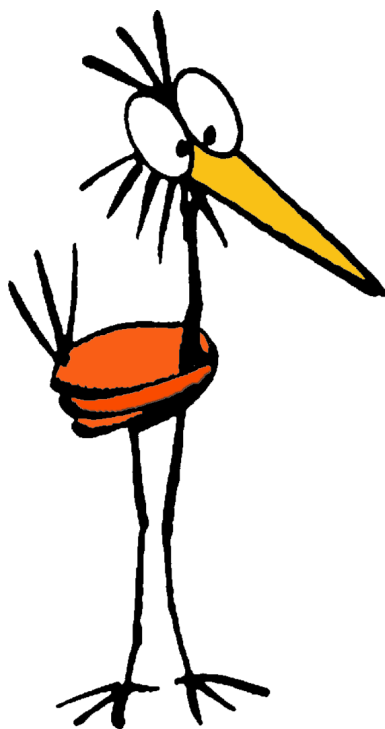


Automatisk validering af webbaserede formularer



Mikkel Ricky
Maj 2002

Tak til Helle Markmann for flittig korrekturlæsning og hårde – men retfærdige – kommentarer til både form (!) og indhold. Der skal også lyde en stor tak til kommatolog Anders Labich.

May the forms be with you!

Indhold

1	Indledning	7
1.1	En formulars livscyklus	7
1.2	Menu	8
1.3	Brugervenlighed	8
1.4	Terminologi	8
2	Overblik	11
2.1	En illustrativ formular	11
2.2	XForms	13
2.2.1	Design	14
2.2.2	Brugerinteraktion	18
2.2.3	Eksemplet	20
2.2.4	Indtryk	24
2.3	Extensible Forms Description Language	25
2.3.1	Design	25
2.3.2	Brugerinteraktion	28
2.3.3	Eksemplet	29
2.3.4	Indtryk	33
2.4	Mozquito	34
2.5	Konklusion	35
3	PowerForms	37
3.1	Introduktion	37
3.2	Design	38
3.3	Validering	39
3.3.1	Constraints	39
3.3.2	Evaluering af udtryk	40
3.4	Brugerinteraktion	42
3.4.1	Visualisering	42

3.4.2	Automatisk fuldførelse	45
3.4.3	Submit	47
3.5	Eksemplet	47
3.6	PowerForms i JWIG	51
3.6.1	Serverside-validering	52
3.6.2	Et eksempel	54
3.7	Implementation	56
3.7.1	Evaluering	58
3.7.2	Teknologier	61
3.7.3	Kodegenerering	63
4	PowerForms i den virkelige verden	65
4.1	Eksempel 1: Extensible Forms Description Language	66
4.1.1	Constraints	67
4.1.2	Resultat	68
4.2	Eksempel 2: Java Developer Connection	69
4.2.1	Constraints	70
4.2.2	Resultat	71
4.3	Eksempel 3: Barnets bog	72
4.3.1	Constraints	73
4.3.2	Resultat	74
5	PowerForms-manual	75
5.1	Overblik	75
5.2	Constraints	76
5.3	Udtryk	77
5.4	Regulære udtryk	78
5.5	Initialisering	80
5.6	Status	80
5.7	Automatisk fuldførelse	81
5.8	Indstillinger	81
5.9	Udvidelser af HTML	82
5.9.1	Submit	82
5.9.2	Status	82
5.9.3	Andet	84
6	Afslutning	85

A	Det illustrative eksempel	87
A.1	XFDL i New York	88
A.2	XForms i New York	90
A.3	PowerForms i New York	93
B	PowerForms i den virkelige verden	96
B.1	Extensible Forms Description Language	97
B.2	Barnets bog	100
B.3	Java Developer Connection	102
C	PowerForms i JWIG	105
D	PowerForms-grammatik	107

Kapitel 1

Indledning

Nærværende dokument er et speciale i datalogi, og som titlen antyder, vil emnet for teksten være validering af webbaserede formularer, og hvad dette nærmere dækker over, vil blive beskrevet om få øjeblikke.

1.1 En formulars livscyklus

En formulars liv falder i tre faser:

1. Den hentes – enten på internettet eller på eksempelvis skatteforvaltningen – af ham der skal udfylde.
2. Den udfyldes.
3. Den afleveres.

Der er ikke meget der kan gå galt i den første fase, så der springes straks til udfyldelsesfasen. I denne fase skal man sørge for at få udfyldt formularen på en konsistent måde, så det ikke både fremgår at man er enke og at ens kone i øvrigt hedder Beate Jørgensen. Når man én gang har prøvet at udfylde en formular af bedste bureaukratiske skuffe, ved man hvor svært det kan være at forstå hvad der egentlig spørges om og hvordan man kan svare uden at komme i modstrid med det kryds man har sat i rubrikken ovenfor.

Når en fysisk formular er blevet udfyldt og derefter afleveres, foregår det ofte med en vis nervøsitet, idet der kan være forbløffende lang udrykningstid på at få meddelelse om at man har glemt at anføre sit kontonummer og derfor ikke får sin SU rettidigt. Hvis man er rigtig uheldig bliver formularen bare kasseret som værende ugyldig grundet det manglende kontonummer ...

Det er oplagt at den store fejkilde i forbindelse med formularer, er den der skal udfylde den, nemlig *brugeren*. Dels er det ikke altid klart hvad bureaukraten, der har udformet formularen, spørger om, og hvis det er klart hvad der spørges om, er det ikke altid oplagt hvad der må svares. Endelig findes der brugere der vil gribe enhver chance for at misforstå meningen med en formular og i deres fornøjelses hellige navn udfylde den så forkert som overhovedet muligt.

Derfor er det nødvendigt at kontrollere en brugers udfyldning af en formular, både for hans egen skyld og for bureaukratens skyld.

I en elektronisk, webbaseret formular er det muligt at lave alverdens kontrol af brugerens udfyldning af formularen, således at brugeren kan få oplysninger om at hans udfyldning er forkert eller inkonsistent, og traditionelt foregår den slags kontrol, ved at en programmør håndkoder nogle valideringsfunktioner der kan afgøre om en formular er udfyldt som krævet.

Dette speciale vil beskæftige sig med hvordan man kan automatisere denne form for validering og derved gøre livet nemmere for programmøren, men også for brugeren, som vil blive hjulpet til at udfylde en formular som programmøren forventer. Selve udfærdigelsen af et fornuftigt skema vil ikke blive berørt, men bør overlades til et fornuftigt menneske uden bureaukratisk uddannelse.

1.2 Menu

Der vil i kapitel 2 blive lagt for med et vue over hvordan verdens tilstand er i forbindelse med design og brug af webbaserede formularer der understøtter validering, og nogle udvalgte forslag vil blive beskrevet med henblik på at afsløre styrker og (eventuelle) svagheder.

Når det eksisterende terræn er blevet sonderet, vil kapitel 3 introducere PowerForms, som er et forslag der endnu ikke er kommet den store verden til kendskab. PowerForms er et værktøj der gennem de sidste par år er blevet udviklet på Brics, og det er resultaterne af dette arbejde der præsenteres i nærværende speciale.

I modsætning til de forslag der beskrives i kapitel 2, går PowerForms målrettet efter at gøre det nemt at lave automatisk validering af værdier på webbaserede formularer og understøtter ikke andre aspekter af webbaserede formularer. Der vil blive givet talrige eksempler på hvordan PowerForms bruges, og det vil blive demonstreret hvorledes allerede eksisterende webformularer vil kunne have glæde af validering ved brug af PowerForms.

1.3 Brugervenlighed

Som nævnt er det ikke alle aspekter af webbaserede formularer der vil blive beskrevet i stor detalje, så for at gøre det nemt for den interesserede læser at fordybe sig i detaljer er en række relevante links samlet på websiden <http://www.brics.dk/~ricky/speciale/>.

1.4 Terminologi

Som det ikke kan være forbigået den bare lidt opmærksomme læser er, dette speciale affattet på dansk. Som en datalogisk skolet læser vil vide, har netop dette sprog voldt en del kvaler indenfor datalogisk sprogbrug, hvor den specielle dialekt *danglish* har vundet indpas. Der er i tidens løb gjort mange forsøg på at oversætte alle engelske udtryk til dansk, og ofte er det gået godt, men andre gange har oversættelserne været *for* konstruerede til lejligheden og er ikke blevet alment accepteret.

For at undgå misforståelser på grund af uklare begreber opremses en liste af ord og den betydning de er tillagt i denne tekst. Listen skal ikke tages som et udtryk for hvad der er rigtigt og forkert, men som et udtryk for at der er gjort bevidste overvejelser over sproget ud fra devisen: „konsekvens frem for korrekthed!“

Bruger Den person der får vist en webbaseret formular i sin browser og efterfølgende udfylder den efter bedste evne.

Browser Et program man bruger til at surfe rundt på internettet.

Formular Indeholder en samling af felter med tilhørende kontrolelementer og andre elementer der præsenterer formularen for en bruger.

Webservice En eller flere interaktioner mellem en webserver og en bruger.

Submit Når man er færdig med at udfylde en formular, „submitter“ man den.

Kontrollement Et element på en formular hvor man kan indtaste eller vælge blandt en række værdier.

Felt En samling af ét eller flere kontrolelementer med samme navn på en formular.

Constraint En betingelse der udtaler sig om ét eller flere felter på en formular og om hvad der skal gælde for værdierne i felterne. Flertalsformen af „constraint“ er „constraints“.

Server En computer der kan levere elektroniske, webbaserede formularer.

Programmør Den person der designer en elektronisk formular og hvad der hører til formularen.

Tekstfelt Et kontrolelement hvori der kan indtastes én linje tekst.

Radioknap Et kontrolelement i en gruppe af kontrolelementer og hvor højst ét af elementerne i gruppen kan være markeret.

Afkrydsningsfelt Et kontrolelement der kan være markeret eller ikke markeret.

Liste Et kontrolelement, hvori man enten kan vælge én værdi, eller nul eller flere værdier.

Script Et program der kan fortolkes af en browser ved brug af en passende JavaScript-implementation i browseren.

ECMAScript Et scriptsprog der definerer grundliggende primitiver, kontrolstruktur og klasser i JavaScript.

JavaScript Det scriptsprog der findes i browsere. JavaScript indeholder udover (en del af) ECMAScript, funktioner til at manipulere et HTML-dokument der vises i browseren. Det er udelukkende af historiske årsager at det hedder „JavaScript“, og sproget har ikke noget med Java at gøre.

Kapitel 2

Overblik

HTML-formularer har stort set ikke udviklet sig siden 1992, hvor de blev beskrevet i den første version af en HTML-specifikation [26]. Siden da er der skrevet metervis af scriptkode til at validere input fra brugere, men det er først for nylig, at man er begyndt at overveje om ikke det er på tide at der skal (nye) boller på suppen så webbaserede formularer bliver mere udtryksfulde. Der er rig mulighed for at forbedre elektroniske webformularer som vi kender dem i dag, og i det følgende vil nogle moderne forslag til design og validering af elektroniske webformularer for øje blive beskrevet.

Beskrivelsen omfatter *XForms* [afsnit 2.2] og *Extensible Forms Definition Language* [afsnit 2.3], som begge tager hånd om hele formularbegrebet og altså ikke kun den del der har med validering at gøre. Eftersom emnet for denne tekst er validering, vil hovedvægten imidlertid blive lagt på mulighederne for netop dette, mens andre muligheder i de beskrevne forslag kun vil blive beskrevet kort.

2.1 En illustrativ formular

For at kunne afsløre styrker og svagheder i de forskellige forslag, er det rimeligt at lade dem løse den samme opgave, eller i hvert fald forsøge, og „opgaven“ er formularen der er vist i figur 1, som den ser ud i HTML.

Umiddelbart ser formularen ganske tilforladelig ud, men under overfladen gemmer sig en række krav man rimeligvis kan stille til værdier i de enkelte felter og til sammenhænge mellem dem. Det drejer sig om følgende:

- I e-mail-feltet skal der indtastes en korrekt e-mail-adresse. Bemærk at der ikke gøres forhåbninger om at afgøre hvorvidt den indtastede e-mail-adresse rent faktisk findes, men blot at den overholder formatet for lovlige e-mail-adresser [14]. Det må være op til brugerens samvittighed – eller mangel på samme – at indtaste sin egen e-mail-adresse ...
- Værdien i feltet „Zip code“ er kun relevant hvis man bor i USA, og i så fald skal der angives en lovlig zip-kode i feltet. For ikke at gøre sagerne alt for indviklede, vedtages det hermed, at en lovlig zip-kode består af præcis fem cifre, og at alle femcifrede tal er lovlige zip-koder.
- Det indtastede telefonnummer skal naturligvis være et nummer der overholder reglerne for hvordan telefonnumre ser ud i det valgte land.

Name:

E-mail:

Country:

Zip code:

Phone:

Request visit from NYC office

Figur 1: En illustrativ formular

- Endelig kan man kun bede om at få besøg fra New York-kontoret hvis man bor i USA og yderligere har et telefonnummer der hører hjemme i New York City. I dette eksempel begynder sådanne telefonnumre med „212“, „347“, „646“, „718“ eller „917“.

Ud over disse krav til værdier i felterne på formularen er der endnu en ting der kan forventes af den illustrative formular, og det drejer sig om hvordan den submittes: Enhver fornuftigt tænkende bruger vil forvente at det er muligt at annullere udfyldelsen af formularen ved at trykke på „Cancel“-knappen selvom værdierne i formularfelterne ikke nødvendigvis overholder alle stillede krav. Det bør derfor være muligt at angive hvorvidt det kræves at alle krav er overholdt når brugeren submitter formularen, idet annullering af udfyldning af en HTML-formular regnes for en speciel måde at submitte den på.

Det burde være klart at de opstillede krav er meget rimelige i forbindelse med en elektronisk webformularer, og der synes ikke at være nogen grund til at nogle af de beskrevne forslag vil have en fordel frem for de andre. Til trods for at eksemplet er ret simpelt, illustrerer det nogle relevante pointer, men det er dog på ingen måde dækkende for alt hvad man kan forestille sig af mere eller mindre interessante egenskaber ved en elektronisk formular.

2.2 XForms

XForms er W3C's bud på en efterfølger til HTML-formularer, hvor erfaringerne fra *the lessons learned in the years of HTML forms implementation experience* [16] høstes, og XForms virker – og er – derfor i mange henseender meget mere moderne og tidssvarende end HTML-formularer. Én af erfaringerne man har gjort sig, er at HTML-formularer simpelthen ikke er udtryksfulde nok til at honorere nutidens – og brugernes – krav, og derfor forbedrer XForms *greatly [...] the expressive capabilities of electronic forms* [16].

XForms er (naturligvis) et XML-sprog [11], og det bruges til at beskrive formularer ved at angive hvilke data formularen skal redigere, hvordan data skal præsenteres for brugeren og endelig hvordan data fra den udfyldte formular skal sendes tilbage til serveren. I modsætning HTML-formularer, er XForms delt op i disjunkte dele der hver for sig beskriver de nævnte aspekter af en formular og dens livscyklus.

En grundlæggende ide bag XForms er at der skal være adskillelse mellem hvad en formular bruges til, altså hvilke informationer den redigerer, og hvordan den præsenteres for brugeren. Beskrivelsen af hvad formularen skal bruges til beskrives i den såkaldte *XForms model*, og når denne er på plads, er XForms meget fleksibelt med hensyn til hvordan formularen skal præsenteres for brugeren idet der indbygget i XForms kun findes *the XForms User Interface, which is a device-independent, platform-neutral set of form controls suitable for general-purpose use* [16]. Som antydnet kan præsentationen kun beskrives meget abstrakt i selve XForms, og det er derfor muligt at bruge en hel række brugergrænseflader som implementation af den abstrakte præsentationsbeskrivelse. Det illustrative eksempel vil forøges formuleret med XHTML som omgivelse for XForms.

Det er på sin plads at nævne, at XForms er et stort og meget omfattende apparat der tager hånd om mange ting i forbindelse med elektroniske formularer. Nogle højdepunkter, hvis detaljerede gennemgang ligger uden for dette speciales område, er kort resumeret nedenfor. Den interesserede læser kan fordybe sig i detaljerne ved læsning af XForms-specifikationen [16].

Dynamiske formularer

I XForms er der mulighed for at lave dynamiske formularer hvor visse dele kun er synlige når et eller andet krav er opfyldt.

Dels kan man lave formularer hvor en meget stor formular kun præsenteres for brugeren lidt ad gangen, og han så kan „hoppe“ til den næste del når den aktuelle del er udfyldt. Ofte indeholder formularer en række adskilte dele der kan (eller skal) udfyldes i en mere eller mindre „naturlig“ rækkefølge, og for ikke at forvirre brugeren, kan disse dele så præsenteres enkeltvist i en passende rækkefølge.

Håndtagene til at få den skitserede dynamik i sving, er beskrevet i kapitel 9 og 10 i XForms-specifikationen [16], hvorfra også kapitel 8 om *the XForms User Interface* kan anbefales at læse.

Internationalisering

Et andet område XForms understøtter er såkaldt *internationalisering* af formularer, altså muligheden for at lave formularer hvori beskrivende tekster ikke er skrevet på et bestemt sprog, men hvor de kan afhænge af hvilket sprog udfyldereren af formularer læser og skriver. Det vil oplagt øge brugervenligheden af en formular hvis en dansker kan læse teksten

„Telefonnummer“ i nærheden af et felt til indtastning af et telefonnummer mens en franskmand ser „Numéro de téléphone“.

