnes, udregnes farven af linjen ud fra `lineplot` elementets position. Derefter gennemløbes hvert enkelt tilknyttet `point1d` element, og det tilsvarende punkt i diagrammets koordinatsystem udregnes. Der skal her tages højde for skaleringen af både x- og y-aksen.

$$(x_{diagram}, y_{diagram}) = (x_{point1d} * skalering_{xakse}, y_{point1d} * skalering_{yakse}) \tag{7}$$

En akse kan angives som værende logaritmisk. Hvis en akse er logaritmisk skal der tages højde for dette når et punkts koordinat, samt skaleringen for en akse udregnes. I følgende eksempel er y-aksen logaritmisk.

$$skalering = \frac{tegneomraade_{hoejde}}{log(yakse_{max} - yakse_{min})} \tag{8}$$

$$(x_{diagram}, y_{diagram}) = (x_{point1d} * skalering_{xakse}, log(y_{point1d}) * skalering_{log-yakse}) \quad (9)$$

Da logaritmen for negative værdier ikke er defineret, undersøges dataen for negative værdier før den behandles. Hvis der optræder negative værdier stoppes tranformeringen, via XSL-elementet `message`.

Listing 7: Transformeringen stoppes hvis exponentiel data indeholder negative værdier

```
<xsl:if test="($yType = 'log' and $yMin &lt; 0 or $yMax &lt; 0) or ($
    xType = 'log' and $xMin &lt; 0 or $xMax &lt; 0)">
  <xsl:message terminate="yes">
    Negative values are not valid for axes with type=log
  </xsl:message>
</xsl:if>
```

Når linjen imellem de enkelte punkter i diagrammet skal tegnes sker det med SVG-elementet `polyline`. XSLT-elementet `element` kan bruges som et alternativ til at skrive SVG-elementer direkte ind i skabelonen. Dette kan være en fordel hvis attributter, som i dette tilfælde, indeholder data der skal beregnes.

Listing 8: `polyline`-elementet bruges til optegning af linjer i punktdiagrammet

```
<xsl:element name="polyline" >
 <xsl:attribute name="fill">none</xsl:attribute>
  <xsl:attribute name="stroke">
   <xsl:value-of select="$strokeColor"/>
  </xsl:attribute>
  <xsl:attribute name="stroke-width">
   <xsl:value-of select="$strokeWidth"/>
  </xsl:attribute>
  <xsl:attribute name="points">
   <xsl:for-each select="$points">
    <xsl:choose>
     <xsl:when test="$xType = 'log'">
      <xsl:variable name="x">
       <xsl:value-of select="pc:x"/>
      </xsl:variable>
      <xsl:value-of select="math:log(pc:x) * $xScale"/>
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="pc:x * $xScale"/>
     </xsl:otherwise>
    </xsl:choose>
    <xsl:text>,</xsl:text>
    <xsl:choose>
    <xsl:when test="$yType = 'log'">
     <xsl:variable name="y">
      <xsl:value-of select="pc:y"/>
     </xsl:variable>
     <xsl:value-of select="math:log(pc:y) * $yScale"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="pc:y * $yScale"/>
    </xsl:otherwise>
   </xsl:choose>
   <xsl:text>&#x20;</xsl:text>
  </xsl:for-each>
 </xsl:attribute>
</xsl:element>
```

Lagkagediagram

Ligesom søjlediagrammer bruges lagkagediagrammer til at sammenligne endimensionelle værdier. Diagrammet har en række `point1d`-elementer knyttet til sig samt en akse, der definerer minimums- og maksimumsgrænserne for de værdier der skal visualiseres. Der tegnes et udsnit af diagrammet for hvert `punkt1d`-element.

Lagkagediagrammet tegnes i to omgange: først tegnes de udfyldte udsnit, derefter tegnes omridset af hvert udsnit sammen med værdien som udsnittet repræsenterer.

Størrelsen for diagrammet afhænger af størrelsen på den mindste kant i det kanvas diagrammet er indlejret i. Størrelsen og orienteringen på det enkelte udsnit afhænger af startvinkelen og slutvinkelen. Startvinkelen findes ved at normalisere summen af samtlige foregående udsnit i forhold til den maksimale værdi defineret i aksen, skaleret med 2π. Slutvinkelen findes ligeledes ved at normalisere udsnittets værdi i forhold til den maksimale værdi og skalere denne med 2π og lægge oven i startvinkelen.

Selve tegningen af et udsnit sker hovedsageligt ved hjælp af SVGs elliptiske bue funktion[15] i et `path`-element, som demonstreret i liste 9. Parametrene `startx`, `starty`, `endx` og `endy` angiver de koordinater hvor buen skal starte og slutte, og `radius` angiver buens radius.

Listing 9: SVG-elementet der tegner en bue

```
1   <path d="M 0,0 L {$startx},{$starty} A{$radius},{$radius} 0 0 1 {$
        endx} {$endy}" />
```

4.3 Implementering af ReportService

I dette afsnit beskrives det hvordan ReportServicen er blevet defineret i WSDL filen vha. af et par eksempler. Derefter følger afsnit om implementeringen af Java klasserne bag webservicen.

Beskrivelse af ReportService

ReportService er beskrevet i en WSDL fil der kan ses i bilag D.3 på side 81.

Webservicen er implementeret som en Message-style webservice under Apache Axis[1], Axis er Apache projektets implementation af SOAP.

En Message-style webservice modtager hele SOAP beskeden uden nogen form for forarbejdning, derved er det op til udvikleren at implementere logik der benytter informationerne der måtte være heri. Dette virker kun hvis klassen der implementere webservicen har helt bestemte metode signaturer, f.eks. en metode der tager et DOM dokument som argument og selv returnerer et DOM dokument. De to DOM dokumenter repræsenterer henholdsvis SOAP request og SOAP response beskeden.

Dette er skal ses i forhold til RPC-stil eller dokument-stil hvor Axis ville konstruere SOAP beskederne fra Java klasser og omvendt. I forbindelse med ReportService er det ikke ønskværdigt at konvertere et XML dokument der repræsenterer en rapport til Java klasser da alle operationer på rapport filerne udføres direkte på en DOM repræsentation af dokumentet.

Dette betyder i praksis at kun en metode i klassen kan nås og at denne afhængig af hvilken type besked den modtager må redelegere til den korrekte metode.