Internationalisering drejer sig også om hvordan datoer skrives og vises, og XForms understøtter også denne gren af internationalisering, fordi *the data presented to the user through a form control must directly correspond to the bound instance data, [but] the display representation is not required to exactly match the lexical value* [16]. Nærmere detaljer om hvordan internationalisering kan klares i XForms fremgår af specifikationenes kapitel 8.12.

2.2.1 Design

I det følgende vil designet af XForms blive beskrevet, hvilket vil sige hvordan XForms er bygget op. For at skabe lidt overblik er der i figur 2 opremset en række centrale begreber fra XForms-verdenen.

Et XForms-dokument består af én eller flere *formularer*. Til hver formular er der, implicit eller eksplicit, knyttet en *model* der beskriver de data den tilhørende formular skal manipulere.

Dokument	Den øverste begrebsmæssige enhed. Består af én eller flere <i>formularer</i>
Formular	... Til hver formular er knyttet én <i>model</i> .
Model	Beskrivelse af hvilke data (<i>instansdata</i>) en formular afspejler og hvilke constraints der er tilknyttet dataene. Modellen beskriver også hvad der skal submittes og hvordan, når den tid kommer.
Instansdata	En XML-struktur som en formular skal manipulere.
Dataknode	En knude i <i>instansdata</i> .
Model item	Dataknode med tilknyttet XForms-constraint (binding).
Skemaconstraint	Et constraint der stammer fra et XML Schema som instansdata skal overholde.
XForms-constraint	Et XForms-specifikt constraint der er angivet eksplicit via en binding i modellen.
Binding	Definition af et XForms-constraint eller en sammenkædning af dataknode og et kontrolelement.
XForms-processor	Et program der er en implementation af XForms-specifikationen. Det er processoren der tager sig af validering af data i en formular.

Figur 2: Nøglebegreber i XForms

Fremover vil alle XML-elementer der stammer fra XForms have navnerummet „xfrm“ [10], hvilket for lægfolk blot betyder at elementnavnet „xfrm:input“ svarer til XForms-elementet „input“.

XForms-modellen

Som nævnt beskrives en XForms-formulars anvendelse i XForms-modellen, og denne beskrivelse består af flere dele. Først og fremmest kan de *instansdata* (instance data) som formularen skal manipulere beskrives, og dette gøres ved at give en skabelon for XML-strukturen. Når brugeren senere udfylder formularen, vil hans input blive afspejlet som ændringer i de underliggende instansdata, dvs. som ændringer af værdier af dataknuder.

Der er også mulighed for at referere til eksterne data som instansdata, hvilket giver nogle interessante fordele. Hvis man bruger eksternt definerede instansdata i sin model, kan man nøjes med at redigere en lille del af en meget stor XML-struktur, og dette gør det videre muligt at lave flere formularer der hver manipulerer én del af strukturen, men som sammenlagt redigerer hele strukturen. En sådan modularisering giver oplagte fordele for både programmør og bruger, idet det er nemmere at overskue en lille del af strukturen frem for at skulle holde styr på hele dynen i én stor mundfuld.

Som det senere vil blive klart, er der en direkte forbindelse mellem instansdata og formularen som brugeren ser den, idet formularen til enhver tid afspejler til aktuelle tilstand i instansdata. Dette medfører, at det ved brug af eksterne instansdata er forholdsvis nemt at præsentere en tilbagevendende bruger for en præudfyldt formular der indeholder de værdier han indtastede sidst han havde fingrene i formularen. Hvis man har leget lidt med HTML-formularer og CGI-programmer, vil man sikkert kende til problemet med automatisk at udfylde en formular med „gamle“ værdier og vide at det ikke er helt så ligetil som man kunne ønske.

Der kan selvfølgelig opstå problemer hvis den eksterne struktur pludselig ændres fundamentalt, men det må formodes at der er en vis form for kommunikation mellem ham der holder styr på XML-strukturen og ham der programmerer XForms, og at de kan finde ud af at koordinere deres indsats hvis gennemgribende ændringer er nødvendige.

Hvis man ikke angiver instansdata i modellen, vil strukturen implicit blive defineret af bindinger af kontrolelementer [afsnit 2.2.1], men i så fald giver man afkald på hele ideen med XForms og ender med noget der svarer til HTML-formularer [16, afsnit 4.2].

I tillæg til en XML-skabelon for instansdata kan man i modellen angive eller referere til et XML Schema [24] som instansdata skal overholde. Som det måske er bekendt, kan man ud fra et skema stille krav til indholdet af elementer og attributter i instansdata, og hermed er der allerede bragt en form for validering på banen, idet XForms-processoren vil tjekke instansdata mod skemaet og rapportere eventuelle fejl. Et constraint for instansdata der stammer fra en skemaangivelse i modellen, kaldes i XForms-terminologi meget naturligt – og direkte oversat fra engelsk – et *skemaconstraint*.

XForms-constraints

Ud over constraints specificeret af et skema kan man i XForms angive en række *XForms-constraints*, der er XForms-specifikke krav til instansdata. I lighed med mulighederne i XML Schema kan man med et XForms-constraint stille krav til værdier af knuder i instansdata, men man har tillige mulighed for at angive mere XForms-specifikke krav. Disse krav kan blandt andet være at en knude skal og kun må have en (ikke-tom) værdi hvis et andet element har en bestemt værdi eller at en knude kun skal forekomme i instansdata når en eller anden betingelse er opfyldt [afsnit 2.2.2]. Disse to sidstnævnte krav ligger begge uden for XML Schemas udtrykskraft, men det viser måske snarere en svaghed i XML Schema end en styrke i XForms ...

En dataknode med tilknyttet constraint kaldes et *model item*, og for at tilknytte et constraint til et dataknode laves en *binding* ved brug af et bind-element, hvis struktur er beskrevet nedenfor.

```
<bind ref=ref [type=type] [required=required] [relevant=relevant]  
  [isValid=isValid] [readOnly=readOnly] [calculate=calculate]/>
```

ref

Et XPath-udtryk [3] der udpeger de dataknuder bindingen skal gælde for.

type

Navnet på en XML Schema-datatype som bundne dataknuders værdi skal være repræsentationer af [4, afsnit 2.3].

required

Et XPath-udtryk der løbende beregnes og hvis boolske værdi angiver om de bundne knuder *skal* have en værdi.

relevant

Et boolsk XPath-udtryk der angiver om de bundne knuder har relevans for resten af modellen; specielt om de skal medtages når instansdata submittes [afsnit 2.2.2].

isValid

Et XPath-udtryk hvis boolsk værdi afgør om værdierne i de bundne knuder er lovlige.

readOnly

Et boolsk XPath-udtryk der bestemmer om værdien i de bundne knuder kan ændres. Når der senere laves en binding mellem dataknuderne og et kontrol-element, vil kontrolelementet signalere til brugeren om en værdi kan ændres eller ej.

calculate

Et XPath-udtryk der bestemmer værdien af de bundne knuder.

Værdien af ref-attributten skal være et XPath-udtryk der ikke afhænger af den aktuelle tilstand i formularen, dvs. at mængden af knuder en binding gælder for hverken kan eller må ændre sig mens formularen udfyldes. De nærmere krav til udtrykket kan findes i afsnit 6.4.2 i XForms-specifikationen [16].

Attributten type angiver navnet på en erklæret XML Schema-datatype der er en type defineret enten af XML Schema [4] eller i skemaet for instansdata i selve modellen i XForms-dokumentet. Det afgørende er at notere sig at der skal angives et *fast navn*, og at typen for dataknuder derfor *ikke* kan afhænge dynamisk af formulartilstanden.

Hvis værdien af attributten required er sand, skal den bundne instansknude have en ikke-tom værdi, hvilket groft sagt svarer til en ikke-tom tekststreng. Den faktiske definition involverer detaljer fra flere specifikationer fra XML-verdenen [3, 24], som den nysgerrige læser – på eget ansvar – kan studere.

Når en XForms-formular submittes, foregår det ved at instansdata sendes tilbage til serveren, og i den forbindelse er relevant-attributten relevant. Hvis den boolske værdi af omtalte attribut er falsk, vil den bundne instansknude blive fjernet fra instansdata når formularen submittes, og det er således kun de dataknuder der er relevante, der sendes tilbage til serveren.

Som eksempel på brug af bind-elementet kan der tages forskud på glæderne og bringes følgende uddrag fra det illustrative eksempel:


```
<xfm:bind id="bind-zip-code" ref="/problem/personal/zip-code"
  type="zip-code"
  relevant="/problem/personal/country = 'usa'"/>
```

Her kræves at en zip-kode har en bestemt type (defineret andetsteds), og samtidig sørges for at zip-code-elementet kun bliver submittet hvis det er relevant, dvs. hvis brugeren bor i USA. Det vil i afsnit 2.2.3 blive klart hvorfor værdierne af ref- og relevant-attributterne ser ud som de gør.

Kontrollementer

Som nævnt findes der i XForms kun abstrakte grænsefladekomponenter og det er op til XForms-processoren på klienten at lave en passende præsentation når formularen skal vises.

På den illustrative formular bruges et HTML-select-element hvori kun én indgang kan vælges til at vælge hvilket land man er fra. I XForms vil et tilsvarende kontrollement blive defineret på følgende vis:

```
<xfm:selectOne ref="/problem/personal/country">
  <xfm:caption>Country</xfm:caption>
  <xfm:choices>
    <xfm:item>
      <xfm:caption>Danmark</xfm:caption>
      <xfm:value>dk</xfm:value>
    </xfm:item>
    <xfm:item>
      <xfm:caption>USA</xfm:caption>
      <xfm:value>usa</xfm:value>
    </xfm:item>
  </xfm:choices>
</xfm:selectOne>
```

Det turde fremgå at det resulterer i et kontrollement hvori man kan vælge netop ét af de angivne item-elementer, men hvordan de faktiske præsentation af elementet bliver bestemmes af XForms-processoren. I ovenstående liste er der kun to indgange, og XForms-processoren kan så vælge at præsentere disse indgange som to radioknapper i stedet for som en meget kort liste. Af og til vil folk kunne styre hvordan tingene ser ud, og derfor er der i XForms trods alt mulighed for at tvinge processoren til at vise et selectOne-element på en bestemt måde [16, afsnit 8.9].

Det lader til, at der i XForms er gjort meget ud af gøre formularer så brugervenlige som muligt, idet *Form controls enable accessibility by taking a uniform approach to such features as captions, help text, tabbing and keyboard shortcuts* [16]. Tilsvarende muligheder er efterhånden kommet med i HTML-specifikationen [22], men en webdesigner skal være godt hjemme i HTML for at få det fulde udbytte, og der opfordres ikke på samme måde som i XForms til at gøre en HTML-formular så brugervenlig som muligt. Ydermere synes ikke alle browserprogrammører at være gode til HTML, hvorfor ens gode intentioner ofte bliver gjort til skamme.

Bindinger

Når man har lavet sin XForms-model og den abstrakte grænseflade, skal de to dele sættes sammen således at grænsefladen kan afspejle værdierne i instansdata. For at snøre tingene

sammen laver man en *binding* mellem et kontrolement og en dataknode, og i denne binding defineres hvilken dataknodes værdi der skal redigeres i det aktuelle kontrolement.

En binding mellem et kontrolement og instansdata kan laves ved enten direkte at angive hvilke knuder der skal bindes til elementet, eller ved at referere til et XForms-constraint defineret ved brug af bind-elementet. Alle elementer i XForms-brugergrænsefladen har med bindinger for øje fire attributter, hvoraf kun én af ref og nodeset må forekomme:

ref

Et XPath-udtryk der udpeger en knude som kontrolementet skal knyttes til. Hvis udtrykket evaluerer til mere end én knude, gælder bindingen kun den første.

nodeset

Et XPath-udtryk der udpeger en mængde knuder som kontrolementet skal knyttes til.

model

En reference til den model bindingen skal gælde for.

bind

Laver en binding via et bind-element defineret i modellen.

Alt efter om det kontrolement der skal bindes kan have én eller flere værdier, er det ref- eller nodeset-attributten der skal bruges til at lave bindingen. En gruppe af afkrydsningsfelter kan bindes til en serie af elementer i instansdata, hvorimod et tekstfelt kun kan bindes til en enkelt dataknode. En binding mellem et kontrolement og dataknode kan laves med kontrolementets ref-attribut, der i så fald skal udpege de knuder der skal bindes. Som eksempel bringes følgende der laver en binding mellem en knude i instansdata og et kontrolement:

```
<xfm:input ref="/problem/personal/zip-code" style="width: 8cm">
  <xfm:caption>Zip code</xfm:caption>
  <xfm:alert>Invalid zip code</xfm:alert>
</xfm:input>
```

Bindingen der laves, sørger for at input-elementet (der svarer til et input-element med type text i XHTML) afspejler og redigerer værdien af et zip-code-element i problem-elementet i instansdata. Alternativt kan bindingen laves ved at lade input-elementet referere til et XForms-constraint for zip-code-elementet som vist ovenfor:

```
<xfm:input bind="bind-zip-code">
  <xfm:caption>Zip code</xfm:caption>
  <xfm:alert>Invalid zip code</xfm:alert>
</xfm:input>
```

og effekten vil være den samme.

2.2.2 Brugerinteraktion

Efter de indledende knæbøjninger i XForms, hvor en formular er blevet beskrevet med en passende model og der er lavet bindinger mellem kontrolementer og instansdata, er det blevet tid til at få brugeren bragt på banen og præsentere ham og formular for hinanden.

De begivenheder der skal indtræffe i forbindelse med en brugers udfyldelse af en formular er beskrevet i *the XForms processing model* [16, afsnit 4], og forløbet af denne er som følger:

Initialisering

Hver XForms-model i dokumentet initialiseres på følgende vis: instansdata for formularen konstrueres ud fra den skabelon der refereres til eller er angivet i modellen, og hvis der ikke er givet en skabelon for strukturen, defineres den på samme måde som i HTML, nemlig implicit ud fra bindinger af dataknuder i kontrolelementer [16, afsnit 4.2]. Hvis modellen indeholder en skemaspecifikation, kontrolleres at instansdata overholder skemaet, og hvis ikke det er tilfældet, stopper al videre behandling af *hele* XForms-dokumentet. Ellers er modellen initialiseret, og når alle modeller er succesfuldt initialiseret, er formularen klar til brug og interaktion.

Udfyldning

Når brugeren udfylder formularen, kan han groft sagt påvirke dens værdier på tre måder: han kan skrive tekst i tekstfelter, han kan (af)markere eksempelvis afkrydsningsfelter og han kan vælge indgange i lister.

Hvis brugeren skriver noget tekst i et tekstfelt og „forlader“ tekstfeltet eller han trykker på en knap eller en listeindgang, vil instansdata umiddelbart blive opdateret til at afspejle den nye værdi. Hvis ændringen i instansdata medfører at et skemaconstraint eller et XForms-constraints ikke er overholdt, markeres det kontrolelement der har forårsaget fejlen som værende *ugyldigt*, og en tilhørende fejlbesked vises. Det er ikke nærmere specificeret hvordan fejlbeskeden skal vises, men i én implementation (X-Smiles) dukker en dialogboks op og det kontrolelement der har forårsaget fejlen fremhæves.

I forbindelse med felter hvor brugeren skal skrive tekst, opstår en speciel situation mens han skriver teksten, da det midlertidige indhold i feltet kan bryde constraints defineret i modellen. I den illustrative formular skal det naturligvis tillades at brugeren indtaster sin e-mail-adresse tegn for tegn, og han skal ikke belemres med fejlbeskeder for hvert tegn han skriver, fordi „james@met“ ikke er en lovlig e-mail-adresse. I henhold til specifikationen må en XForms-processor opdatere instansdata med midlertidige værdier hvis de overholder alle relevante constraints og ellers må instansdata først opdateres når den nye værdi er endeligt fastlagt, dvs. når brugeren forlader feltet.

Aflevering

Når brugeren er tilfreds med sin udfyldelse af den forelagte formular og ønsker at submitte den, skal det afgøres om formularen også er tilfreds, hvilket vil sige at det skal afgøres om alle constraints er overholdt. Hvis ikke, må brugeren præsenteres for nogle informative fejlbeskeder, hvorefter han kan udbedre sine fejl og sørge for at alle bliver tilfredse. Efter som det i XForms er muligt kun at submitte en del af instansdata, beregnes indledningsvist den del af instansdata der faktisk skal submittes. Dernæst tjekkes ganske naturligt at alle constraints for netop denne del af instansdata er, overholdt og hvis dette ikke er tilfældet stopper, submit-processen. Hvis alle relevante constraints er overholdt, serialiseres de dataelementer der skal submittes som angivet i XForms-modellen og sendes til „rette vedkommende“, som forhåbentlig er en webserver med en XForms-processor.

Det vil her føre for vidt at komme ind på nærmere detaljer om forløbet af en brugers interaktion med en formular, så igen henvises til XForms-specifikationen [16].

2.2.3 Eksemplet

Det er nu på tide at se om og hvordan det illustrative eksempel kan formuleres i XForms, men inden man kaster sig ud i projektet, er der lige nogle overvejelser at gøre sig.

Strukturen af instansdata skal på plads, men som det ofte er tilfældet når man designer XML-strukturer, er det ikke entydigt hvordan strukturen skal se ud. Naturligvis skal alle de ønskede informationer kunne repræsenteres, men der er forskel på hvor meget information der skal være i instansdata. Det er kun hvis man bor i USA at en zip-kode giver mening, og tilsvarende kan man kun bede om et besøg når man bor et bestemt sted i USA. Da XForms understøtter muligheden for at udelade visse knuder når formularen submittes, er der for eksemplets skyld grund til eksperimenterer lidt med netop dette.

Som det vil blive klart, er eksemplet blevet formuleret ved at indlejre XForms i XHTML, hvilket ikke har den store relevans for selve eksemplet, men det er meget praktisk hvis man vil bygge en komplet formular op med forklarende tekster og overskrifter og lignende.

Nedenfor vil eksemplet blive gennemgået stump for stump, og det fuldstændige XForms-dokument og en tilhørende XML Schema-definition kan ses i appendiks A.2.

Modellen

Indledningsvist skal XForms-modellen på plads, og dette består først og fremmest i at definere strukturen af instansdata. For eksemplets og øvelsens skyld er der opskrevet et XML Schema for instansdata som definerer en række skemaconstraints. Dernæst angives XForms-constraints ved at lave passende bindinger af relevante knuder i instansdata.

Som antydnet ovenfor er det ikke entydigt hvordan XML-strukturen af instansdata skal se ud, men hvis det antages at den illustrative formular skal bruges i forbindelse med fejlrapportering, er følgende skabelon et godt bud:

```
<xfm:instance xmlns="">
  <problem>
    <personal>
      <name/>
      <e-mail/>
      <country>dk</country>
      <zip-code/>
      <phone/>
    </personal>
    <visit>>false</visit>
  </problem>
</xfm:instance>

<xfm:schema href="NYC.xsd"/>
```

Ud over selve skabelonen refereres der i modellen til et XML Schema for instansdata som blandt andet lægger visse begrænsninger på værdierne i de forskellige knuder i instansdata. Derudover angiver skemaet at elementerne zip-code og visit ikke nødvendigvis behøver at være til stede i instansdata, men det overlades til XForms at holde styr på præcist i hvilke situationer de må være fraværende.

Det skal være muligt for brugeren at submitte formularen på to måder, nemlig på normal vis ved at trykke på "OK"-knappen, og ved at trykke på "Cancel"-knappen, hvorved han meddeler at der ikke er angivet fornuftige oplysninger i formularen. Man kan i et submitInfo-element angive hvilken del af instansdata der skal submittes, men hvis man

ikke udtaler sig om hvilken del der skal submittes, bliver hele instansdata submittet. Som beskrevet i forrige afsnit vil kun constraints for den udvalgte delmængde af instansdata blive kontrolleret, og den usædvanlige situation at brugeren vil submitte en ikke-udfyldt formular kan måske klares med følgende:

```
<xfm:submitInfo id="cancel" method="post" ref=""  
    action="http://example.com/submit"/>
```

Den opmærksomme læser vil have bemærket den vage formulering „kan måske klares“ ovenfor, og hvis man prøver at køre eksemplet i en tilgængelig XForms-processor, vil man erfare at der i stedet for „måske“ bør stå „ikke“, idet det ikke synes at være muligt at annullere udfyldning af en formular ved at udvælge ingen knuder i instansdata og kun få tjekket deres constraints . . .

En e-mail-adresse er en e-mail-adresse uanset hvor i verden man kommer fra, så der skal ikke stilles ekstra krav til den end der allerede gøres i skemaet, hvor det kræves at den overholder et bestemt regulært udtryk.

Som bekendt er der ingen grund til at medtage zip-code-elementet med mindre man bor i USA, og dette krav kan, som tidligere vist, formuleres ved følgende binding:

```
<xfm:bind id="bind-zip-code" ref="/problem/personal/zip-code"  
    type="zip-code"  
    relevant="/problem/personal/country = 'usa'"/>
```

Dette angiver at elementet kun skal medtages i instansdata når country-elementet har værdien usa, hvilket er i trit med det ønskede. For kontrolementer bundet til ikke-relevante dataknuder hedder det at *The form control (and any children) should be hidden or unavailable to the user* [16], og brugeren bliver således gjort opmærksom på om han forventes at indtaste en zip-kode eller ej.

Enten har man en zip-kode eller også har man ikke! Anderledes forholder det sig med et telefonnummer. Enten har man ikke et, eller også har man et hvis udseende afhænger af hvorhenne man bor. XForms kommer til kort når man vil lade lovlige værdier af telefonnumre afhænge af hvilket land man kommer fra, så i eksemplet defineres lovlige værdier for telefonnumre til at være foreningen af lovlige værdier for USA og lovlige værdier for Danmark, idet der så i det mindste er lidt styr på brugerens indtastninger. Det er skemaet for instansdata der definerer lovlige telefonnumre og der er ikke yderligere XForms-specifikke krav der kan stilles – med mindre man vil til at skrive indviklet scriptkode som eksemplificeret nedenfor . . .

Ud over at det kræves at man bor i USA for at visit-elementet relevant, er der yderligere krav der skal være opfyldt, idet ens telefonnummer skal høre hjemme i New York City. En binding der udtrykker dette, ser ud som følger:

```
<xfm:bind id="bind-visit" ref="/problem/visit"  
    relevant="(/problem/personal/country = 'usa')  
    and ((substring(/problem/personal/phone, 1, 3) = '212')  
    or (substring(/problem/personal/phone, 1, 3) = '347')  
    or (substring(/problem/personal/phone, 1, 3) = '646')  
    or (substring(/problem/personal/phone, 1, 3) = '718')  
    or (substring(/problem/personal/phone, 1, 3) = '917'))"/>
```

Udtrykket i relevant-attributværdien minder i foruroligende grad om gammeldags scriptkode og her øjnes et problem ved XForms, idet der ikke er noget nyt og moderne eller smart ved håndskrevet, traditionel scriptkode . . .

Kontrollementer

Efter at XForms-modellen er på plads – så godt som det nu lader sig gøre – er det tid til at få defineret kontrollementerne der skal afspejle værdierne i instansdata, og dette foregår meget nemt ved brug af de abstrakte kontrollementer XForms stiller til rådighed. Da der ikke er nogen problemer eller banebrydende afsløringer i de XML-besværgelser der skal til for at få lavet de relevante kontrollementer, vil der blot blive henvist til appendiks A.2, hvor kildeteksten på det nærmeste forklarer sig selv: se appendiks A.2, hvor kildeteksten på det nærmeste forklarer sig selv!

Den opmærksomme læser vil under studering af eksemplet bide mærke i at alle bindinger mellem instansdata og kontrollementer er lavet med `ref`-attributten i kontrollementerne. Som tidligere nævnt kan disse bindinger i følge XForms-specifikationen også laves ved at lade et kontrollement pege på et `bind`-element, men for at få eksemplet til at køre bare nogenlunde i praksis, har den første løsning været nødvendig.

Eksemplet i praksis

I figur 3 ses hvordan eksemplet vises i X-Smiles, der for tiden er den eneste tilgængelige implementation af XForms.



The image shows a screenshot of a dialog box from the X-Smiles application. The dialog is titled with a light gray background and contains the following elements from top to bottom: a text input field labeled 'Name', another text input field labeled 'E-mail', a dropdown menu labeled 'Country' with 'Danmark' selected, and a text input field labeled 'Phone'. At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

Figur 3: Et eksempel på hvordan en XForms-version af det illustrative eksempel tager sig ud i X-Smiles.

Når en bruger begiver sig ud at udfylde formularen, kan scenariet der er afbildet i figur 4, udspille sig. Først indtaster brugeren sit navn, hvorefter han går i gang med at indtaste sin e-mail-adresse (*a*). Når e-mail-adressen er på plads, anfører brugeren at han bor i USA og som et lyn fra en klar himmel dukker zip-kode-feltet op. Underligt nok vælger brugeren så at indtaste de tre første cifre i sit telefonnummer og derefter forlade telefonnummerfeltet, hvilket medfører at feltet farves rødt og et afkrydsningsfelt dukker op (*b*). Den røde farve antyder at det sværtede felt indeholder en ulovlig værdi. Brugeren besinder sig og udfylder hele formularen på lovlig vis (*c*), men kort efter vælger han at flytte til Danmark og det medfører at hans zip-kode og afkrydsningsfeltet ikke længere er at se (*d*).

Som nævnt er det i XForms ikke muligt at formulere kravet om at et telefonnummer skal svare til det valgte land. Derfor adviseres brugeren desværre ikke om at der er inkonsistens

(a)

(b)

(c)

(d)

Figur 4: Det illustrative eksempel som det udvikler sig mens brugeren fylder ud.

i hans udfyldning af formularen som den ser ud i det sydøstlige hjørne af figur 4. Yderligere vil brugeren blive slemt skuffet hvis han trykker på „Cancel“-knappen før alle constraints for felterne i formularen er opfyldt, idet han vil få at vide at formularen indeholder fejl og derfor ikke kan submittes.

2.2.4 Indtryk

Det er oplagt at XForms er et skridt i den rigtige retning i forbindelse med at bringe web-baserede formularer på højde med kravene der rimeligvis kan stilles i dag. Især muligheden for at lave dynamiske formularer, hvor felter dukker op efter behov mangler i HTML-formularer. Det er også oplagt at det er nødvendigt med mere struktur på de værdier der returneres fra en formular, og i den disciplin står XForms meget stærkt fordi det er XML-strukturer der returneres, og der er således (formentlig) ingen grænser for hvilke informationer man kan få fra en elektronisk formular.

Når talen falder på emnet for dette speciale nemlig validering af værdier i formularfelter, lader XForms dog noget tilbage at ønske. For det første virker det ikke videre fleksibelt at typen af værdier i et felt ikke kan ændres dynamisk, hvilket det illustrative eksempel med telefonnummeret viser at der til tider er brug for. For det andet er det bøvlet at man kan risikere at skulle angive (mindst) tre attributter for at specificere at et felt *skal* have en værdi (required), at værdien skal være af en bestemt type (type) og at værdien i øvrigt skal sendes med tilbage til serveren (relevant). Det er svært at se hvorfor man skulle være interesseret i at kræve værdier af en bestemt type og samtidig være ligeglad med værdierne. Det er mere intuitivt hvis man kan sige: „hvis en eller andet betingelse er opfyldt, så skal der stå en zip-kode. Ellers er det ligemeget hvad der står.“

I forlængelse af dette synes det at være et designmæssigt problem at XForms-constraints skal indkodes i attributter i et bind-element, idet værdier af attributter er tekststreng og det derfor er umuligt at udtrykke struktur uden at benytte en form for scriptsprog i værdierne. Hvis W3C i stedet havde valgt at lade underelementer af bind-elementet beskrive relevante egenskaber for et constraint, ville det være nemmere at skrive og overskue constraints der er mere avancerede end blot at kræve for eksempel en bestemt type for et felt.

Som beskrevet i forbindelse med det illustrative eksempel er det i XForms ikke umiddelbart muligt at lade brugeren annullere udfyldelsen af en formular, hvilket må betegnes som en oplagt mangel i XForms. Hvis en bruger ønsker at afbryde sin interaktion med en web-service, bør der være en måde eksplicit at tilkendegive dette på således at serveren er klar over at den ikke skal forvente yderligere kommunikation med brugeren.

Et andet problem ved XForms – og et meget alvorligt et af slagsen – er at det er meget svært at prøve hvordan skidt rent faktisk virker – og om det virker. For det første er specifikationen af XForms [16] kun et såkaldt *Working Draft*, og til trods for at det er den sidste „kladde“, er der stadig løse ender hist og her, der mangler at blive afsluttet. Et andet problem ved XForms – og mange andre W3C-specifikationer – er at en arbejdsgruppe sidder og finder på en hel masse ting uden at give en faktisk implementation der kan underbygge at ideerne ikke er helt hen i vejret. Det er helt overladt til „andre“ at implementere XForms og i skrivende stund findes kun én implementation, og denne går under navnet *X-Smiles* [<http://www.xsmiles.org/>]. For at gøre ondt værre implementerer X-Smiles kun ca. 75% af den nuværende, ikke-færdige XForms-specifikation, og man kan så selv gange 75% med „ikke færdig“ for at få et indtryk af hvor nemt – eller svært – det er at prøve XForms i „virkeligheden“.

2.3 Extensible Forms Description Language

I årene 1993–1998 udviklede firmaet Unisoft Wares Incorporated (UWI) *Universal Forms Description Language* (UFDL) der, som navnet antyder, er et sprog til beskrivelse af (elektroniske) formularer og de arbejdsgange der knytter sig til dem. Projektet tog udgangspunkt i det faktum at et stigende antal virksomheder benyttede internettet til at sende formularer frem og tilbage mellem sig selv og hinanden og kunderne, og at HTML-formularer ikke var udtrykksfulde nok til mange af de anvendelser der viste sig. I 1998 dukkede XML [11] op og pludselig skulle alverden skrives i XML. Kort efter blev *Extensible Forms Description Language* (XFDL) [6] introduceret som en XML-version af UFDL, og denne version blev introduceret for alle dem der gad høre efter. Den 14. februar 2000 skiftede UWI så navn til PureEdge for at vise at de sandelig er med helt fremme når det drejer sig om elektronisk handel og den slags. Den interesserede læser kan oplyse sig om firmaets gøren og laden på deres hjemmeside [<http://www.PureEdge.com>], hvorfra det også er muligt at hente en implementation af XFDL.

Udover at offentliggøre sine ideer for interesserede købere, valgte PureEdge også at indsende XFDL-forslaget til W3C med standardisering for øje, og W3C har siden udtrykt stor begejstring for tiltaget. Siden da er arbejdet med XFDL fortsat støt og roligt, og beskrivelsen af version 4.5 er nys udkommet med farveforside, men det er stadig PureEdge der beskriver, så det er måske så som så med standardiseringen . . .

2.3.1 Design

Under arbejdet med den nuværende version af XFDL har PureEdge stillet nogle krav som projektet skal leve op til, og nogle centrale aspekter fremgår af nedenstående liste.

- Formularer skal være selvstændige enheder der er uafhængige af ydre faktorer så man ved hvad der er en del af formularen og hvad der ikke er.
- Man skal kunne specificere udformning og præsentation af en formular så man kan præsentere store og omfattende formularer uden problemer.
- I så stort omfang det lader sig gøre, skal kontrol og formatering af data ske hos klienten.
- Der skal kunne stilles krav til sikkerhed med digitale signaturer på formularer eller dele af dem.
- Det skal være en offentligt tilgængelig og åben standard.

Begrebsverden

Det lader til at PureEdge grundigt har studeret en „rigtig“ papirformulars livsforløb i forbindelse med designet af XFDL og så har forsøgt at efterligne det i forbløffende høj grad.

Hvis man stopper op et øjeblik og overvejer hvad der egentlig foregår i en formulars livsforløb, vil følgende billede tegne sig: Der er tre aktører, nemlig én der ved noget („klienten“), én der gerne vil vide noget („paven“) og endelig er der en formular der består af et eller flere stykker papir – fraregnet eventuelt karbonpapir. Som nævnt indledningsvist udspilles følgende scenarie så:

1. Paven bag skranken udleverer formularen til klienten.
2. Klienten udfylder formularen.
3. Den udfyldte formular leveres tilbage til paven.

Herefter er formularen i pavens varetægt, og det ligger udenfor dette speciales fagområde at beskæftige sig med dens videre skæbne ...

Det interessante i det beskrevne scenarie er at klienten *udfylder* formularen. Det han faktisk gør, er at tilføje noget information til formularen, og han ændrer således ikke i selve formularen eller dens udformning. Herved tilegner paven sig ikke blot nogle oplysninger fra klienten, men det fremgår også klart i hvilken sammenhæng oplysningerne er givet.

Dette virkelighedsnære menageri imiteres ganske præcist i XFDL hvor klienten tilsendes en XFDL-formular der i stor detalje beskriver alle aspekter af formularen både hvad angår form og indhold. Når formularen er blevet udfyldt, altså opdateret, sendes den i sin helhed tilbage til serveren, der vil vide præcis hvilken formular der er blevet udfyldt. Dette er noget anderledes end situationen i HTML og XForms, hvor serveren kun modtager værdierne der stammer fra udfyldelsen af formularen – eller hvor værdierne nu stammer fra.

Ved hele tiden at bevare en formular og dens informationer som en helhed, opnås mange fordele, blandt andet kan vedhæftninger indlejres i selve formularen, og en lang række sikkerhedsmæssige aspekter (digitale signaturer og lignende) kan håndteres på fornuftig vis, fordi man aldrig mister sammenhængen en formular indgår i. Specifikationen af XFDL [6] kan konsulteres for nærmere detaljer.

Figur 5 beskriver kort nogle centrale begreber i XFDL.