WSDL filen beskriver to metoder: getReportList og getReport. De er begge beskrevet i følgende udsnit af WSDL filen:

```
<wsdl:operation name="getReport">
   <wsdl:input message="pax:getReportRequest" name="getReportRequest" />
   <wsdl:output message="pax:getReportResponse" name="getReportResponse" />
</wsdl:operation>
<wsdl:operation name="getReportList">
   <wsdl:input message="pax:listReportRequest" name="listReportRequest" />
   <wsdl:output message="pax:listReportResponse" name="listReportResponse" />
</wsdl:operation>
```

Her defineres de to operationer og hvilket input de accepterer. Input og output elementerne refererer til deklarationer af SOAP beskeder i WSDL filen. Disse er eksempelvis defineret således:

```
<wsdl:message name="listReportResponse">
   <wsdl:part element="pax:reportListReturn" name="reportListReturn" />
</wsdl:message>
```

Message elementet fortæller at SOAP beskeden med navne listReportResponse indholder en enkelt del, nemlig et reportListReturn element. Dette element er specificeret i et XML Schema. Schemaet kan enten være i en ekstern fil der så inkluderes i WSDL filen eller være defineret i WSDL filen selv. I denne WSDL fil bruges begge. Selve rapport sproget er defineret i sin egen fil hvorimod strengarray samt heltalsargumenter er specificeret direkte i WSDL filen. reportListReturn elementet er defineret således:

```
<complexType name="ArrayOfString">
   <complexContent>
      <restriction base="soapenc:Array">
         <sequence>
            <element name="item" type="string" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
         </sequence>
         <attribute ref="soapenc:arrayType" wsdl:arrayType="string[]" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/>
      </restriction>
   </complexContent>
</complexType>
<element name="reportListReturn" type="pax:ArrayOfString"/>
```

Her defineres først en type der kan indeholde et array af strenge, herefter defineres elementet reportListReturn som værende af typen ArrayOfString. Operationer grupperes sammen til at danne en port.

Derved er er operationerne specificeret, men for at kunne kalde dem er det også nødvendigt at de associeres med en service samt at de bliver gjort tilgængelige enten gennem SOAP kald over HTTP eller Java. Dette opnås gennem bindinger, hvorved en port associeres med en eller flere måder at kalde denne operation på. getReport er f.eks. associeret til SOAP kald vha følgende binding.

```
<wsdl:binding name="PaxReportSoapBinding" type="pax:PaxReport">
<wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getReportList">
   <wsdlsoap:operation soapAction="getReportList" />
   <wsdl:input name="listReportRequest">
      <wsdlsoap:body namespace="http://paxflow.zurich.ibm.com" use="literal" />
   </wsdl:input>
```

```
   <wsdl:output name="listReportResponse">
      <wsdlsoap:body namespace="http://paxflow.zurich.ibm.com" use="
          literal" />
   </wsdl:output>
</wsdl:operation>
...
```

SOAP bindings elementet bestemmer transport protokollen der ønskes benyttet, i dette tilfælde HTTP, hvorefter SOAP bindingen til operationerne oprettes. Derefter mangler kun at tilbyde disse port bindinger til omverdenen gennem en service. En service tilbyder porte ved at liste dem således:

```
<wsdl:service name="PaxReportService">
   <wsdl:port binding="pax:PaxReportSoapBinding" name="PaxReport">
      <wsdlsoap:address location="http://ancona.zurich.ibm.com:9080/
          AxisTest/services/PaxFlow" />
   </wsdl:port>
...
```

Derved bliver SOAP bindingen tilbudt gennem PaxReportServicen.

For at kunne bruge servicen mangler der dog lidt endnu. Det er en (Web Service Deployment Descriptor) WSDD-fil der er specifik for Axis der specificerer hvilken klasse der tilbyder dette interface samt hvilke porte den skal tilbyde. Derudover bestemmes det også udfra WSDD-filen hvilken type service det er. I dette tilfælde er det message style.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://
    xml.apache.org/axis/wsdd/providers/java">
   <service name="PaxReport" provider="java:MSG" style="message">
      <parameter name="wsdlTargetNamespace" value="http://www.zurich.ibm
          .com/paxflow" />
      <parameter name="wsdlServicePort" value="PaxReport" />
      <parameter name="className" value="com.ibm.zurich.paxflow.
          reporting.PaxReportService" />
      <parameter name="wsdlPortType" value="PaxReport" />
      <parameter name="allowedMethods" value="method" />
      <parameter name="scope" value="Session" />
      <wsdlFile>/service.wsdl</wsdlFile>
...
```

Derudover definerer WSDD filen også "scope" for servicen. I dette tilfælde er den sat til session da der senere vil blive brug for at holde styr på brugere mv. WSDD filen kan ses i bilag D.3 på side 79.

Implementering i Java

ReportService klassen modtager kald fra Axis for hvert SOAP kald til PaxReport servicen. Ved hvert kald undersøges hvilken metode der blev kaldt gennem SOAP. Dette bruges til, via Javas Reflection API, at bestemme hvilken metode der skal kaldes på PaxFlowServiceImpl klassen.

PaxFlowServiceImlp indeholder to metoder: getReport og getReportList, præcis som defineret i WSDL filen. Klienter kan nu kalde webservicen med følgende kode:

```
Service service = new Service();
Call call = (Call) service.createCall();

//Hvor ligger servicen?
call.setTargetEndpointAddress(new URL("http://localhost:8080/AxisTest/
    services/PaxReport"));
```