<p>Formular Den øverste begrebsmæssige enhed. Består af én eller flere <i>sider</i></p> <p>Side En samling af <i>items</i> der beskriver en del af en formular. Hvis en formular indeholder flere sider, er det muligt at skifte mellem disse uden at kontakte serveren der har leveret formularen.</p> <p>Item Et selvstændigt element på en side. Items inkluderer kontrolelementer, men der findes også items der bruges til at inkludere binære data eller til at lave digitale signaturer.</p> <p>Egenskab Alle elementer i XFDL har en række egenskaber kaldet „options“, der kan sættes. Egenskaberne kan blandt andet angive hvordan et element skal vises for brugeren, men præcist hvilke egenskaber der kan sættes, afhænger naturligvis af elementet.</p>

Figur 5: Nøglebegreber i XFDL

Struktur

Et XFDL-dokument består af præcis én formularerklæring og en eller flere sideerklæringer. I toppen af både formular- og sideerklæringer kan man angive en række egenskaber der er instruktioner om hvordan den omsluttende enhed (formular eller side) eksempelvis skal præsenteres. Det vil her komme for vidt at beskrive de tilgængelige options i detaljer og den flittige læser opfordres til at læse videre i XFDL-specifikationen [6], hvor især kapitel 7 giver nyttige oplysninger om den lidt specielle måde egenskaber kan angives på.

Formularer opbygges i XFDL med en lang række kontrolelementer og hvert af disse har en sværm af egenskaber der kan sættes på passende vis. Der er i XFDL lagt op til at de forskellige kontrolelementer på en formular skal se ud som brugeren „er vant til“, hvilket betyder at de ikke må adskille sig fra tilsvarende kontrolelementer i andre programmer som brugeren benytter. Til gengæld er det nøje angivet hvordan labels skal placeres i forhold til det kontrolelement de er tilknyttet, og placeringen er „ovenfor og venstrestillet“. Man kan mene hvad man vil om dette valg, men det er rart at vide at „sådan er det bare“ og så kan man forholde sig til dét. Når man ikke skal bekymre sig om eller angive hvor en label dukker op har man mere af sin mentale kapacitet til rådighed for selve det logiske design, og det kan gøre en formular betragteligt meget nemmere at gå til. Desværre er folk sjældent tilfredse med at „nogen“ har bestemt hvordan noget skal se ud, og så bliver der hevet i alverdens håndtag for at kunne styre tingene.

HTML er et skræmmende eksempel på denne tendens, idet det HTML – meget idealistisk – begyndte som et næsten rent *markeringssprog*, hvor man angiver den logiske struktur og så lader en browser bestemme hvordan strukturen skal vises i den fysiske verden. Det er folk bare ikke tilfredse med, og derfor er HTML blevet sølet ind i alverdens attributter og elementer til at styre det mere eller mindre præcise layout af et dokument. Det har videre ført til at det er muligt at skrive HTML-dokumenter der ikke kan vises i alle browsere! Heldigvis er der nu kræfter i spil for at få ryddet op i roderiet og få fjernet formatering fra selve dokumentet og få den slags samlet i style sheets og lignende ...

Validering

I XFDL er det muligt at tilknytte et *format* til visse kontrolelementer. Formatet består af en datatype som kontrolelementets værdi skal have og eventuelt yderligere krav til værdien. For tiden er der defineret 11 XFDL-specifikke datatyper man kan knytte til et kontrolelementet, men *XFDL's data typing controls can and should be replaced by a W3C-standardized set of data type specifiers* [6]. De typer XFDL stiller til rådighed, minder om en begrænset udgave af XML Schema-typerne, så disse W3C-typer er formentlig gode kandidater til fremtidige XFDL-typer.

Det virker meget stift at der er et begrænset antal typer man kan tilknytte et element – især når regulære udtryk ikke er én af dem – og hvis XFDL vil frem i verden, må der nok boller på suppen hvilket der jo også lægges op til.

Udover typen for værdien af et kontrolelement kan man angive en række tjekflag (*check flags*) som lægger yderligere begrænsninger på værdien. Disse begrænsninger kan blandt andet være at et tal skal ligge i et bestemt interval eller at værdien skal være med i en opremset liste af værdier. Igen virker udtrykskraften meget lidt fleksibel, og den bærer præg af at være designet af (og for) amerikanere, idet det synes lidt indspist med typen „dollar“ fremfor en mere generel „currency“-type. Den mest lovende valideringsform i XFDL synes at være muligheden for at opremse nogle mønstre (*templates*) som en værdi skal passe til. Ved første øjekast leder dette tanken hen på regulære udtryk, men ved nærmere eftersyn viser det sig at mønstrene blot er tekststrengene, hvori enkelt karakterer er tillagt en speciel betydning. Som eksempel kan nævnes at '#' er pladsholder „for netop ét ciffer“, hvilket er tilstrækkeligt for mange formål, men som hurtigt viser sig utilstrækkeligt når man vil have styr over *hvilke* cifre der må forekomme.

Som det tredje krydderi på værdien af et kontrolelement, kan man angive formateringsflag (*format flags*) som er nogle regler for hvordan værdien skal formateres når den præsenteres for brugeren. Et element der har tilknyttet formateringsflag, vil få sin værdi formateret i henhold til disse før værdien tjekkes op imod eventuelle tjekflag.

I figur 6 er vist et uddrag fra et XFDL-dokument, hvor et kontrolelement „amount“, får knyttet type- og formateringsinformationer til sin værdi. Kontrolelementet er et tekstfelt, hvis værdi er „23000“ og typen af værdien skal være „dollar“, hvilket i XFDL-terminologi er et decimaltal med to cifre efter kommaet – som i øvrigt er et punktum da XFDL snakker amerikansk. Yderligere skal værdien formateres således at tusinder adskilles af komma („comma_delimit“), og der skal foranstilles et dollartegn („add_ds“). Som resultat af disse formateringsflag vil feltets værdi blive præsenteret for brugeren som „\$23,000.00“, og det er også denne formaterede værdi der vil blive undersøgt af eventuelle tjekflag. Eksemplet illustrerer i betænkelig grad den meget lidt fleksible amerikanisering af XFDL.

```
<field sid="amount">
  <label>Amount</label>
  <value>23000</value>
  <format content="array">
    <type>dollar</type>
    <format>add_ds</format>
    <format>comma_delimit</format>
  </format>
</field>
```

Figur 6: Eksempel på et format i XFDL

2.3.2 Brugerinteraktion

I modsætning til XForms, hvor instansdata skal initialiseres før en bruger kan begynde på at udfylde en formular, kræver XFDL ikke de store armbevægelser før en formular er klar til at blive udfyldt. Dette skyldes at XFDL opfatter formularen og dens værdier som en sammenhængende helhed, men der skal trods alt alligevel laves lidt inden en formular er klar til brug.

Initialisering

En lang række egenskaber for elementer kan i XFDL beregnes dynamisk. Specielt kan værdier i kontrolelementer afhænge af andre værdier i formularen, og sådanne værdier skal beregnes inden brugeren giver sig i kast med at udfylde formularen. Når brugeren udfylder formularen skal værdierne naturligvis opdateres på passende vis for at afspejle den nye tilstand i formularen.

For at få styr på hvad der afhænger af hvad i en XFDL-formular, konstrueres en orienteret graf, hvori der er en knude for hver dynamisk egenskab og for hvert element der indgår i en beregning. Kanterne trækkes naturligt fra en beregningsknude til alle de elementer beregningen afhænger af. Herefter laves en topologisk sortering af den konstruerede graf, og det fremgår herefter i hvilken rækkefølge beregningerne skal foretages. Som konsekvens af denne tilgang, er cykliske afhængigheder ikke tilladt i XFDL – bortset fra at en beregning må afhænge af sin egen (tidligere) værdi i specielle tilfælde. Detaljer om selve beregningsprocessen kan læses i afsnit 3.9 i XFDL-specifikationen [6].

Udfyldning

Det fremgår ikke klart af XFDL-specifikationen hvad der sker når brugeren har skrevet en ulovlig værdi i et tekst-felt, alt der nævnes er at *An error message appears if the data is*

not entered correctly [6]. Der er naturligvis mulighed for selv at definere indholdet af den fejlbesked der gives, men det synes ikke at være muligt at styre hvordan og hvornår den dukker op.

Man kan hente en demoversion af en XFDL-processor (desværre kun til Microsoft Windows) og i denne dukker en dialogboks op når man har indtastet en ulovlig værdi, og man kan ikke „komme videre“ før de stillede krav til værdien er opfyldt. Denne meddelelsesform kan synes en smule aggressiv, og en mere stilfærdig advarsel vil være at foretrække mens brugeren er ved at udfylde formularen. Når formularen skal afleveres, er det naturligvis på sin plads at lade alle fløjter pibe og tydeligt gøre opmærksom på at dette eller hint krav ikke er opfyldt.

Når værdien af et element er blevet ændret, genevalueres alle afhængige værdier ud fra en del-graf af den topologisk sorterede graf beskrevet i forrige afsnit således at formularen afspejler den nye værdi.

En fiks detalje i XFDL er at hvis værdien af en indgang i en liste forbyrder sig mod et format knyttet til listen, vil indgangen enten blive fjernet eller være markeret som værende „ulovlig“. Dette er en oplagt hjælp til brugeren i bestræbelserne på at gøre udfyldningen af en formular så nem og ligetil som muligt.

Aflevering

Når brugeren er færdig med at udfylde formularen, kan den uden videre sendes til en XFDL-processor på serveren for videre behandling.

2.3.3 Eksemplet

Det er nu tid til at smide XFDL efter det illustrative eksempel og se hvad det fører med sig af både gode og dårlige oplevelser. Som allerede antydet er det ikke inden for validering at XFDL har sine stærke sider.

For at komme forvirringen i forkøbet bør det nævnes at XFDL kræver at alle kontrolelementer har en sid-attribut, at den hedder „sid“, og at det følgelig ikke er en trykfejl at der står „sid“. Nærmere detaljer om dette kan læses i XFDL-specifikationen [6].

Eftersom en XFDL-formular og dens data er knyttet uløseligt sammen, er der ikke bestemte indgangsbøtner der skal afsiges for at beskrive hvad formularen skal bruges til, idet beskrivelsen af kontrolelementer og deres formater definerer det nødvendige.

Kontrollementer til indtastning af tekstuelle værdier laves i XFDL ved brug af et field-element, og et felt til indtastning af ens navn laves på følgende vis:

```
<field sid="name">
  <label>Name</label>
</field>
```

E-mail-feltet skal laves på tilsvarende vis, men yderligere skal det gerne gælde at den indtastede værdi er en lovlig e-mail-adresse. Desværre er det ikke oplagt hvordan man skal kræve dette, idet de tilrådeværende typer ikke indeholder en „e-mail“-type og mønstre for værdier er heller ikke udtryksfulde nok. Derfor kræves der blot at der skrives en ikke-tom streng i feltet.

```

<field sid="e_mail">
  <label>E-mail</label>
  <format content="array">
    <ae>string</ae>
    <ae>mandatory</ae>
    <message>Invalid e-mail address</message>
  </format>
</field>

```

Det fremgår rimelig klart af appendiks A.1 hvordan man med popup- og cell-elementer opbygger en liste hvori der kan vælges netop én indgang. For ikke at kede læseren bliver der derfor straks blive sprunget videre til et felt til indtastning af en zip-kode, og et sådant kan realiseres på følgende måde:

```

<field sid="zip_code">
  <label>Zip code</label>
  <format content="array">
    <ae>string</ae>
    <ae content="compute">
      <compute>
        NYC.country.value == "country_usa" ? "mandatory" : "optional"
      </compute>
    </ae>
    <template content="array">
      <ae>#####</ae>
    </template>
    <message>Invalid zip code</message>
  </format>
  <visible content="compute">
    <compute>NYC.country.value == "country_usa" ? "on" : "off"</compute>
  </visible>
</field>

```

Besværgelserne i format-elementet gør at det er krævet at skrive en zip-kode hvis man bor i USA, og at en eventuel zip-kode skal bestå af fem cifre. Ydermere bruges visible-egenskaben til at skjule feltet hvis der ikke skal indtastes en zip-kode.

I følge XFDL-specifikationen [6, afsnit 6.13] burde den kontrollerende betingelse

$$\text{NYC.country.value} == \text{"country_usa"}$$

i compute-elementerne i ovenstående uddrag i stedet være

$$\text{NYC.country.value} \rightarrow \text{value} == \text{"usa"}$$

Desværre virker dette ikke i praksis, og derfor er den første version brugt ovenfor og vil blive brugt igen senere.

Der skal være mulighed for at angive et dansk eller et amerikansk telefonnummer alt efter hvor i verden man kommer fra, og desværre er løsningen af dette i XFDL at altid tillade enten et dansk eller et amerikansk nummer.

```

<field sid="phone">
  <label>Phone</label>
  <format content="array">
    <ae>string</ae>
    <template content="array">
      <ae>#####</ae>
      <ae>###-###-####</ae>
    </template>
    <message>Invalid phone number</message>
  </format>
</field>

```

Det bør bemærkes at hvis det skal tillades brugeren at adskille sit amerikanske telefonnummer med mellemrum i stedet for '-', så vil det kræve at der angives fire mønstre i ovenstående format. Hvis han yderligere skal have love til at udelade separatorer skal alle ni kombinationer af separatorer – og mangel på samme – opremses ...

Endelig skal der styr på et afkrydsningsfelt som helst kun skal kunne afkrydses når visse betingelser er opfyldt. Sarte sjæle advares om at der i det følgende vil forekomme et skummelt hack for at sørge for netop dette.

```

<check sid="visit">
  <label>Request visit from NYC office</label>
  <visible content="compute">
    <compute>
      (NYC.country.value == "country_usa"
      && ((strmatch("212*", NYC.phone.value) == "1")
      || (strmatch("347*", NYC.phone.value) == "1")
      || (strmatch("646*", NYC.phone.value) == "1")
      || (strmatch("718*", NYC.phone.value) == "1")
      || (strmatch("917*", NYC.phone.value) == "1"))) ? value : "off"
    </compute>
  </visible>
</check>

```

Det der foregår er at værdien af afkrydsningsfeltet (som kan være „on“ eller „off“) sættes til „off“ hvis betingelserne ikke er opfyldt og ellers sættes værdien til værdien selv. Normalt må værdier i XFDL ikke afhænge cyklisk af hinanden, men under visse omstændigheder er det tilladt at værdien af et felt afhænger af selvsamme felts værdi [6, afsnit 3.9.2], og det er denne egenskab ved XFDL der på snedig vis udnyttes ovenfor.

Som kronen på værket skal der laves et par knapper, og en submitknop er hurtigt lavet:

```

<button sid="ok">
  <value>OK</value>
  <type>submit</type>
  <url content="array">
    <ae>http://example.com/submit</ae>
  </url>
</button>

```

Når knappen aktiveres, vil formularen blive sendt til den XFDL-processor der er angivet i url-elementet.

Angående annullerknappen viser XFDL sig fra sin bedste side, idet det under designet er blevet indset at brugeren af og til vil have brug for at kunne fortryde udfyldning af en formular, hvorfor dette naturligvis er understøttet direkte i XFDL:

```

<button sid="cancel">
  <value>Cancel</value>
  <type>cancel</type>
  <itemlocation content="array">
    <ae content="array">
      <ae>after</ae>
      <ae>ok</ae>
    </ae>
  </itemlocation>
</button>

```

Den komplette XFDL-beskrivelse af det illustrative eksempel kan findes i appendiks A.1 og i næste afsnit vises en serie fine farveillustrationer der viser hvordan XFDL håndterer eksemplet i praksis.

Eksemplet i praksis

I et forsøg på at underbygge at XFDL har sin berettigelse, har PureEdge lavet en implementation af en XFDL-processor, som man med en del besvær kan hente fra deres hjemmeside [afsnit 4.1].

I det følgende vises en håndfuld eksempler på hvordan det illustrative eksempel ser ud når man kører det i XFDL. Først viser figur 7 hvordan formularen ser ud inden brugeren giver sig i kast med at udfylde den.

The image shows a graphical user interface for a form. It has a grey background. At the top is a text input field labeled 'Name'. Below it is another text input field labeled 'E-mail'. Under that is a dropdown menu labeled 'Country' with a small downward arrow icon. Below the dropdown is another text input field labeled 'Phone'. At the bottom of the form are two buttons: 'OK' and 'Cancel'.

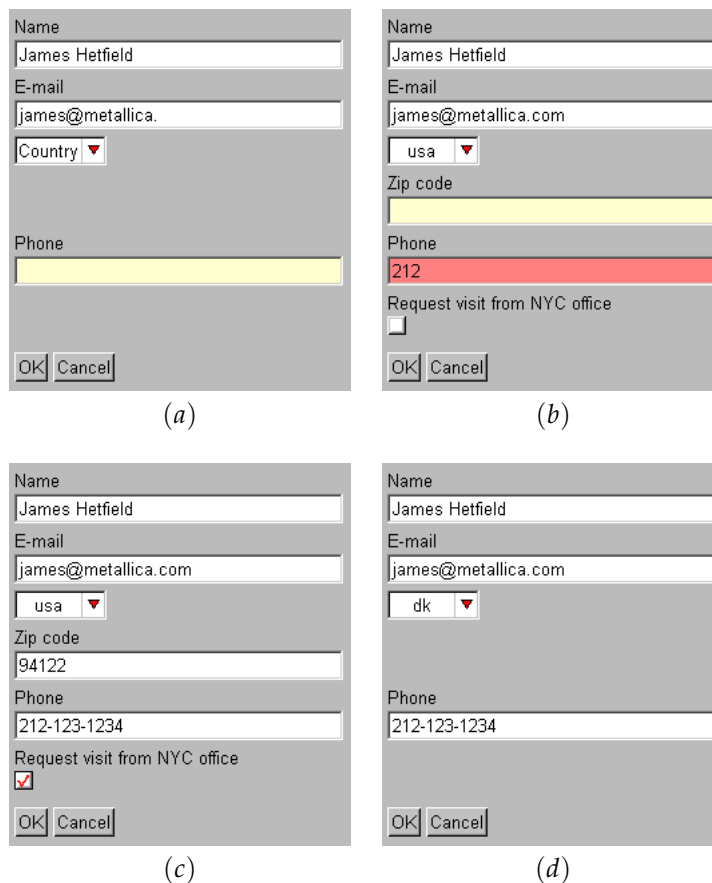
Figur 7: Det illustrative eksempel før udfyldning.

Det bemærkes at tekstfelter der skal udfyldes, har en farve der i folkemunde har et ikke specielt tiltalende navn, men som en billedkunstlærer nok vil kalde „lysegul“ eller måske endda „sommergul“ ...

Hvis det antages at en bruger der udfylder formularen gebærder sig på samme måde som beskrevet i afsnit 2.2.3, vil formularens udseende tilpasse sig som illustreret af tegneserien i figur 8.

I det store hele forløber formularudfyldningen på samme måde som beskrevet i afsnit 2.2.3, så der vil ikke komme lange udredninger om hvad brugeren foretager sig.

Én ting er dog værd at spekulere lidt nærmere over, og det er listen hvori man skal vælge sit land. Når denne vises første gang, er der ikke valgt nogen værdi, hvilket signaleres ved at det er selve listens label der vises, og det er meget rimeligt. Hvis man aktiverer listen for at vælge en værdi, dukker alle indgangene i listen op repræsenteret ved deres respektive



Figur 8: En billedsekvens der viser en brugers interaktion med en XFDL-formular. Brugeren er ikke afbildet.

label, men når man vælger en indgang i listen, er det den faktiske værdi af indgangen der vises i kontrolelementet. Som det fremgår af figur 8 vises således enten „usa“ eller „dk“ når der er valgt en værdi. Det ville være mere brugervenligt hvis værdien i listen blev repræsenteret ved den label der er tilknyttet den valgte indgang så brugeren i stedet for „dk“ ser „Denmark“.

2.3.4 Indtryk

Som bidragyder i forbindelse med modernisering af webbaserede formularer er det ikke indenfor validering af formularværdier at XFDL viser sig fra sin stærkeste side, men som det fremgår, er der lagt op til ændringer på dette område. Til gengæld må det siges at ideen med at opfatte en formular og dens indhold som en sammensat enhed, hvor formularen eller dens værdier ikke kan eksistere adskilt, er meget interessant. Set i bagklogskabens ulideligt klare skær fristes man til at sige: „Ja, selvfølgelig skal det fungere på den måde!“, men traditionelle HTML-formularer har måske fastlåst folk i vanetænkning omkring elektroniske formularer. Folkene bag XForms har konstateret at værdier fra en formular skal være mere struktureret end blot at være en række par af navne og værdier (som i HTML), men det er ikke en revolutionerende ny måde at opfatte formularer på. Som tidligere nævnt synes designerne bag XFDL at have magtet – og turdet – at starte helt fra bunden og genoverveje hele formularbegrebet, hvilket kan være en meget sund øvelse.

Som demonstreret i forbindelse med det illustrative eksempel er det i XFDL ofte nødvendigt at skrive ikke-trivielle kodestumper for at opnå den ønskede funktionalitet, hvorfor det er nødvendigt med en vis portion erfaring med programmering før man kan få udbytte af XFDL. Naturligvis kan det have sin berettigelse med traditionel programmering når man vil lave rigtigt snedige ting i en formular, men i XFDL skal der programmeres for tit, og det virker ofte meget omstændeligt hvad der skal til. Eksempelvis er det besværligt at man skal beregne værdien "on" eller "off" for at styre om et kontrolelement skal være synligt eller ej. Det er mere intuitivt blot at angive et boolsk udtryk hvis beregnede værdi angiver synligheden af elementet som i XForms.

Én ting er at der på serveren skal installeres nyt software og at allerede eksisterende servicekode skal skrives om. Noget helt andet er at brugeren *også* skal belemres med at installere en XFDL-processor så han kan udfylde en XFDL-formular.

2.4 Mosquito

Firmaet Mosquito [<http://www.mosquito.com>] er en tredje part der har meldt sig som bannerfører i forbindelse med at gøre formularer på internettet tidssvarende. I stedet for at starte helt fra bunden og designe et nyt formularsprog og dertil hørende værktøjer, har Mosquito valgt at genbruge meget fra HTML-formularer.

I grove træk er Mosquitos forslag, *XHTML-FML* [21], blot en reformulering af HTML-formularer, men – som navnet antyder – udtrykt i XHTML.

Mosquito har på klientsiden valgt at basere sig på allerede eksisterende teknologi, og derfor bringes JavaScript i sving. For at sikre sig at XHTML-FML fungerer i alle browsere oversættes en formular skrevet i XHTML-FML til noget JavaScript-kode der implementerer W3C's DOM-specifikation [25], og videre udtrykkes formularen og dens opførsel så ved brug af denne DOM-implementation. Det må siges at være en særdeles snedig tilgangsvinkel, da brugeren således fritages for problemer med at skulle installere ekstra programmer og derfor umiddelbart kan bruge XHTML-FML – hvis der er nogen der kan levere nogle formularer.

I dokumentationen til XHTML-FML står at læse at *Verification is based on regular expressions* [21], hvorefter samtlige ni regulære udtryk nævnes! Det må siges at være lidt ufleksibelt ...

Ud over selv at finde på et formularsprog har Mosquito haft ild i et andet jern og fået en fremragende ide, nemlig at forsøge at implementere XForms i JavaScript. I *XML WebAccess 2.0* [<http://www.mosquito.org/html/lang-english/xmlwebaccess.html>] er de kommet et langt stykke vej med at implementere XForms i JavaScript, der påstås at kunne køre i alle gængse browsere. Desværre lever det ikke helt op til løftet om at kunne køre i alle browsere, og det kan tydeligt mærkes at HTML og JavaScript ikke er beregnet til at implementere funktionaliteten i XForms. Men uanset hvad man mener om XForms, er det en god ide at forsøge om nye ideer kan implementeres på en måde som umiddelbart gør det muligt for almindelige brugere at få nytte af dem.

2.5 Konklusion

Der er efterhånden mange der har givet deres besyv med i forbindelse med at få webbaserede formularer gjort tidssvarende, og problemet er blevet angrebet fra mange forskellige vinkler. Som den opmærksomme læser vil kunne huske, har W3C brændt nogle broer bag sig og er startet helt forfra med at designe formularer og deres funktionalitet. Der introduceres en adskillelse af formularens data og dens præsentation for brugeren. Et stort problem med XForms er at W3C ikke selv har givet en implementation der kan vise – eller afvise – at XForms kan bruges til noget og har sin berettigelse. Det overlades helt til andre at implementere XForms, og det er tilmed blevet rapporteret som en bug i Mozilla [<http://www.mozilla.org/>] at XForms ikke er implementeret!

Folkene bag XFDL opfatter en webbaseret formular som en modellering af en traditionel papirformular hvor der er en naturlig sammenhæng mellem selve formularen og dens værdier. Under hele formularens livsforløb opretholdes denne sammenhæng så man aldrig kommer i tvivl om hvilken formular en givet mængde data stammer fra.

Både XForms og XFDL baserer sig på nye teknologier og stiller derfor ekstra krav til både programmør og bruger, der begge skal installere nyt programmel for at få udbytte af de nye ideer.

Det turde være klart for enhver der har beskæftiget sig med elektroniske formularer på internettet, at den nuværende teknologi, altså HTML-formularer, ikke lever op til de krav der er rimelige at stille. Det gælder både når man tænker på præsentation for brugeren, der skal udfylde formularen, men også for designeren der skal udforme formularen og for programmøren, der skal skrive programmer til at behandle formularen og de tilhørende værdier.

XForms og XFDL har som mål at bringe elektroniske formularer up to date hvad angår funktionalitet og udtrykskraft, men som anskueliggjort medfører det visse besværligheder for både programmør og bruger, idet man ikke umiddelbart kan anvende formularer der er skrevet i XForms eller XFDL. Der går formentlig endnu nogle år inden de almindelige browsere implementerer XForms eller XFDL, og derfor vil „fru Sørensen fra Tolne“ ikke umiddelbart kunne bruge eksempelvis XForms til noget.

Et andet problem er at store mængder af allerede eksisterende formularer og programmer til at behandle dem, skal omskrives eller skrives helt forfra. Der er mange eksempler på at „gamle“ og gennemtestede programmer lever i bedste velgående, til trods for at der med tiden er kommet bedre værktøjer til at lave tilsvarende programmer – det er jo ikke for sjov skyld at fysikere stadig programmerer i Fortran og lignende arkaiske sprog.

En oplagt ide er at bruge eksisterende teknologi til en faktisk implementation af et nyt apparat til formulardesign, og der er allerede taget et par skridt i den retning, hvoraf ét er Mozquitos forsøg på at implementere XForms i JavaScript.

Kapitel 3

PowerForms

Efter at have kigget på hvad der findes af muligheder for validering af formularer i den store vide verden, er tiden nu kommet til at kigge på PowerForms.

I modsætning til XForms og andre formularformuleringsformalismer som beskrevet i litteraturen [23] er PowerForms noget mere ydmyg i sin tilgangsvinkel til elektroniske formularer, idet det ikke er hele formularbegrebet der står for skud, men blot selve valideringen af input hvad angår lovlige værdier og sammenhæng mellem værdier. Dette gør at PowerForms ved første øjekast virker som en lille undermåler i selskab med de store drenge, men som det forhåbentlig vil blive klart, er PowerForms faktisk et meget udtryksfuldt apparat der løser sin begrænsede opgave på en fornuftig og simpel måde.

3.1 Introduktion

PowerForms er en videreudvikling af en række ideer som dukkede op i foråret 1998 i forbindelse med kurset *WIG Projects* [7, 15], der som dagsorden havde at udforske og -vikle forskellige aspekter af interaktive webapplikationer. Resultaterne opnået i forbindelse med det omtalte kursus viste sig meget lovende, og ideerne blev ført videre i <bigwig>-projektet [8, 9]. Det viste sig hurtigt at PowerForms også har relevans uden for <bigwig>, hvorfor der blev brugt kræfter på at lave et selvstændigt værktøj. Forbindelsen til <bigwig> blev dog bibeholdt under hele udviklingen, og den dag i dag er PowerForms stadig en integreret del af <bigwig>.

Den version af PowerForms der bruges i <bigwig> er beskrevet i en artikel som blev publiceret i år 2000 [8], og det skal retfærdigvis nævnes at den daværende version af PowerForms ikke helt svarer til den nuværende version som beskrives i dette dokument.

Som arvtager til <bigwig> er Jwig [13] nu kommet på banen, og validering af formularer på både klient- og server-siden varetages igen af PowerForms. Nærmere detaljer om dette samarbejde kan findes i afsnit 3.6.

En implementation af PowerForms skrevet i Java kan hentes fra den officielle PowerForms-hjemmeside [<http://www.brics.dk/~ricky/powerforms/>], hvor der også findes en sværm af eksempler på hvad PowerForms kan bruges til.

Regulært udtryk	„Alt der er interessant er regulært“ og derfor er validering i PowerForms hovedsageligt baseret på regulære udtryk. Der er oven i købet tale om regulære udtryk i „bibelsk forstand“, hvilket er den slags regulære udtryk hvortil der svarer en deterministisk endelig automat.
Formularconstraint	Et constraint der gælder for en hel formular og dens feltværdier og således ikke bare for et enkelt felt.
Feltconstraint	Et constraint der stiller krav til værdierne af et enkelt felt på en formular.
Statuselement	Et element som løbende signalerer tilstanden af et constraint til brugeren.
PowerForms-specifikation	Et XML-dokument der beskriver en mængde constraints.
Powertransformation	En transformation af et HTML-dokument ud fra en PowerForms-specifikation.

Figur 9: Nøglebegreber i PowerForms

3.2 Design

Et meget vigtigt designkriterie for PowerForms har været at det umiddelbart skal kunne bruges (til noget), så det er muligt at danne sig et indtryk af hvordan det fungerer i praksis. Derfor må PowerForms nødvendigvis basere sig på teknologier der allerede findes og kan bruges af alle der kan risikere at komme i kontakt med en webbaseret formular. Det er ikke uden problemer at bruge gammel – og af og til gammeldags – teknologi til at realisere helt nye måder at lave formularvalidering på, men hvis det lykkes at overvinde problemerne, vil succesen umiddelbart kunne føles når man ser tingene i aktion.

Siden PowerForms opstod i forbindelse med et værktøj der er baseret på HTML-formularer, er PowerForms ganske naturligt blevet designet til at kunne tilføje ekstra funktionalitet til HTML-formularer uden ændringer i selve formularerne. Det er et velkendt faktum at der findes utallige webbaserede formularer der fungerer ganske glimrende, og i den anledning er det interessant hvis man som salgsargument for sit formularvalideringsværktøj, kan anføre at allerede eksisterende formularer kan drage nytte af valideringen.

I forbindelse med udviklingen af PowerForms har det til tider været nødvendigt med en ikke ubetydelig mængde fantasi og grundige overvejelser for at kunne tilbyde den ønskede funktionalitet under de teknologiske begrænsninger. Af og til har det været nødvendigt at nedsætte ambitionsniveauet på grund af begrænsninger i de anvendte teknologier og enkelte gange har det været nødvendigt at opgive visse ideer, fordi de ikke har kunnet implementeres på tilfredsstillende vis. Til trods for disse begrænsninger er det lykkedes at vise at PowerForms kan bruges til noget i praksis, hvilket man kan overbevise sig om ved at prøve eksemplerne på PowerForms-hjemmesiden [<http://www.brics.dk/~ricky/powerforms/>].

3.3 Validering

Validering af en formular foregår i PowerForms ved brug af *constraints* der hver især er en regel der skal opfyldes af værdierne i formularen. Der findes to slags constraints, og det drejer sig om såkaldte *feltconstraints* og *formularconstraints*. Som navnet antyder gælder et feltconstraint for et enkelt felt, mens et formularconstraint gælder for en formular i sin helhed. Et feltconstraint kan bruges til at sørge for at værdien af feltet overholder bestemte regler, hvorimod et formularconstraint kan bruges til at udtrykke krav til blandt andet relationer mellem forskellige felter på formularen. Brug af de to slags constraints beskrives nærmere i det følgende.

3.3.1 Constraints

Et constraint er et binært beslutningstræ hvor de interne knuder indeholder *boolske udtryk* og hvor bladene indeholder et *udtryk* eller det specielle ignore-element [afsnit 3.4.1] der svarer til at constraintet trivielt er opfyldt.

Det er værd at bemærke, at der i PowerForms refereres til felter ved brug af deres navn (værdien af deres name-attribut) og et feltconstraint gælder derfor for *alle* kontrolelementer med et givet navn på en formular. Felterne med et bestemt navn giver anledning til en række værdier, og det er mængden af disse værdier der undersøges når det skal afgøres om et constraint er opfyldt eller ej. I mange tilfælde vil feltconstraints i sidste ende blot gælde for et enkelt kontrolelement, men hvis man vil stille krav til værdier i afkrydsningsfelter eller lister vil constraintet have indflydelse på mange elementer.

I figur 10 ses et eksempel på en feltconstraint der stiller krav til indholdet af felter med navn „e-mail“.

```
<constraint field="e-mail">
  <match warning="Incomplete e-mail address"
    error="Invalid e-mail address">
    <regexp pattern="[a-z]+\@[a-z]\.[a-z]{2,3}"/>
  </match>
</constraint>
```

Figur 10: Et feltconstraint der kræver, at felter med navn „e-mail“ skal indeholde lovlige e-mail-adresser. For overskuelighedens skyld er det regulære udtryk for e-mail-adresser kraftigt forenklet.

Ud over krav til værdier af bestemte felter vil man også ofte kunne kræve at der skal være bestemte sammenhænge mellem forskellige felter på en formular. En ofte forekommende situation opstår i forbindelse med valg af løsen til en webapplikation, hvor man skal kunne identificere sig overfor applikationen. I sådanne tilfælde ser man ofte to inputelementer med type password hvor man i den andet element skal bekræfte det løsen man har indtastet i det første. En nærliggende løsning på dette problem er at tilknytte et constraint til det andet felt og kræve at værdien af dette er den samme som værdien af det første. Dette giver imidlertid anledning til en vis asymmetri, idet lovlige værdier i det andet felt kommer til at afhænge af det førstes værdi, hvilket ikke er i trit med at relationen „lig med“ (for alle praktiske formål) er symmetrisk.

En pænere løsning er at angive et formularconstraint og i dette angive at de to felter skal have samme værdi. Figur 11 viser et eksempel på et constraint som kræver at to indtastede løsener er ens. Hvis der yderligere garneres med et feltconstraint der kræver at der indtastes et lovligt løsen – hvad det så end er – i det ene felt, er effekten ganske imponerende.

```

<constraint id="matching-password">
  <equal error="Passwords do not match!">
    <field name="password"/>
    <field name="confirm"/>
  </equal>
</constraint>

```

Figur 11: Et simpelt formularconstraint der kræver at felterne „password“ og „confirm“ skal have samme værdi.