```
6
7  //Hvilken metode vi vil have fat i
8  call.setOperationName(new QName("http://www.zurich.ibm.com/paxflow/chart
       ","getReport"));
9
10 //Kald servicen og læg output i et Document objekt
11 Document ret = (Document) call.invoke(new QName("http://www.zurich.ibm.
       com/paxflow/chart","getReportInput"),new Object[] { new Integer(2) })
       ;
```

Hele kildekoden til implementeringen i Java kan findes i bilag D.3 på side 84.

Implementering af PDFGenerator

PDFGenerator klassen konverterer en rapport XML-fil til PDF. PDFGenerator klassen har en instansmetode createPDFFromXML(Document dom, OutputStream stream) der som argument tager et DOM dokument indeholdende en rapport beskrivelse samt en OutputStream hvortil den færdige PDF-fil skal skrives. PDF-filen genereres ved at traversere DOM dokumentet og konstruere PDF delene efterhånden. Da PDFGeneratoren ikke har været det egentlige fokus i projektet vil den ikke blive beskrevet nærmere, men koden kan ses i bilag D.3.

5 Afprøvning

Det følgende afsnit beskriver en grundlæggende afprøvning der er lavet af transformeringen fra diagrambeskrivelsen til SVG. Der er under implementeringen af transformationerne og webservicen foretaget løbende tests. Da en udtømmende test af transformationerne såvel som webservicen vil være særdeles omfattende, foretages der kun en overfladisk afprøvning.

Før hver afprøvning opstilles der et krav til inputdata. Derefter gives der et eksempel på inputdata der opfylder kravene, og resultatet af transformeringen beskrives. De fleste af disse krav er opfyldt hvis inputdataen er validt ifølge Schema-filen.

5.1 Lagkagediagram

Krav

Inputdataen til lagkagediagrammet skal indeholde et `axis` element, der igen skal indeholdende et `xaxis` element. Barnet `max` af `xaxis` angiver summen af værdierne der indgår i lagkagediagrammet. Angives denne maxværdi forkert, vil lagkagediagrammet blive optegnet forkert.

Inputdataen skal indeholde et `plots` element der indeholder et `piechart`-element. Dette skal indeholde en reference til et `xaxis` element, samt en reference til et `data1d`-element. Det refererede `data1d`-element, skal indeholde en serie af `point1d`-elementer, der hver indeholder et `label` element der angiver punktets mærkat.

Resultat

I afsnit D.1 på side 48 i bilaget, findes et eksempel på inputdata der overholder de ovenstående krav. Resultatet af transformeringen kan ses på figur 9 på side 45 i bilaget.

Transformeringen af det viste inputdata forløb uden problemer.

5.2 Søjlediagram

Krav

Inputdata til søjlediagrammet skal indeholde et `axis`-element, der indeholder et `yaxis` element.

Inputdataen skal indeholde et `plots`-element, der igen indeholder it `barchart` element. Dette skal indeholde en reference til et `yaxis`-element, samt et `data1d`-element. `data1d`-elementet skal indeholde en serie af `point1d` elementer.

Resultat

I afsnit D.1 på side 50 i bilaget, findes et eksempel på inputdata der overholder de ovenstående krav. Resultatet af transformeringen kan ses på figur 10 på side 46 i bilaget.

Transformeringen af det inputdata til SVG forløb uden problemer

5.3 Punktdiagram

Krav

Inputdata til punktdiagrammet skal indeholder et `axis` element, der indeholder et `xaxis`-samt et `yaxis`-element. Inputdata skal indeholde et `lineplots`-element, der indeholder en serie af `lineplot`-elementer. Hver af disse skal have en reference til et `data2d`-element, et `xaxis`-element samt et `yaxis`-element.

Resultat

Der er lavet to afprøvninger af punktdiagrammet.

I afsnit D.1 på side 52 i bilaget, findes et eksempel på inputdata der overholder de ovenstående krav. Resultatet af transformeringen kan ses på figur 11 på side 46 i bilaget.

I afsnit D.1 på side 55 i bilaget, findes et eksempel på inputdata der overholder de ovenstående krav. Resultatet af transformeringen kan ses på figur 12 på side 47 i bilaget.

Begge transformeringer forløb uden problemer.

5.4 Webservice

Afprøvningen af foregår ved at sende en gyldig SOAP-forespørgsel til webservicen. Metoden der kaldes er getReport med et ID som argument. Forespørgslen kan ses på side 58 i afsnit D.1 i bilaget. Resultatet bør være en rapport XML-fil med indlejrede SVG diagrammer.

Resultat

Webservicen returnerer et korrekt udtryk i rapport-sproget, inklusiv et indlejret SVG-dokument. Svaret kan ses på side 60 i afsnit D.1 i bilaget.

5.5 Konklusion af afprøvningen

Afprøvningen af transformeringen er relativ overfladisk, og gør en del antagelser om input data. Det eneste der derfor kan siges ud fra afprøvningen er at transformeringerne tilsyneladende fungerer efter hensigten i de beskrevne eksempler.