3.3.2 Evaluering af udtryk

Når et constraint skal *evalueres*, foregår det ved at evaluere et af bladene i det tilhørende beslutningstræ. Bladet der skal evalueres, findes på traditionel vis ved at finde en sti fra roden af træet til et blad, hvor der undervejs er valgt retning ved at evaluere de boolske udtryk i de interne knuder. Hvordan den faktiske evaluering af udtrykket i bladet forløber, vil blive beskrevet nedenfor.

Ved validering af værdier benytter PowerForms sig blandt andet af regulære udtryk, og hermed menes *alle* regulære udtryk og *rigtige* regulære udtryk! Mange programmeringssprog – deriblandt JavaScript – stiller såkaldte regulære udtryk til rådighed, men det er så som så med regulariteten ...

En fordel ved *regulære* regulære udtryk er at der til hvert af dem findes netop én entydig endelig minimal deterministisk automat [19], og denne *dfa* er endog særdeles effektiv til at validere værdier i elektroniske formularer.

Ideen med at basere inputvalidering på regulære udtryk er ikke ny – både XForms og mange JavaScript-programmører bruger dem – men måden de bruges på i PowerForms, synes at være ny. I kraft af brugen af rigtige regulære udtryk og den deraf følgende medgift, *dfa*'er, kan der nemlig laves *dynamisk* validering af værdier. Dette skal forstås derhen at det ved brug af *dfa*'er er nemt at afgøre om en værdi er lovlig eller om den er „næsten“ lovlig, hvilket vil blive forklaret nærmere i de følgende afsnit.

Som det bør være bekendt, gælder der givet en streng, *s*, og en *dfa*, *a*, enten at *a* *accepterer* *s* eller at *a* *afviser* (rejecter) *s*. Hvis *a* accepterer *s*, ligger *s* i det regulære sprog defineret af *a*. I tilfælde af at *a* ikke accepterer *s*, kan dette skyldes to ting:

1. Efter at have „spist“ alle tegnene i *s* er *a* endt i en ikke-accepterende tilstand.
2. Undervejs i konsumeringen af *s* har *a* nået en tilstand hvorfra der ikke findes en transition der passer til det næste ikke-konsumerede tegn i *s*.

I det første tilfælde siges at „*a* afviser *s*“, mens det andet tilfælde udtrykkes som „*s* crasher *a*“. Bemærk her – og fremover – hvor elegant det gode engelske udtryk „to crash“ umiddelbart kan finde indpas og naturlige bøjningsformer i det danske sprog ...

Det er værd at notere sig at der pr. konstruktion af en minimal *dfa* ikke findes „døde“ tilstande i automaten, og derfor findes en sti fra enhver ikke-accepterende tilstand til en accepterende tilstand. Med andre ord betyder dette at hvis en *dfa* *a* afviser en streng *s*, så findes en (ikke-tom) streng *t* således at *a* accepterer sammensætningen, *st*, af *s* og *t*.

Denne observation er særdeles interessant i forbindelse med validering af værdier i elektroniske formularer. Antag nemlig at valideringen består i at afgøre om den aktuelle værdi tilhører et passende regulært sprog, dvs. om den accepteres af en *dfa*. I så fald betyder det at det for en givet værdi, *s*, som *ikke* accepteres af *dfa*'en, kan afgøres om *s* er et *præfix* af

en streng der accepteres, hvilket i forbindelse med et tekstfelt svarer til at afgøre om det er muligt at nå en accepterende tilstand ved at tilføje tekst til enden af s , altså ved „at skrive noget mere“.

Omvendt gælder at hvis s crasher en dfa, så kan s ikke forlænges til at blive accepteret af dfa'en. Udtrykt med et tekstfelt i mente betyder det at den aktuelle værdi i feltet aldrig vil kunne lede til en accepterende tilstand, men at det er nødvendigt at slette noget af teksten indtil man når en tilstand der enten er accepterende eller hvorfra en accepterende tilstand kan nås.

Populært sagt kan validering af værdier i et tekstfelt med et regulært sprog lede til tre situationer:

1. „Alt er fint!": værdien accepteres.
2. „Der er håb endnu": den underliggende dfa afviser værdien, men er ikke crashet.
3. „Det går aldrig godt": dfa'en er crashet, og der må sving i slettetasten.

Disse tre tilstande som en værdi i en formular kan give anledning til, kan bruges til at lave dynamisk validering ved løbende at informere brugeren om hvorvidt den foreløbige værdi kan lede til en accepteret værdi.

Når man arbejder med validering af værdier i elektroniske formularer bliver det hurtigt klart at regulære udtryk ikke altid er nok, idet de kun kan bruges til at tjekke enkelte værdier, og man af og til ønsker at kunne tjekke om to værdier er ens. Som netop anskueliggjort muliggør brugen af regulære udtryk dynamisk validering, hvor en værdi hele tiden giver anledning til én af tre „lovlighedsværdier“, og noget tilsvarende vil være ønskeligt når andre interessante udsagn skal evalueres.

Ved nærmere overvejelse viser det sig at mange interessante udsagn kan evalueres således at der til ethvert tidspunkt kan svares „udsagnet er opfyldt“, „udsagnet er måske opfyldt“, eller „udsagnet kan ikke opfyldes“. Det er oplagt hvad det betyder at et udsagn er opfyldt, men de to sidste betegnelser kræver lidt nærmere forklaring:

Inspireret af situationen med validering ud fra et regulært sprog, hvor „håbet er lysegrønt“ og der kan tilføjes tekst og opnås accept, er det i PowerForms-verdenen vedtaget at et udsagn er „måske opfyldt“ hvis det er muligt at tilføje information ved for eksempel at tilføje tekst i tekstfelter eller markere afkrydsningsfelter og derved opfylde udsagnet. Eksempelvis gælder at udsagnet „12 er lig med 123“ er „måske opfyldt“, idet „12“ kan forlænges til „123“ som pr. symmetri af relationen „lig med“ er lig med „123“. Hvis man er i en situation hvor et udsagn ikke er opfyldt, men det er muligt at opfylde det, svares således at det er „måske opfyldt“ frem for at sige at udsagnet ikke er opfyldt.

At et udtryk „ikke kan opfyldes“, svarer til situationen hvor en værdi crasher en dfa – altså hvor der må fjernes information for at kunne få værdien accepteret. Som et eksempel på et udsagn der ikke kan opfyldes kan tages „-1 er større end 7“, idet hverken „-1“ eller „7“ kan forlænges således at udsagnet bliver opfyldt.

En konsekvens af ønsket om at kunne evaluere udsagn til tre forskellige tilstande, hvor den ene angiver at der kan tilføjes information for at opfylde udsagnet, er at det er ikke-trivielt at bruge sammensatte udsagn som constraints. Sammensatte udsagn dækker over konstruktioner som „og“, „eller“ og „ikke“ og for disse er det ikke ligetil at beregne en passende tilstand for sammensætningen, og derfor tillades disse udtryk kun som boolske udtryk i PowerForms. Set i bagklogskabens klare skær er dette egentlig en meget rimelig begrænsning for feltconstraints, idet det må anses som en formulardesignfejl hvis det kræves at værdien af et felt er både et tal og en e-mail-adresse.

Som antydnet ovenfor kan udsagn også fungere som traditionelle boolske udsagn, og i så fald opfattes udsagnet som værende sandt hvis og kun hvis det er opfyldt.

Da en evaluering af et constraint i sidste ende sker ved evaluering af et udsagn, siges at et constraint „er opfyldt“, „måske er opfyldt“ eller „ikke kan opfyldes“ svarende til tilstanden af det evaluerede udsagn.

3.4 Brugerinteraktion

I PowerForms laves der dynamisk validering, hvilket vil sige at constraints evalueres hver gang brugeren ændrer en værdi i et felt på en formular. Et formularfelt ændrer værdi når teksten i et tekstfelt ændres eller en knap (af)markeres, og når det sker skal alle berørte constraints genevalueres. De „berørte“ constraints er alle feltconstraints for det ændrede felt og alle andre constraints der afhænger af værdien af feltet og afsnit 3.7 beskriver hvordan det afgøres præcist hvilke constraints der skal genevalueres og hvordan evalueringen faktisk foregår.

3.4.1 Visualisering

Én ting er at constraints genevalueres på passende tidspunkter. En anden ting er hvordan resultatet af evalueringen præsenteres for brugeren og netop dette er emnet som nu vil blive taget op.

Når et constraint evalueres foregår det, som beskrevet i afsnit 3.3, ved at bevæge sig fra roden i det tilhørende beslutningstræ ned til et blad, og så evalueres udtrykket i bladet for at afgøre om constraintet er overholdt eller ej. Som beskrevet tidligere kan evaluering af udtrykket resultere i tre forskellige tilstande, og disse tilstande skal afspejles i den fysiske præsentation af formularen så brugeren hjælpes til at udfylde en formular korrekt.

Hvis det aktuelle constraint er et feltconstraint, vil navnet på feltet som constraintet gælder for udpege et eller flere kontrolelementer i en formularen. For hvert af disse elementer evalueres udtrykket ud fra elementets værdi og præsentationen af resultatet afhænger af typen af kontrolelementet. Nedenfor vil det blive beskrevet hvordan resultatet af evalueringen giver sig til kende for de implicerede kontrolelementer.

Groft sagt kan kontrolelementer i en HTML-formular opdeles i to grupper, nemlig tekstfelter og knapper, og disse to grupper adskiller sig væsentligt i den måde hvorpå de afspejler tilstanden af et tilknyttet feltconstraint.

Tekstfelter

Gruppen „tekstfelter“ dækker over input-elementer med type text eller password. Som bekendt kan værdien i et tekstfelt medføre at det tilhørende constraint evaluerer til én af følgende tre tilstande:

- „opfyldt“, hvis værdien i tekstfeltet opfylder alle krav
- „måske opfyldt“, hvis værdien i tekstfeltet ikke opfylder alle krav, men er et præfiks af en værdi der gør
- „ikke opfyldt“, hvis tekstfeltet indeholder en ulovlig værdi

Da det i sin tid, altså i de glade WIG Projects-dage [7], blev overvejet hvordan de forskellige tilstande skal signaleres til brugeren blev der i hjernerne på hinanden tænkt: „vi farver teksten, gør vi!“ Efterfølgende blev der gjort alverdens forsøg på at farve teksten i et tekstfelt i en HTML-formular, men selv de mest velmente hacks kunne ikke løse problemet på tilfredsstillende vis. Fantasien blev så bragt i sving og en ny ide opstod: „Et billede ved siden af tekstfeltet kan vise et lille trafiklys der viser den aktuelle tilstand!“ Umiddelbart lyder det måske lidt barnligt, men teknologien var på daværende tidspunkt ikke bevendt til andet – og visse akademiske browsere er stadig ikke kommet videre ...

Ved nærmere eftertanke er et lille *statusikon* ved siden af et tekstfelt ikke så dumt endda. Det er i hvert fald illustrativt – og forhåbentlig intuitivt. Da ideen første gang blev bragt på banen var det soleklart hvordan trafiklyset skal bruges: når constraintet er opfyldt vises grønt lys, når constraintet måske er opfyldt vises gult lys og hvis constraintet ikke kan opfyldes vises rødt lys. Ved nærmere omtanke virker det underligt nok ikke videre intuitivt, hvorfor det måske netop *er* intuitivt. Adskillige sagesløse brugere af PowerForms har i hvert fald umiddelbart fattet hvad der foregår!

Hvorom alt er, viser figur 12 hvordan et tekstfelt med tilknyttet statusikon kan tage sig ud i en browser.



Figur 12: Illustration af dynamisk validering med et regulært udtryk, hvor et statusikon signalerer tilstanden af et constraint.

Ovenstående beskrivelse har beskrevet hvad der sker når et constraint evaluerer til én af tre tilstande, men som det svagt blev antydnet i afsnit 3.3.1 findes faktisk en fjerde mulighed for resultatet af en constraintevaluering. Det er det specielle ignore-element, som angiver at det berørte felt – eller rettere *værdien af* feltet – er ligegyldig (skal ignoreres), hvilket videre betyder at værdien må være hvad som helst. For alle beregningsmæssige formål svarer dette helt til enten ikke at knytte et constraint til feltet eller at tillade alle værdier, men præsenteringsmæssigt er der forskel. Statusikonet for et felt der skal ignoreres vil vise et billede der signalerer dette og som standard vises et billede som illustreret i figur 13, hvor „NA“ er inspireret af betegnelsen „not applicable“ som den ses på engelsksprogede formularer.



Figur 13: Et illustrativt statusikon for et felt hvis værdi er „ligegyldig“.

Hvis man gransker afsnittet om syntaksen af PowerForms-dokumenter og mere specifikt koncentrerer sig om status-elementet [afsnit 5.6], vil man bemærke at dette element har en

række attributter med navnene *red*, *yellow*, *green* og *na*. Den vakse læser vil formentlig have en begrundet mistanke om hvad disse attributter bruges til og hvorfor de hedder som de gør og som en facitliste kan kapitel 5 konsulteres.

Som det fremgår af „PowerForms-manualen“ i kapitel 5, kan tilstanden af et constraint også vises ved brug af *Cascaded Style Sheets* (CSS) [5] eller tekstbeskeder, hvor tekstbeskeder dog kræver endog *meget* moderne browsere. Ved brug af CSS kan den oprindelige ide med at farve teksten i et tekstfelt, alt efter om den er lovlig eller ej, realiseres, men det synes ikke at virke nær så intuitivt som det lille trafiklys. En lækker detalje som kan observeres når man bruger CSS-status-elementer er, at tekstfelter der skal ignoreres bliver gjort inaktive, således at man ikke kan skrive i dem [figur 24].

PowerForms giver en mulighed for manuelt at tilknytte status-elementer til tekstfelter, og det er derved muligt at tilpasse udseendet af status-elementet til hvert tekstfelt. Hvis et tekstfelt har tilknyttet et status-element, vil PowerForms naturligvis undlade automatisk at tilføje et. Det er endvidere muligt at tilføje et status-element til at formularconstraint. Nærmere detaljer om status-elementer er at finde i afsnit 5.6.

Knapper

„Knapper“ er et vidt begreb og dækker i PowerForms-verdenen over input-elementer med type *radio* og *checkbox* og *select*-elementer.

På samme måde som tekstfelter får tilknyttet et statusikon der viser tilstanden for et tilknyttet feltconstraint, får knapper det ikke. I stedet bliver de „filtreret“ således at det ikke er muligt at forbyrde sig mod et constraint ved at trykke på de rigtige – eller snarere de *forkerte* – knapper. For *radio*-knapper og *afkrydsningsfelter* vil det sige at de kun kan markeres, hvis deres værdi overholder constraintet, mens *select*-elementer kun indeholder indgange der overholder constraintet.

I figur 14 ses et *select*-element ved navn *letter* der er underlagt følgende constraint:

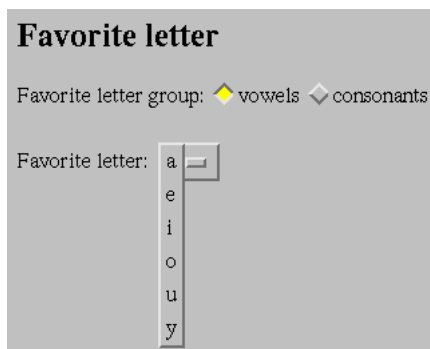
```
<constraint field="letter">
  <if><equal field="group" value="vowel"/>
    <then><match><charset value="aeioy"/></match></then>
    <else><match><charset value="bcdfghjklmnpqrstvwxyz"/></match></else>
  </if>
</constraint>
```

Der loves hermed højt og helligt – og på tro og love – at *select*-elementet indeholder 26 *option*-elementer på formen

```
<option value="σ"> σ </option> ,
```

for $\sigma = \{a, b, \dots, z\}$, men som det fremgår af figuren, er det kun vokalerne der kan vælges, når *radio*-knappen „*vowels*“ er markeret. Den tvivlende eller blot interesserede læser kan prøve eksemplet på <http://www.brics.dk/~ricky/powerforms/examples/letter.html>, hvortil det skal bemærkes at kommaet ikke er en del af url'en ...

Det er i PowerForms kun muligt at tilknytte constraints til tekstfelter, *radio*-knapper, *afkrydsningsfelter* og *lister* som omtalt ovenfor. I HTML findes der imidlertid flere kontrol-elementer, men da de ikke kan tilknyttes constraints, vil de naturligvis ikke være påvirket af PowerForms, men fungere ganske som i almindeligt HTML.



Figur 14: Et dynamisk select-element der er underlagt et constraint.

3.4.2 Automatisk fuldførelse

Når det er muligt at afgøre om en værdi i et tekstfelt er lovlig eller ej i henhold til et feltconstraint, er det nærliggende at spørge: „er det muligt at afgøre *hvilke* værdier der er lovlige?“ Som beskrevet kan et constraint være „måske opfyldt“ hvilket som bekendt betyder at der kan tilføjes tekst i et tekstfelt således at constraintet bliver opfyldt. Et mere relevant spørgsmål i den anledning er så: „*hvilke tilføjelser* vil gøre constraintet opfyldt?“.

Disse spørgsmål – og deres eventuelle svar – leder tanken hen på noget der i mistænkelig grad minder om automatisk fuldførelse som kendt fra utallige mere eller mindre brugervenlige computerprogrammer. Traditionelt foregår automatisk fuldførelse ved at programmet står og lurer på brugerens indtastninger, og hvis indholdet i et tekstfelt er et præfiks af en værdi der engang har stået i feltet, tilbydes automatisk fuldførelse. Det interessante – eller måske uinteressante – ved dette er, at det kun er gamle værdier der automatisk kan fuldføres. Automatisk fuldførelse er mere anvendeligt hvis alle lovlige værdier – „nye“ som „gamle“ – kan fuldføres.

En nærmere overvejelse over spørgsmålene viser at det søgte svar er noget i retning af: „ja, langt hen ad vejen er det faktisk muligt at afgøre hvilke tilføjelser der vil gøre constraintet opfyldt!“.

Et rimeligt krav at stille i forbindelse med automatisk fuldførelse er at constraintet der søges opfyldt ikke allerede *er* opfyldt, men at det *kan* opfyldes, dvs. at dets tilstand skal være „måske opfyldt“. Heraf følger at en eventuel automatisk fuldførelse er en ikke-tom streng, hvilket må siges at være særdeles rimeligt.

Et nyt spørgsmål trænger sig på, og det drejer sig om hvilken slags fuldførelse der skal laves, og hvis der er flere mulige fuldførelser, hvilken der så skal vælges. Følgende tre former for automatisk fuldførelse virker fornuftige:

Entydig Hvis der findes én og kun én lovlig fuldførelse der vil gøre constraintet opfyldt fuldføres der med denne.

Præfiks Fuldførelse hvis der findes én og kun én ikke-tom streng der enten vil gøre constraintet opfyldt eller måske opfyldt. Dette er en variant af entydig fuldførelse hvor fuldførelser, der ikke nødvendigvis gør constraintet opfyldt, tillades.

First Fuldførelse med den leksikalsk mindste ikke-tomme streng der vil opfylde constraintet.

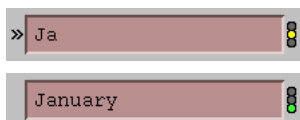
Tilbage står så at overveje for hvilke udsagn det er muligt på fornuftig vis at bestemme mængden af lovlige fuldførelser. I første omgang er kun tekstfelter kandidater til automatisk fuldførelse og så kan nedenstående udsagn føre til en veldefineret mængde lovlige fuldførelser:

match At constraintet måske er opfyldt, svarer til at den bagvedliggende dfa er i en ikke-accepterende tilstand. Entydig fuldførelse (inklusive præfiks-varianten) er mulig hvis der findes en entydig sti fra den nuværende dfa-tilstand til en accepterende tilstand. Fuldførelse med den leksikalsk mindste værdi laves ved at lave en bredde-først-søgning i dfa'en, idet transitionerne undersøges i leksikografisk rækkefølge.

equal Hvis værdien der ønskes fuldført skal være lig en bestemt værdi, er fuldførelse mulig når værdien er et præfiks af den „bestemte“ værdi. I dette tilfælde er entydig og leksikografisk mindst fuldførelse det samme. Hvis kravet er lighed med én af flere værdier, undersøges om den ufuldførte værdi er et præfiks af en af de „flere“ værdier, og hvis den er et præfiks af netop én værdi, er entydig fuldførelse mulig. Hvis den er et præfiks af mere end én værdi, er leksikografisk mindst fuldførelse mulig.

Når automatisk fuldførelse i et tekstfelt i diverse programmer er mulig, giver det sig ofte til kende ved at fuldførelsen vises i selve tekstfeltet, men „markeret“ så brugeren er klar over at noget er på færde. Eftersom den almindelige bruger er vant til denne opførelse, vil noget tilsvarende være ønskeligt i forbindelse med automatisk fuldførelse i PowerForms. Imidlertid viser mulighederne – eller snarere begrænsningerne – i browsere sig igen fra sin værste side, idet det kun i ganske få browsere er muligt at markere noget tekst automatisk, så igen må fantasien i sving. Opgaven er altså at signalere til brugeren at automatisk fuldførelse er mulig, og en måde at gøre dette på ligger forbløffende nær.

Som bekendt har PowerForms allerede introduceret statusikoner til højre for tekstfelter, hvilket efterlader venstresiden af tekstfeltet bar og nøgen, så hvorfor ikke putte et billede der signalerer hvorvidt automatisk fuldførelse er mulig ind dér? Der skal bruges to billeder: ét der vises når automatisk fuldførelse er aktuel og ét der vises ellers. Et eksempel på hvordan det kan se ud er afbildet i figur 15, hvor billedet der viser at automatisk fuldførelse ikke er mulig, er gennemsigtigt.



Figur 15: Et „før og efter“-billede af et tekstfelt med automatisk fuldførelse. Til venstre ses et ikon der indikerer at automatisk fuldførelse er mulig og til højre ses statusikonet for feltet.

I forbindelse med automatisk fuldførelse er der endnu en måde man kan forestille sig det gjort på, nemlig ved at præsentere brugeren for en liste af mulige fuldførelser, som han så kan vælge fra. I praksis er det dog meget svært at implementere denne funktionalitet i en webbrowser, så PowerForms understøtter (endnu) ikke dette.

Kapitel 5.7 beskriver hvordan man bringer automatisk fuldførelse i aktion.

Ovenstående beskriver kun automatisk fuldførelse af værdier i tekstfelter, men det er indlysende at fuldførelse også er relevant for andre kontrolelementer. Eksempelvis kan afkrydsningsfelter automatisk afkrydses hvis det kan afgøres at kun ét bestemt af dem kan være afkrydset. For at lave den slags automatisk udfyldning af en formular, kræves en noget mere involveret analyse af PowerForms-specifikationen og det transformerede HTML-

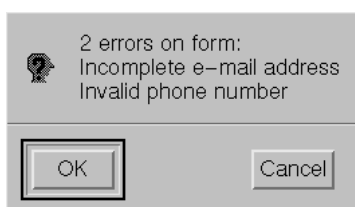
dokument end der laves for tiden, så det må vente til en senere version af PowerForms, at muliggøre denne form for udfyldning.

3.4.3 Submit

Når brugeren under kyndig vejledning fra trafiklys og andet hurlumhej har udfyldt en formular på bedste beskub, skal den på et tidspunkt submittes. Dette foregår ved at brugeren aktiverer et submit-element, hvor „submit-element“ dækker over en traditionel HTML-submit-knap eller et element PowerForms introducerer i samme øjemed [afsnit 5.9.1].

I forbindelse med en brugers interaktion med en webservice skal det til tider være muligt for brugeren at fortryde udfyldelsen af formularen og submitte den til trods for at tilknyttede constraints ikke er overholdt. PowerForms understøtter dette, idet der i et submit-element kan angives hvorvidt constraints skal tjekkes når elementet aktiveres.

Før en formular kan submittes på normal vis, skal det tjekkes at alle constraints for selve formularen og for felter på den er opfyldt. Hvis alt er i den fineste orden, submittes formularen nøjagtig som det foregår i HTML. Ellers vises en fejlbesked a la den der er afbildet i figur 16, og formularen submittes ikke. En årvågen PowerForms-bruger har undret sig over at der både er en „OK“- og en „Cancel“-knap under fejlbeskeden, men dette skyldes udelukkende begrænsende muligheder i browsere – det er hip som hap om brugeren klikker på den ene eller den anden knap efter at have læst beskeden.



Figur 16: En fejlbesked produceret af PowerForms ved et forsøg på at submitte en formular der ikke er korrekt udfyldt.

3.5 Eksemplet

Ovenfor er givet et par eksempler på hvordan man specificerer constraints i PowerForms, men et større eksempel er på sin plads, og som lovet bringes den illustrative formular fra afsnit 2.1 igen på banen, og opgaven er nu at få de stillede krav formuleret i PowerForms.

Figur 10 viste hvordan man får styr på værdien i e-mail-feltet, men som bemærket var der en lidt lemfældig omgang med lovlige e-mail-adresser, så her vises den „rigtige“ måde at kræve en lovlig e-mail-adresse på:

```
<constraint field="e-mail">  
  <match warning="Incomplete e-mail address"  
    error="Invalid e-mail address">  
    <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/e-mail.dfa"/>  
  </match>  
</constraint>
```

I stedet for at konstruere et regulært udtryk for en lovlig e-mail-adresse inkluderes en allerede oversat dfa der accepterer lovlige e-mail-adresser fra en url. Det regulære udtryk for

e-mail-adresser er konstrueret ud fra definitionen af en lovlig e-mail-adresse [14] og overholder derfor alle regler i den henseende. Som sidebemærkning kan det nævnes, at den minimale deterministiske automat der accepterer lovlige e-mail-adresser har 12 tilstande.

Det næste projekt er feltet benævnt „Zip code“. Let omskrevet i forhold til formulering i afsnit 2.1 lyder kravet: „hvis landet er USA skal man skrive en lovlig zip-kode; ellers må der stå hvad som helst“. Ved brug af et prædefineret regulært udtryk for lovlige zip-koder, ser det i PowerForms således ud:

```
<constraint field="zip-code">
  <if>
    <equal field="country" value="usa"/>
    <then>
      <match>
        <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/zip-code.dfa"/>
      </match>
    </then>
    <else>
      <ignore/>
    </else>
  </if>
</constraint>
```

Bemærk den nærmest prosaagtige beskrivelse: „Hvis landet er USA, så skriv en zip-kode; ellers kan det være lige meget“. Enhver bedstemor med respekt for sig selv og kendskab til XML vil kunne opskrive dette PowerForms-constraint!

Telefonnummerfeltet har et constraint der minder meget om constraintet for zip-kodefeltet:

```
<regexp-def id="sep">
  <optional><charset value="-"/></optional>
</regexp-def>

<regexp-def id="digit">
  <charset value="0123456789"/>
</regexp-def>

<constraint field="phone" error="Invalid phone number">
  <if>
    <equal field="country" value="usa"/>
    <then>
      <concat>
        <repeat count="3">
          <regexp idref="digit"/>
        </repeat>
        <regexp idref="sep"/>
        <repeat count="3">
          <regexp idref="digit"/>
        </repeat>
        <regexp idref="sep"/>
        <repeat count="4">
          <regexp idref="digit"/>
        </repeat>
      </concat>
    </then>
```



```

    <else>
      <concat>
        <repeat count="8">
          <regexp idref="digit"/>
        </repeat>
      </concat>
    </else>
  </if>
</constraint>

```

Man kan diskutere langt og længe hvordan telefonnumre skal skrives, men for eksemplets skyld er det rigeligt at vedtage at et dansk telefonnummer består af otte på hinanden følgende cifre, mens et amerikansk telefonnummer består af ti cifre med eventuelle bindestreger eller mellemrum hist og her.

Ovenstående eksempel viser hvordan et regulært udtryk kan opbygges ved brug af XML-elementer, hvilket er praktisk hvis udtrykket skal opbygges dynamisk i forbindelse med eksempelvis en JWIG-service [13] som omtalt i afsnit 3.6. Som det fremgår fylder, XML-strukturen dog en del til trods for at udtrykket er forholdsvis simpelt. For simple regulære udtryk vil man derfor nok oftest foretrække en noget kortere skrivemåde, og PowerForms understøtter da også dette, idet man kan opskrive et regulært udtryk i en tekststreng på stort set samme måde som i utallige programmeringssprog. Constraintet for telefonnummeret kan således også skrives på følgende måde:

```

<constraint field="phone" error="Invalid phone number">
  <if>
    <equal field="country" value="usa"/>
    <then>
      <regexp pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"/>
    </then>
    <else>
      <regexp pattern="[0-9]{8}"/>
    </else>
  </if>
</constraint>

```

Endelig skal der styr på knappen hvormed man kan bede om at få besøg fra kontoret i New York, og som beskrevet i afsnit 2.1 er effekten der ønskes at det kun er muligt at markere feltet hvis man bor i USA og ens telefonnummer starter med et bestemt tal. Som beskrevet i afsnit 3.4.1 kan dette klares ved at tillade alle værdier i feltet hvis dette krav er opfyldt og ellers ikke tillade nogen værdier. Her viser de regulære udtryk defineret af **<anything/>** og **<empty/>** [afsnit 5.4] deres duelighed, og en passende PowerForms-besværgelse ser ud som følger:

```

<constraint field="visit">
  <if>
    <and>
      <equal field="country" value="usa"/>
      <match field="phone">
        <concat>
          <union>
            <const value="212"/><const value="347"/><const value="646"/>
            <const value="718"/><const value="917"/>
          </union>
          <anything/>
        </concat>
      </match>
    </and>
  </if>
</constraint>

```

```

    <then><match><anything/></match></then>
    <else><match><empty/></match></else>
  </if>
</constraint>

```

Når alle disse constraints samles og puttes ind i et powerforms-element, haves en komplet PowerForms-specifikation som kan inspiceres i appendiks A.3. Inden den opbyggede specifikation hældes gennem PowerForms-oversætteren, skal der laves en lille ændring i det oprindelige HTML-dokument. Ændringen består i at tilføje en ignoreconstraints-attribut med værdien yes til "Cancel"-knappen, således at brugeren kan fortryde udfyldelsen af formularen uden at han bliver præsenteret for en masse fejlbeskeder:

```

<input type="submit" name="Submit" value="Cancel"
       ignoreconstraints="yes"/>

```

Efter at have kørt PowerForms-specifikationen og det let modificerede HTML-dokument gennem PowerForms-oversætteren, fås et resultat som illustreret i figur 17.

Figur 17: Den illustrative formular fra figur 1 efter en powertransformation, men før brugeren begynder at udfylde den.

Billedserien i figur 18 viser formularens omskiftelige udseende mens en bruger pådutter den de samme begivenheder som beskrevet for eksemplet under XForms og XFDL. Det fremgår tydeligt at PowerForms-versionen ikke er helt så „action packed“ i forbindelse med felter der dukker op og forsvinder.

Til gengæld er især transitionen fra tilstand (c) til tilstand (d) interessant, når det holdes for øje at det eneste brugeren gør er at skifte land fra USA til Danmark. Dette skifte har flere konsekvenser:

For det første skal der ikke længere angives en zip-kode, hvilket vises ved at statusikonet til højre for zip-kode-feltet skifter til det specielle „n/a“-symbol. For det andet er det indtastede telefonnummer ikke længere lovligt, idet det som bekendt er et amerikansk telefonnummer og derfor ikke overholder de opstillede regler for hvordan et dansk telefonnummer ser ud. Derfor markeres værdien i „Phone“-feltet som værende ulovlig via det røde trafiklys. Endelig er markeringen i afkrydsningsfeltet nederst i formularen blevet fjernet, fordi brugeren ikke længere bor i USA og derfor heller ikke bor i New York. Hvis brugeren i stedet for at skifte land blot havde ændret sit telefonnummer, så det ikke længere hører hjemme i New York, ville markeringen i afkrydsningsfeltet også forsvinde.

For at afprøve denne påstand og i øvrigt danne sig et indtryk af hvordan eksemplet fungerer i praksis, kan det fuldt funktionsdygtige eksempel afprøves for enden af følgende url: <http://www.brics.dk/~ricky/powerforms/examples/NYC.html>.



Figur 18: Det illustrative eksempel i fuldt sving og PowerForms.

3.6 PowerForms i Jwig

Som antydnet under introduktionen af PowerForms benyttes PowerForms til validering af formularer i Jwig [13], og dette afsnit vil give nærmere detaljer om hvordan det foregår.

Jwig tilbyder opbygning af *dynamiske dokumenter* ud fra skabeloner med en række navngivne *huller* i. Dokumenterne der kan opbygges, er generelle XML-dokumenter, og det er oven i købet muligt at kontrollere gyldigheden af de opbyggede dokumenter i henhold til en *Document Structure Description* (DSD) [20]. Da et PowerForms-dokument netop er et XML-dokument, er det altså i Jwig muligt dynamisk at opbygge en PowerForms-specifikation og derefter kontrollere at den overholder den officielle DSD for PowerForms-dokumenter [<http://www.brics.dk/~ricky/powerforms/powerforms.dsd>].

JWIG benytter PowerForms til validering både på klienten og på serveren, for som det vil blive klart om et øjeblik, kan man ikke forlade sig på klientvalidering alene. Det kan ikke overraske, at den validering af værdier der skal laves på serveren, er identisk med den der laves på klienten, og det er derfor oplagt at lade PowerForms klare begge dele. Yderligere kan det gøre selve JWIG-koden betragteligt mere overskuelig hvis valideringen kan klares „et andet sted“, så programmøren kan bekymre sig om selve servicen og ikke behøver at bryde sit lille hoved med at skrive valideringskode ud over en række PowerForms-constraints.