Om webservicen kan man ud fra den noget overfladiske afprøvning konkludere at den på en gyldig forespørgsel returnere et tilsyneladende korrekt svar.

6 Diskussion og konklusion

De foregående afsnit har beskrevet de valg der er gjort for at nå frem til implementeringen. Da de valgte teknologier langt fra er de eneste der kan løse problemstillingerne i implementeringen, er der ingen garanti for at valgte er optimale. Dette afsnit diskutere hvorvidt de valgte teknologier har kunne leve op til forventningerne. Der gives en gennemgang af i hvilket omfang implementeringen lever op til kravspecifikationen, samt en diskussion af hvordan problemerne kunne være løst.

Følgende dele af implementeringen lever op til kravspecifikationen.

- Diagrammer
  - Lagkagediagrammer
  - Søjlediagrammer
  - Punktdiagrammer

Implementeringen understøtter ikke stablede søjlediagrammer. I den nuværende implementering stables flere søjlediagrammer ved at indsætte flere punkter med samme mærkat i plotdataen. Under implementeringen af dette viste det sig ikke at være muligt at udtrække mærkatinfomationen på en pålidelig måde. Den umiddelbare løsning på dette problem vil være at introducere et `barcharts` element der indeholder en serie af `barchart` elementer, hvor hele `barchart` opfattes som ét lag af det stablede søjlediagram.

Hvis en forkert maximumværdi angives for aksen i et lagkagediagram i den nuværende implementering, vil elementerne i lagkagediagrammet enten ikke nå hele vejen rundt eller begynde at overlappe hinanden. En mulig løsning er at lade stylesheetet udregne maximumværdien selv, men som tidligere nævnt vil det besværligegøre implementeringen. Den optimale løsning ville være hvis valideringen kunne fange denne fejl.

Oprindeligt var det tænkt at webservicen skulle ligge rundt om diagramgenerering, men med tiden blev det klart at det var mere oplagt at lægge webservicen rundt om rapportgenereringen i stedet. Dette ville give et mere overskueligt design og ville derved være nemmere at implementere.

Set i bakspejlet er webservices, særligt SOAP APIerne, ikke modne nok hvad angår dokumentation og stabilitet. Skulle vi starte forfra ville valget nok ikke være faldet på SOAP men i stedet på XML over HTTP.

Vi har været tilfredse med XSLT og har ikke haft noget behov for at lave transformeringen i Java. XSLT kræver lidt tilvænning da det er funktionsorienteret, men er yderst udtryksfyldt når man først har opnået fortrolighed med det.

Mange af de værdier der er tilknyttet `axis` burde leveres som parametre til transformeringen i stedet, således at den kontekstuelle data omkring hvordan data skulle vises helt blev adskilt fra data.

Alt i alt har vi været tilfredse med XSLT, hvorimod webservices ikke viste sig at være det bedste valg, og SVG har levet fuldt op til vores forventninger.

7 Perspektivering

Næste skridt bliver at finpudse implementeringen og tilpasse layout af diagrammerne, dernæst skrive en indføring til sproget på engelsk samt udvikle en fleksibel måde at definere layout for PDF-filerne på. Dette er et arbejde der vil foregå først i det nye år i samarbejde med IBM. Derefter overtager IBM helt implementeringen og benytter den i PaxFlow. Det hele skal være klar til 10. februar hvor PaxFlow for første gang skal vises offentligt.

Desuden forventes det at der udsendes en pressemeddelelse sidst i januar 2004, hvor PaxFlow annonceres. Dette forventes at resultere i mange henvendelser fra presse og branchen der ønsker at se hvad PaxFlow er. I den forbindelse er det også vigtigt at kunne fremvise et flot eksempel på en daglig rapport.

I et lidt større perspektiv vil det være en naturlig udvidelse af systemet, hvis det også kunne levere SVG-filer til mobile apparater. Langt de fleste mennesker har idag deres mobiltelefon med i lommen, så hvorfor ikke udnytte det, og præsentere rapporter på mobilen? Den tekniske udfordring består i at holde størrelsen på SVG filen nede, samt at de fleste implementeringer af SVG på mobile apparater ikke understøtter den fulde specifikation.

På længere sigt vil der øjensynligt komme yderligere diagramtyper til samt mulighed for interaktive diagrammer vha. scripting i SVG-filerne.

Figurer

Kodelister

A Referencer

[1] Apache axis. http://ws.apache.org/axis/.

[2] Basic research in computer science. http://www.brics.dk.

[3] Batik svg toolkit. http://xml.apache.org/batik/.

[4] Direct internet message encapsulation (dime). http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt.

[5] Document structure description 2.0. http://www.brics.dk/DSD/dsd2.html.

[6] Extensible markup language (xml) 1.0 (second edition). http://www.w3.org/TR/REC-xml.

[7] Extensible stylesheet language (xsl). http://www.w3.org/TR/2001/REC-xsl-20011015/.

[8] Java message service. http://java.sun.com/products/jms/docs.html.

[9] Oasis. http://www.oasis-open.org/.

[10] Recommendations for charts and graphics. http://www.sapdesignguild.org/resources/diagram_guidelines/diagram_guidelines.pdf.

[11] Relax. http://www.xml.gr.jp/relax/.

[12] Relax ng. http://relaxng.org/.

[13] Scalable vector graphics (svg) 1.1 specification. http://www.w3.org/TR/SVG/.

[14] Soap version 1.2. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[15] Svg 1.1: The elliptical arc curve commands. http://www.w3.org/TR/2000/CR-SVG-20001102/paths.html#PathDataEllipticalArcCommands.

[16] Trex. http://www.thaiopensource.com/trex/.

[17] Uddi version 3.0. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[18] Web services architecture. http://www.w3.org/TR/ws-arch.

[19] Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.

[20] Ws-attachments. http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-soap-01.txt.

[21] Xalan-java. http://xml.apache.org/xalan-j/.

[22] Xml path language (xpath). http://www.w3.org/TR/xpath.

[23] Xml schema. http://www.w3.org/XML/Schema.

[24] Xsl transformations (xslt). http://www.w3.org/TR/1999/REC-xslt-19991116/.

[25] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. In *Proceedings of the ACM Symposium on Operating System Principles*, page 3, Bretton Woods, NH, 1983. Association for Computing Machinery. http://citeseer.nj.nec.com/birrell84implementing.html.

[26] Matthew J. Duftler, Nirmal K. Mukhi, Aleksander Slominski, and Sanjiva Weerawarana. Web services invocation framework (wsif), 2001. http://www.research.ibm.com/people/b/bth/OOWS2001/duftler.pdf.

[27] Will Provost. Uml for w3c xml schema design. http://www.xml.com/pub/a/2002/08/07/wxs_uml.html, 2002.

B Supplerende Litteratur

[28] Alan Brown, Simon Johnston, and Kevin Kelly. Using service-oriented architecture and component based development to build web services applications.

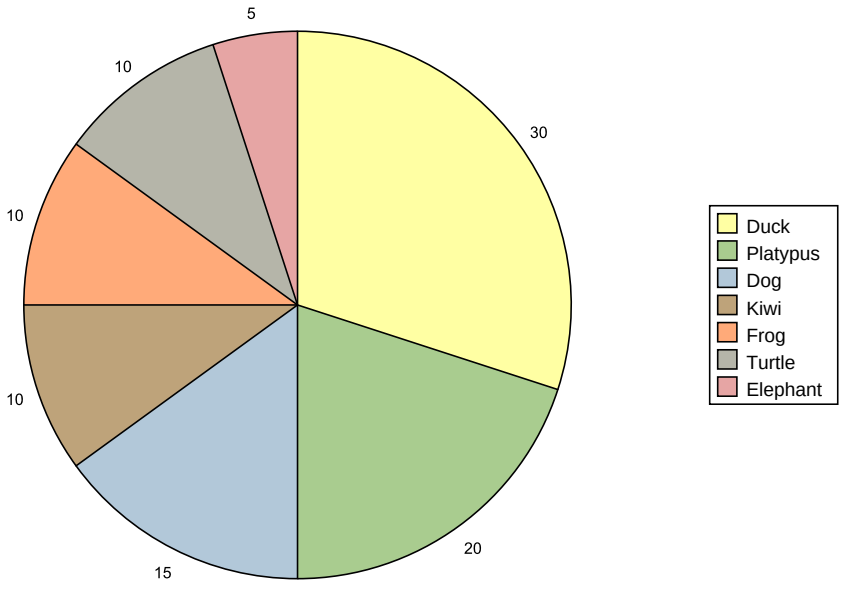
[29] James Clark. http://web.archive.org/web/20021017092150/http://www.imc.org/ietf-xml-use/mail-archive/msg00217.html.

[30] Anders Møller. Problems with xml schema. http://www.brics.dk/~amoeller/XML/schemas/xmlschema-problems.html.

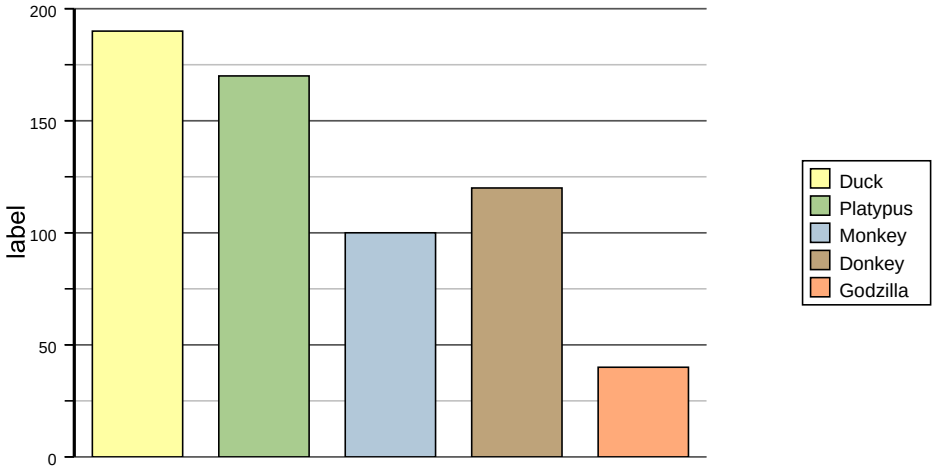
[31] Eric van der Vlist. Comparing xml schema languages, 2001. http://www.xml.com/pub/a/2001/12/12/schemacompare.html.

C Afprøvning

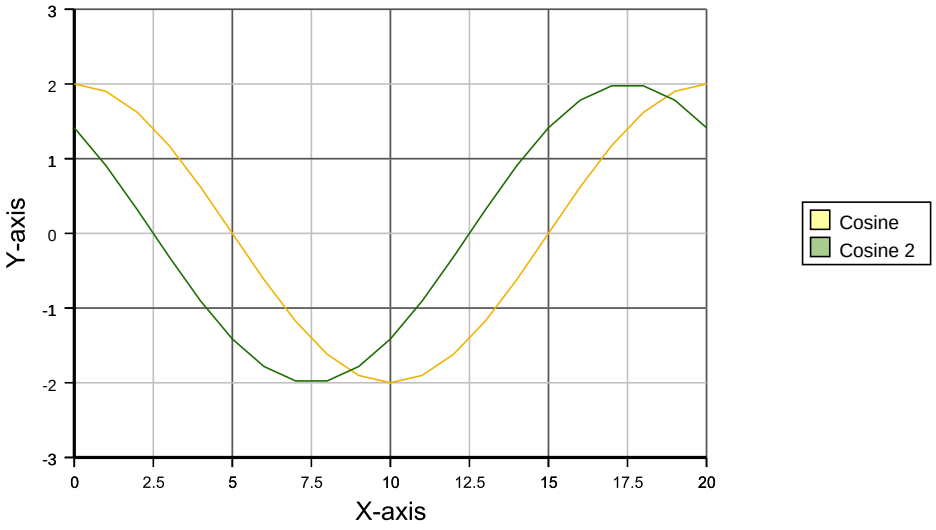
De følgende figure er resultatet af brugertesten. Inputdataen



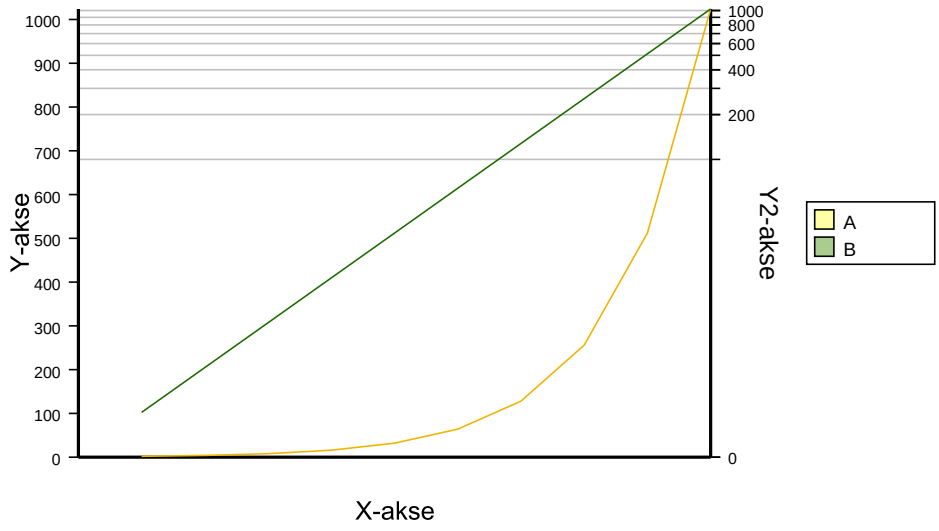
Figur 9: Et lagkagediagram



Figur 10: Et søjlediagram



Figur 11: Et punktdiagram indeholdende to `lineplot`-elementer



Figur 12: Et punktdiagram med både liniær og logaritmiske akser

D Kildekode

D.1 Tests

Lagkagediagram

Kildekoden følger på næste side

piechart_data.xml


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    (C) Copyright IBM Corp. 2003

    The source code for this program is not published or otherwise
    divested of its trade secrets, irrespective of what has
    been deposited with the U.S. Copyright Office.

    IBM Zurich Research Laboratory
-->
<chart xmlns="http://zurich.ibm.com/paxflow/chart" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://zurich.ibm.com/paxflow/chart
 data.xsd ">
    <axes>
        <xaxis id="axis">
            <label>Flow of Pax</label>
            <min>0.0</min>
            <max>100</max>
        </xaxis>
    </axes>
    <plots>
        <piechart axisref="axis" dataref="data" />
    </plots>
    <plotdata>
        <data1d id="data">
            <point1d>
                <label>Duck</label>
                <x>30</x>
            </point1d>
            <point1d>
                <label>Platypus</label>
                <x>20</x>
            </point1d>
            <point1d>
                <label>Dog</label>
                <x>15</x>
            </point1d>
            <point1d>
                <label>Kiwi</label>
                <x>10</x>
            </point1d>
            <point1d>
                <label>Frog</label>
                <x>10</x>
            </point1d>
            <point1d>
                <label>Turtle</label>
                <x>10</x>
            </point1d>
            <point1d>
                <label>Elephant</label>
                <x>5</x>
            </point1d>
        </data1d>
    </plotdata>
</chart>
```

Søjlediagram

Kildekoden følger på næste side

barchart_data.xml


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    (C) Copyright IBM Corp. 2003

    The source code for this program is not published or otherwise
    divested of its trade secrets, irrespective of what has
    been deposited with the U.S. Copyright Office.

    IBM Zurich Research Laboratory
-->
<chart xmlns="http://zurich.ibm.com/paxflow/chart" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://zurich.ibm.com/paxflow/chart
 ../source/data.xsd ">
    <axes>
        <yaxis id="axis">
            <label>label</label>
            <min>0</min>
            <max>200</max>
            <interval>
                <value>50</value>
                <extend>true</extend>
                <showlabel>true</showlabel>
            </interval>
            <subinterval>
                <value>25</value>
                <extend>true</extend>
                <showlabel>false</showlabel>
            </subinterval>
        </yaxis>
    </axes>
    <plots>
        <barchart axisref="axis" dataref="barchartdata" />
    </plots>
    <plotdata>
        <data1d id="barchartdata">
            <point1d>
                <label>Duck</label>
                <x>190</x>
            </point1d>
            <point1d>
                <label>Platypus</label>
                <x>170</x>
            </point1d>
            <point1d>
                <label>Monkey</label>
                <x>100</x>
            </point1d>
            <point1d>
                <label>Donkey</label>
                <x>120</x>
            </point1d>
            <point1d>
                <label>Godzilla</label>
                <x>40</x>
            </point1d>

        </data1d>
    </plotdata>
</chart>
```

Punktdiagram - cosinus

Kildekoden følger på næste side

cosine_data.xml


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    (C) Copyright IBM Corp. 2003

    The source code for this program is not published or otherwise
    divested of its trade secrets, irrespective of what has
    been deposited with the U.S. Copyright Office.

    IBM Zurich Research Laboratory
-->
<chart xmlns="http://zurich.ibm.com/paxflow/chart" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://zurich.ibm.com/paxflow/chart
 ../source/data.xsd ">
    <axes>
        <xaxis id="x">
            <label>X-axis</label>
            <min>0</min>
            <max>20</max>
            <interval>
                <value>5</value>
                <extend>true</extend>
                <showlabel>true</showlabel>
            </interval>
            <subinterval>
                <value>2.5</value>
                <extend>true</extend>
                <showlabel>true</showlabel>
            </subinterval>
        </xaxis>
        <yaxis id="y">
            <label>Y-axis</label>
            <min>-3</min>
            <max>3</max>
            <interval>
                <value>2</value>
                <extend>true</extend>
                <showlabel>true</showlabel>
            </interval>
            <subinterval>
                <value>1</value>
                <extend>true</extend>
                <showlabel>true</showlabel>
            </subinterval>
        </yaxis>
    </axes>
    <plots>
        <lineplots>
            <lineplot dataref="cos1" xaxisref="x" yaxisref="y" />
            <lineplot dataref="cos2" xaxisref="x" yaxisref="y" />
        </lineplots>
    </plots>
    <plotdata>
        <data2d id="cos1">
            <label>Cosine</label>
            <point2d>
                <x>0</x>
                <y>2</y>
            </point2d>
            <point2d>
                <x>1</x>
                <y>1.90211303259</y>
            </point2d>
            <point2d>
```

cosine_data.xml


```
                <x>2</x>
                <y>1.61803398875</y>
            </point2d>
            <point2d>
                <x>3</x>
                <y>1.17557050458</y>
            </point2d>
            <point2d>
                <x>4</x>
                <y>0.61803398875</y>
            </point2d>
            <point2d>
                <x>5</x>
                <y>0</y>
            </point2d>
            <point2d>
                <x>6</x>
                <y>-0.61803398875</y>
            </point2d>
            <point2d>
                <x>7</x>
                <y>-1.17557050458</y>
            </point2d>
            <point2d>
                <x>8</x>
                <y>-1.61803398875</y>
            </point2d>
            <point2d>
                <x>9</x>
                <y>-1.90211303259</y>
            </point2d>
            <point2d>
                <x>10</x>
                <y>-2</y>
            </point2d>
            <point2d>
                <x>11</x>
                <y>-1.90211303259</y>
            </point2d>
            <point2d>
                <x>12</x>
                <y>-1.61803398875</y>
            </point2d>
            <point2d>
                <x>13</x>
                <y>-1.17557050458</y>
            </point2d>
            <point2d>
                <x>14</x>
                <y>-0.61803398875</y>
            </point2d>
            <point2d>
                <x>15</x>
                <y>-0</y>
            </point2d>
            <point2d>
                <x>16</x>
                <y>0.61803398875</y>
            </point2d>
            <point2d>
                <x>17</x>
                <y>1.17557050458</y>
            </point2d>
```

cosine_data.xml Page 3/4

```
            <point2d>
                <x>18</x>
                <y>1.61803398875</y>
            </point2d>
            <point2d>
                <x>19</x>
                <y>1.90211303259</y>
            </point2d>
            <point2d>
                <x>20</x>
                <y>2</y>
            </point2d>
        </data2d>
        <data2d id="cos2">
            <label>Cosine 2</label>
            <point2d>
                <x>0</x>
                <y>1.41421356237</y>
            </point2d>
            <point2d>
                <x>1</x>
                <y>0.90798099948</y>
            </point2d>
            <point2d>
                <x>2</x>
                <y>0.31286893008</y>
            </point2d>
            <point2d>
                <x>3</x>
                <y>-0.31286893008</y>
            </point2d>
            <point2d>
                <x>4</x>
                <y>-0.90798099948</y>
            </point2d>
            <point2d>
                <x>5</x>
                <y>-1.41421356237</y>
            </point2d>
            <point2d>
                <x>6</x>
                <y>-1.78201304838</y>
            </point2d>
            <point2d>
                <x>7</x>
                <y>-1.97537668119</y>
            </point2d>
            <point2d>
                <x>8</x>
                <y>-1.97537668119</y>
            </point2d>
            <point2d>
                <x>9</x>
                <y>-1.78201304838</y>
            </point2d>
            <point2d>
                <x>10</x>
                <y>-1.41421356237</y>
            </point2d>
            <point2d>
                <x>11</x>
                <y>-0.90798099948</y>
            </point2d>
```

cosine_data.xml Page 4/4

```
            <point2d>
                <x>12</x>
                <y>-0.31286893008</y>
            </point2d>
            <point2d>
                <x>13</x>
                <y>0.31286893008</y>
            </point2d>
            <point2d>
                <x>14</x>
                <y>0.90798099948</y>
            </point2d>
            <point2d>
                <x>15</x>
                <y>1.41421356237</y>
            </point2d>
            <point2d>
                <x>16</x>
                <y>1.78201304838</y>
            </point2d>
            <point2d>
                <x>17</x>
                <y>1.97537668119</y>
            </point2d>
            <point2d>
                <x>18</x>
                <y>1.97537668119</y>
            </point2d>
            <point2d>
                <x>19</x>
                <y>1.78201304838</y>
            </point2d>
            <point2d>
                <x>20</x>
                <y>1.41421356237</y>
            </point2d>
        </data2d>
    </plotdata>
</chart>
```

Punktdiagram - log

Kildekoden følger på næste side

exp_data.xml

Page 1/3

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    (C) Copyright IBM Corp. 2003

    The source code for this program is not published or otherwise
    divested of its trade secrets, irrespective of what has
    been deposited with the U.S. Copyright Office.

    IBM Zurich Research Laboratory
-->
<chart xmlns="http://zurich.ibm.com/paxflow/chart" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://zurich.ibm.com/paxflow/chart
 ../source/data.xsd ">
    <axes>
        <xaxis id="x">
            <label>X-akse</label>
            <min>0</min>
            <max>10</max>
        </xaxis>
        <yaxis id="y">
            <label>Y-akse</label>
            <min>0</min>
            <max>1024</max>
            <interval>
                <value>100</value>
                <extend>false</extend>
                <showlabel>true</showlabel>
            </interval>
        </yaxis>
        <yaxis id="y2">
            <label>Y2-akse</label>
            <min>0</min>
            <max>1024</max>
            <interval>
                <value>200</value>
                <extend>false</extend>
                <showlabel>true</showlabel>
            </interval>
            <subinterval>
                <value>100</value>
                <extend>true</extend>
                <showlabel>false</showlabel>
            </subinterval>
            <type>log</type>
        </yaxis>
    </axes>
    <plots>
        <lineplots>
            <!-- Plot the data on a liniar y-axis -->
            <lineplot dataref="exp" xaxisref="x" yaxisref="y" />
            <!-- Plot the data on a logarithmic y-axis -->
            <lineplot dataref="lin" xaxisref="x" yaxisref="y2" />
        </lineplots>
    </plots>
    <plotdata>
        <data2d id="exp">
            <label>A</label>
            <point2d>
                <x>1</x>
                <y>2</y>
            </point2d>
            <point2d>
                <x>2</x>
```

exp_data.xml

Page 2/3

```
                <y>4</y>
            </point2d>
            <point2d>
                <x>3</x>
                <y>8</y>
            </point2d>
            <point2d>
                <x>4</x>
                <y>16</y>
            </point2d>
            <point2d>
                <x>5</x>
                <y>32</y>
            </point2d>
            <point2d>
                <x>6</x>
                <y>64</y>
            </point2d>
            <point2d>
                <x>7</x>
                <y>128</y>
            </point2d>
            <point2d>
                <x>8</x>
                <y>256</y>
            </point2d>
            <point2d>
                <x>9</x>
                <y>512</y>
            </point2d>
            <point2d>
                <x>10</x>
                <y>1024</y>
            </point2d>
        </data2d>
        <data2d id="lin">
            <label>B</label>
            <point2d>
                <x>1</x>
                <y>2</y>
            </point2d>
            <point2d>
                <x>2</x>
                <y>4</y>
            </point2d>
            <point2d>
                <x>3</x>
                <y>8</y>
            </point2d>
            <point2d>
                <x>4</x>
                <y>16</y>
            </point2d>
            <point2d>
                <x>5</x>
                <y>32</y>
            </point2d>
            <point2d>
                <x>6</x>
                <y>64</y>
            </point2d>
            <point2d>
                <x>7</x>
```

exp_data.xml Page 3/3

```
                <y>128</y>
            </point2d>
            <point2d>
                <x>8</x>
                <y>256</y>
            </point2d>
            <point2d>
                <x>9</x>
                <y>512</y>
            </point2d>
            <point2d>
                <x>10</x>
                <y>1024</y>
            </point2d>
        </data2d>
    </plotdata>
</chart>
```

SOAP-request

Den følgende side indeholder et eksempel på en forespørgsel sendt til webservicen.

request.xml Page 1/1

```
POST /AxisTest/services/PaxReport HTTP/1.0

Content-Type: text/xml; charset=utf-8

Accept: application/soap+xml, application/dime, multipart/related, text/*

User-Agent: Axis/1.1

Host: 127.0.0.1

Cache-Control: no-cache

Pragma: no-cache

SOAPAction: "getReport"

Content-Length: 477



<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance">
 <soapenv:Body>
  <ns1:getReportInput soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/enc
oding/" xmlns:ns1="http://www.zurich.ibm.com/paxflow/chart">
   <ns1:arg0 xsi:type="xsd:int">2</ns1:arg0>
  </ns1:getReportInput>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP-reponse

Den følgende side indeholder et eksempel på et svar fra webservicen.

response.xml


```
HTTP/1.1 200 OK

Server: WebSphere Application Server/5.0

Expires: Thu, 01 Dec 1994 16:00:00 GMT

Set-Cookie: JSESSIONID=00007r17O6aUF4otuG-9WUafLqd:-1;Path=/

Cache-Control: no-cache="set-cookie,set-cookie2"

Content-Type: text/xml; charset=utf-8

Content-Language: en-GB

Connection: close



<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance">
 <soapenv:Body><report>
    <id>com.ibm.zurich.paxflow.reports.TerminalBRetail</id>
    <title>Daily summary for Terminal B, 24. December 2032</title>
    <section>
        <title>Percent Pax pr geography</title>
        <p/>
        <svg height="400" width="600" xmlns="http://www.w3.org/2000/svg" xmlns:m
ath="http://exslt.org/math" xmlns:pc="http://zurich.ibm.com/paxflow/chart"><g tr
ansform="translate(10,10)"><g transform="translate(247.5, 190)" xmlns="" xmlns:j
ava="http://xml.apache.org/xslt/java"><path d="M 0,0 L -0.000000000000031412190
39812961,-171 A171,171 0 0 1 135.99508098775593 -103.6645452752951" style="fill:r
gb(255,248,163); stroke:none; stroke-width: 0;"/><path d="M 0,0 L 135.9950809877
5593,-103.6645452752951 A171,171 0 0 1 143.53148786856343 92.95005105020884" sty
le="fill:rgb(169,204,143); stroke:none; stroke-width: 0;"/><path d="M 0,0 L 143.
53148786856352,92.9500510502087 A171,171 0 0 1 38.962950022436665 166.5019174831
002" style="fill:rgb(178,200,217); stroke:none; stroke-width: 0;"/><path d="M 0,
0 L 38.962950022436665,166.5019174831002 A171,171 0 0 1 -164.88734772248708 45.3
11836875629304" style="fill:rgb(190,163,122); stroke:none; stroke-width: 0;"/><p
ath d="M 0,0 L -164.88734772248714,45.31183687562901 A171,171 0 0 1 0.0000000000
0023046190860847373 -171" style="fill:rgb(243,170,121); stroke:none; stroke-widt
h: 0;"/><path d="M 0,0 L -0.000000000000031412190 39812961,-171 A171,171 0 0 1 13
5.99508098775593 -103.6645452752951" style="fill:none; stroke:black; stroke-widt
h: 1;"/><text font-size="10" text-anchor="middle" x="82.53188984325449" y="-166.
68679359475635"><tspan style="baseline-shift:-50%;">1.5</tspan></text><path d="M
 0,0 L 135.99508098775593,-103.6645452752951 A171,171 0 0 1 143.53148786856343 9
2.95005105020884" style="fill:none; stroke:black; stroke-width: 1;"/><text font-
size="10" text-anchor="middle" x="185.86350967545067" y="-7.124308466346636"><ts
pan style="baseline-shift:-50%;">2</tspan></text><path d="M 0,0 L 143.5314878685
6352,92.9500510502087 A171,171 0 0 1 38.962950022436665 166.5019174831002" style
="fill:none; stroke:black; stroke-width: 1;"/><text font-size="10" text-anchor="
middle" x="107.00927080066354" y="152.1348611026094"><tspan style="baseline-shif
t:-50%;">1.25</tspan></text><path d="M 0,0 L 38.962950022436665,166.501917483100
2 A171,171 0 0 1 -164.88734772248708 45.311836875629304" style="fill:none; strok
e:black; stroke-width: 1;"/><text font-size="10" text-anchor="middle" x="-95.049
4743771624" y="159.8799468996195"><tspan style="baseline-shift:-50%;">2.5</tspan
></text><path d="M 0,0 L -164.88734772248714,45.31183687562901 A171,171 0 0 1 0.
0000000000002304619086084737 3 -171" style="fill:none; stroke:black; stroke-width
: 1;"/><text font-size="10" text-anchor="middle" x="-147.92447405685718" y="-112
.75792643979483"><tspan style="baseline-shift:-50%;">3</tspan></text></g></g><g
transform="translate(515,10)">Flow of Pax0.010.25<g transform="translate(0,150)"
 xmlns=""><rect fill="none" height="90" stroke="blue" width="80" x="0" y="-5"/><
```

response.xml


```
rect fill="rgb(255,248,163)" height="12" stroke="black" stroke-width="1" width="
12" x="5" y="0"/><text font-size="12" x="23" y="12">Mads</text><rect fill="rgb(1
69,204,143)" height="12" stroke="black" stroke-width="1" width="12" x="5" y="17"
/><text font-size="12" x="23" y="29">Thomas</text><rect fill="rgb(178,200,217)"
height="12" stroke="black" stroke-width="1" width="12" x="5" y="34"/><text font-
size="12" x="23" y="46">Ulrich</text><rect fill="rgb(190,163,122)" height="12" s
troke="black" stroke-width="1" width="12" x="5" y="51"/><text font-size="12" x="
23" y="63">Ulf</text><rect fill="rgb(243,170,121)" height="12" stroke="black" st
roke-width="1" width="12" x="5" y="68"/><text font-size="12" x="23" y="80">Linda
</text></g>Mads1.5Thomas2Ulrich1.25Ulf2.5Linda3</g></svg>
    </section>
    <section>
        <title>Average Pax Waiting Time in Queues</title>
        <p>Following <b><emph><b>chart</b></emph></b> shows the average waiting
time spent in queues pr Pax type. If avg is above 30 min it basically means thom
as was the one serving them..</p>
        <chart>
            <data id="com.ibm.zurich.paxflow.reports.PaxGeoPie"/>
        </chart>
    </section>
</report> </soapenv:Body>
</soapenv:Envelope>
```

E Resumé

Vi føler at vi har nået vores mål med projektet. Vi har udviklet fundamentet til en fleksibel diagrammeringsløsning baseret på åbne standarder. Implementeringen vil bliver finpudset først i det nye år, hvorefter den overdrages til IBM.

På det rent faglige plan har vi tilegnet os en god portion viden om XSLT, SVG, SOAP og PDF. Vi er overbeviste om at det er en viden vi vil drage stor nytte af i fremtidige sammenhænge. Rent projektmæssigt har arbejdet forløbet godt, det har været rart at have et kontor, fast plads og faste arbejdstider. Dette har også gjort at opholdet i Schweiz har kunnet byde på andet end blot projektskrivning.

Infrastrukturen omkring projektet har fungeret fortrinneligt. Vi har fra tidligere projekter både rapport skabeloner, cvs server, mailing lister og IDE. Rapporten er lavet i LaTeXog udviklingen er fortrinsvis foregået i WebSphere Application Developer, en IBM branded version af Eclipse med ekstra plugins til alt mellem himmel og jord.

På trods af at gruppen har mange semesterprojekter på bagen, må vi erkende, at vi endnu engang har været lige ambitiøse nok i forhold til, hvormeget man rent faktisk kan nå på 4 uger. Dette har desværre medfødt at vores afprøvnings afsnit ikke er grundigt nok til at kunne konkludere noget udfra.

Men ved at lave projektet i samarbejde med IBM har projektet fået tilført en ekstra dimension da der pludselig indgår fremmedord som politik og rigtige brugeres krav. Det har øget kravene til projektet, men samtidig motiveret. De første tilbagemeldinger fra IBM er at de ønsker at bruge løsningen i andre sammenhænge så snart vi har pudset løsningen af. Desuden har det for 2 af gruppens medlemmer været det første møde med arbejdsgangene i en stor international virksomhed, til både morskab og skræk.