3.6.1 Serverside-validering

En af de væsentligste grunde til at man er interesseret i at validere en brugers indtastninger i en elektronisk formular, er at man – hvis valideringen er succesfuld – kan tillade sig at gøre antagelser om de indtastede værdier. Som det efterhånden turde være klart, muliggør PowerForms validering af data på klienten, således at man ikke kan sende data tilbage til serveren før de angivne krav er overholdt. Imidlertid er det nødvendigt at tjekke data en ekstra gang når de kommer tilbage til serveren, og dette kan der være flere årsager til:

1. Brugers browser understøtter ikke JavaScript eller han har slået det fra. Der er desværre nogle personer i denne verden der ikke kan opføre sig ordentligt, og det har hos folket medført en vis modvilje mod at lade JavaScript køre i deres browsere.
2. PowerForms har genereret noget JavaScript der ligger (delvist) uden for den mængde JavaScript der understøttes i brugers browser, hvorfor validering på klienten ikke er udført korrekt.
3. Der er nogen der forsøger at snyde. Der kan umiddelbart snydes på to måder: man kan foregive at have udfyldt formularen uden faktisk at have gjort det, eller man kan være ekstra fræk og ændre i brugers data under transmissionen fra klient til server.

Den sidste fejlkilde er noget mere alvorlig end de første, men som så meget andet ligger det uden for dette speciale at bekymre sig om dette alvorlige problem.

Til trods for at alle værdier er blevet tjekket på klienten, er det, for at være på den sikre side, altså nødvendigt at kontrollere dem endnu en gang når de ankommer til serveren. Kontrollen, der skal laves på serveren, er naturligvis den samme som der laves på klienten, og det er derfor oplagt at generere nogle stumper kode til serverside-validering ud fra den samme specifikation som er brugt til generering af klientside-valideringen.

Da der kan være flere formularer i et HTML-dokument, indsætter PowerForms under transformationen i hver formular et skjult felt således at den formular der faktisk bliver submittet kan identificeres.

Det første der gøres når PowerForms laver validering på serveren, er at afgøre om den submittede formular er blevet submittet med et submit-element der har ignoreret constraints. Dette klares ved at PowerForms under transformationen af et HTML-dokument holder styr på hvilke submit-elementer der har en ignoreconstraints-attribut med værdien „yes“. Disse elementer definerer en mængde af par bestående af et submit-navn og en submit-værdi, og når formularværdierne kommer tilbage til serveren, kan det således afgøres om constraints skal ignoreres eller ej. I JWIG forholder det sig således, at det kun er navnet på det element der er blevet brugt til at submitte en formular der kan aflæses, og man kan derfor bede PowerForms om kun at undersøge submitnavnet for at afgøre om constraints skal ignoreres.

Hvis constraints skal ignoreres, sker der naturligvis ikke yderligere validering på serveren.

I sidste ende består validering på serveren naturligvis i at afgøre om alle relevante constraints i en PowerForms-specifikation er overholdt, men inden man når så langt, er der nogle indledende knæbøjning der passende kan foretages:

Fase 1

En PowerForms-specifikation angiver nogle krav som de returnerede værdierne skal overholde, men den indeholder ingen oplysninger om hvilke værdier der skal være til stede, da netop dette hænger tæt sammen med det aktuelle HTML-dokument der er blevet transformeret. Når værdierne på en elektronisk formular skal behandles på serversiden, skal det rimeligvis kunne antages at værdierne af de felter der findes på en formular findes i svaret der sendes til serveren. Det første der således skal foretages på serveren, er at undersøge om alle værdier er til stede, og først derefter kan det kontrolleres om de tillige overholder kravene der er formuleret i PowerForms.

For at få styr på hvilke værdier der returneres når en HTML-formular submittes holder PowerForms styr på en *øvre* en og *nedre* grænse for antallet af værdier et formularfelt kan have. Hvis det faktiske antal ligger udenfor intervallet bestemt af disse grænser, er noget gået galt, nogen har snydt eller grænserne er forkerte . . .

Som bekendt findes der forskellige kontrolelementer i HTML, og disse elementer kan producere et forskelligt antal værdier. Eksempelvis vil et tekstfelt altid give anledning til netop én værdi, mens en radioknap kun returnerer en værdi hvis den er markeret. En HTML-ekspert vil her indvende at foregående sætning faktisk ikke er helt sand, idet et kontrolelement skal være *succesfuldt* [22, afsnit 17.13.2] for at dets værdi er med når data sendes tilbage til serveren, men for alle praktiske PowerForms-formål er alle kontrolelementer succesfulde.

For at studere de præcise detaljer om succesfulde kontrolelementer sidder den flittige læser allerede med kapitel 17 i HTML-specifikationen [22] foran sig. Afsnit 17.13 i denne beskriver i stor detalje hvor mange eller hvor få værdier hvert enkelt kontrolelement kan producere, når en HTML-formular submittes.

Hvor mange værdier hvert enkelt kontrolelement kan give anledning til er resumeret i førstkommende tabel.

Kontrol-element	Antal værdier
input	
text	1
password	1
checkbox	0 eller 1
radio	0 eller 1
select	
single	1
multiple	højst antallet af indgange
textarea	1

I forbindelse med lister, hvori kun én indgang kan vælges, og grupper af radioknapper er der en enkelt krølle på historien:

Da HTML i sin tid blev defineret [1], var det et krav til browsere at de skulle sørge for at der *altid* var valgt netop én radioknap i hver gruppe og at der altid var valgt netop én indgang i en liste. Grundet sjuske(n)de browserprogrammører er der imidlertid sket det at dette krav er blevet fjernet fra den nuværende HTML-specifikation [22], og det overlades nu til HTML-skribentens samvittighed at sørge for at der altid er valgt noget – og ham kan

man ikke stole på! Derfor er det ikke veldefineret om der returneres ingen eller én værdi fra en gruppe af radioknapper, og for en sikkerheds skyld må det antages at der returneres mindst nul værdier og højst én.

For at bestemme den øvre og nedre grænse for antallet af værdier for et bestemt felt, summeres ganske enkelt over alle feltets kontrolelementer, idet man husker at en gruppe af radioknapper højst kan returnere én værdi.

Inden værdierne sendes videre til validering ud fra selve PowerForms-specifikationen, er der endnu et tjek det er på sin plads at lave:

Fase 2

Én ting er at afgøre om antallet af værdier for et felt ligger inden for grænserne defineret i HTML-formularen, én anden ting er at afgøre om værdierne er *lovlige*. Her menes ikke „lovlige“ i henhold til PowerForms-specifikationens constraints, men i henhold til den faktiske HTML-formular. Problematikken er, at hvis et felt på en formular kun indeholder et select-element, så er det naturligvis en fejl hvis det på serveren viser sig at feltet indeholder en værdi der *ikke* er værdien af en indgang i listen. Derfor bør man på serveren også kontrollere at de returnerede værdier fra en formular faktisk kan vælges på formularen som den i sin tid blev sendt ud til klienten. PowerForms understøtter i sin nuværende udformning ikke denne form for tjek af værdier, men det vil uden de store anstrengelser kunne tilføjes.

Hvis det viser sig at der mangler nogle værdier fra klienten, producerer PowerForms en fejlmeddelelse som det er op til Jwig-programmøren at håndtere. Ellers er det endelig tid til at lave selve PowerForms-valideringen, hvor det undersøges, om alle constraints i specifikationen er overholdt. Hvis dette ikke er tilfældet, produceres en fejlmeddelelse som Jwig-programmøren igen skal håndtere som han finder passende. Han kan for eksempel vælge at vise fejlbeskeden til den formastelige bruger og lade ham udfylde formularen igen.

Som nævnt overlades det for tiden til Jwig-programmøren at håndtere valideringsfejl, hvilket ofte vil bestå at at gentage en løkke indtil valideringen er succesfuld. I en kommende version af Jwig forventes en konstruktion til automatisk håndtering af dette indført.

3.6.2 Et eksempel

Et eksempel kan sige mere end 1000 ord, og i den anledning vil en række illustrative farveillustrationer på de næste par sider illustrere hvorledes en bruger oplever en Jwig-service der betjener sig af PowerForms til validering af værdier i formularfelter. Hvis illustrationerne ikke kan tilfredsstille den kræsne læser, kan eksemplet prøves fra websiden <http://www.brics.dk/~amoeller/WWW/powerforms/pwfwjwig.html>.

Servicen der stilles til rådighed, giver brugeren mulighed for at bestille et antal gratis T-shirts, men for at undgå hamstring, er der en øvre grænse for hvor mange T-shirts man kan bestille.

Det vil føre for vidt at beskrive Jwig i stor detalje, men sløret kan da løftes for at dynamiske dokumenter opbygges ved brug af XML-skabeloner med navngivne *huller* som kan fyldes ud med tekststreng eller XML-skabeloner. Detaljer om hvordan udfyldning af huller og andre spændende ting foregår i forbindelse med Jwig kan findes på Jwig-hjemmesiden [<http://www.brics.dk/Jwig/>], mens det nedenfor blot vil blive demonstreret hvordan et hul fyldes ud.

I det nært forestående eksempel skal værdien i et felt begrænses til et bestemt interval, men i stedet for at mejsle den øvre grænse direkte ind i PowerForms-specifikationen, indsættes et hul der kan udfyldes med en værdi beregnet i JWIG. Et uddrag fra kildeteksten til eksemplet ser ud som følger:

```
XML format = [[ <powerforms>
  <constraint field="amount">
    <match> <interval low="1" high=[high]/> </match>
  </constraint>
</powerforms> ]];
```

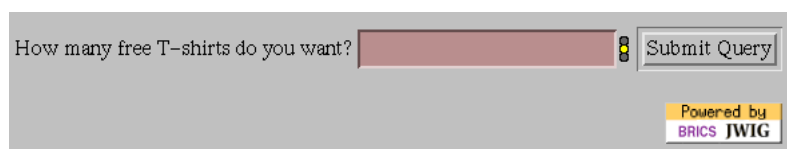
hvilket konstruerer et XML-dokument – i dette tilfælde et PowerForms-dokument – med et hul, „high“, som senere kan udfyldes med en passende værdi. Når en side skal vises i en JWIG-service, foregår det ved brug af nøgleordet „show“ på følgende måde:

```
show templateAsk<[msg=msg] powerforms format<[high=MAX];
```

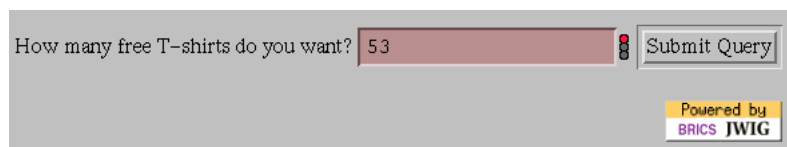
Besværgelsen format<[high=MAX] udfylder et hul, high, i dokumentet format med værdien af JWIG-variablen MAX og returnerer det opdaterede dokument. Tilsvarende udfyldes et hul, msg, i templateAsk. Nøgleordet powerforms angiver at argumentet til show skal powertransformeres inden det vises, og transformationen laves ud fra argumentet til powerforms.

Hele kildeteksten til JWIG-eksemplet kan findes i appendiks C, hvoraf det fremgår at den værdi der faktisk vil blive fyldt ind i hullet high er „5“, således at der maksimalt kan bestilles fem T-shirts.

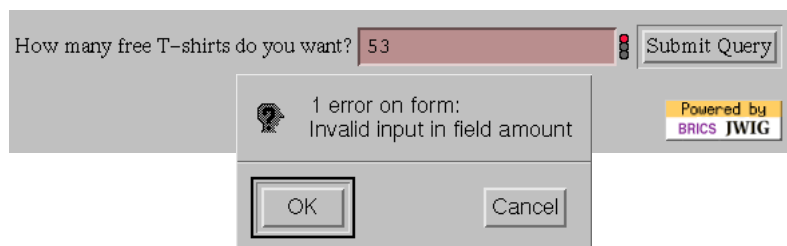
I det følgende vises forløbet af en brugers interaktion med servicen, og det første han præsenteres for, når han starter servicen er følgende formular:



Som det ses, gøres der opmærksom på at servicen er skrevet i JWIG og hvis man tidligere har prøvet PowerForms, vil man også nikke genkendende til det lille trafiklys til højre for tekstfeltet. Antag nu at brugeren er meget grådig og vil bestille 53 T-shirts på én gang, hvorfor han indtaster dette:



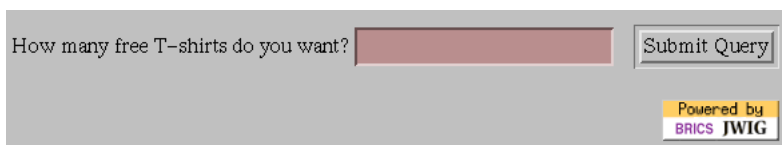
Trafiklyset indikerer nu at noget er helt galt og en erfaren PowerForms-bruger vil notere sig at han har indtastet noget han ikke må. En opmærksom begynder vil måske undre sig over hvad trafiklyset betyder, men uanset hvad vil han formentlig forsøge at submitte formularen ved at trykke på knappen. Kort efter vil han blive mødt af en dialog som vist her:



Efter at have læst meddelelsen og have sundet sig lidt, vil en fornuftig bruger nok erkende at det måske er lidt urimeligt at rekvirere 53 T-shirt, hvorfor nedsætter sine forventninger og beder om tre T-shirts. Med værdien „3“ i tekstfeltet vil et tryk på knappen resultere at følgende side dukker op i browseren:

You will receive 3 k001 T-shirts any day now...

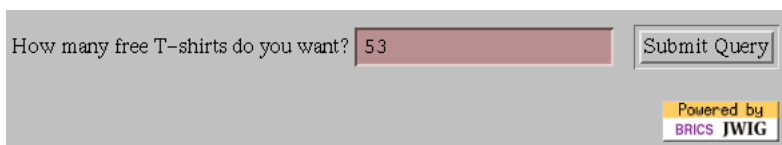
En stædig bruger der har sat sig i hovedet at han pinedød *vil* have sine 53 T-shirts, vil formentlig ikke slå sig til tåls med kun at kunne bestille fem af gangen. Han gransker derfor kildeteksten bag HTML-siden og opdager at valideringen af værdien i tekstfeltet varetages af noget JavaScript-kode. „Ha!“, tænker han, „de tror de kan sikre sig med noget scriptkode, men de bli’r klogere, gør de!“, hvorefter han resolut instruerer sin browser om *ikke* at udføre JavaScript på HTML-siden. Formularen i browseren skifter nu udseende til følgende, hvor det lille trafiklys ikke er at se:



How many free T-shirts do you want? Submit Query

Powered by BRICS JWIG

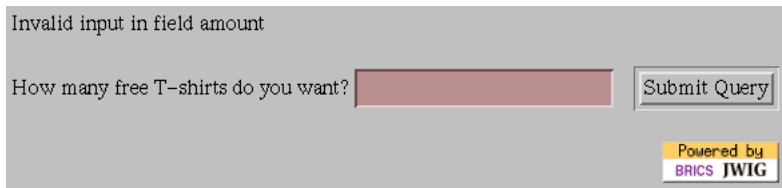
Den store ordre på 53 T-shirts afgives, og formularen synes at være glad og tilfreds.



How many free T-shirts do you want? 53 Submit Query

Powered by BRICS JWIG

„Glimrende, nu er julegaverne reddet!“ lykønsker den obsternasige bruger sig selv og submitter formularen. Kort efter dukker følgende op i browseren



Invalid input in field amount

How many free T-shirts do you want? Submit Query

Powered by BRICS JWIG

og den grådige bruger må tage til takke med en lang næse i stedet for 53 T-shirts ...

Ovenstående er et eksempel på hvordan PowerForms udover validering på klienten også kan sørge for validering på serveren, hvorved fejl og snyderi kan opdages. Desuden viser eksemplet vigtigheden af at validere formularværdier på serveren, fordi man ikke har en jordisk chance for at opdage hvilke krumspring en bruger foretager sig for at få lov til at submitte ulovlige værdier. Det turde også fremgå, at en bruger der lader sin browser køre JavaScript vil undgå unødigt nettrafik, idet „dumme“ fejl – eksempelvis sålfjel – vil blive fanget på klienten uden af serveren skal forstyrres.

3.7 Implementation

En PowerForms-specifikation består af en mængde af constraints og initialiseringer, og nogle ekstra besværgelser der skal få det hele til at hænge sammen. Mængden af constraints og initialiseringer definerer implicit en mængde af formularfelter der hver især er et par bestående af et formularnavn og et feltnavn: (*form, field*). Hvis et formularnavn

ikke er angivet, svarer det til en reference til den første formular på en HTML-side. Hvis et feltnavn ikke er angivet, svarer det til *alle* felter på den aktuelle formular.

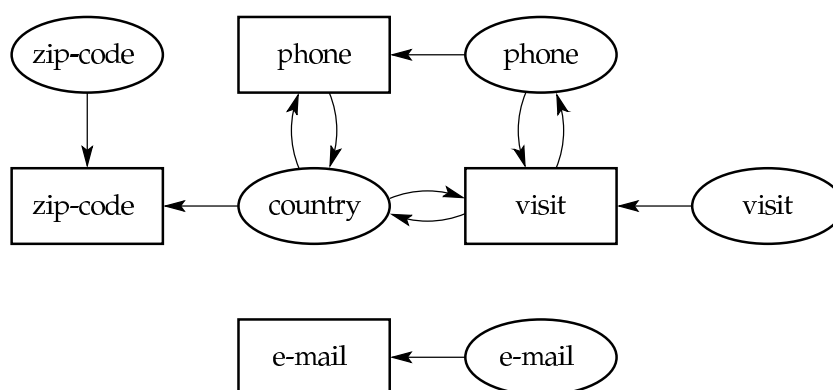
For at få hele PowerForms-meknikken til at fungere, er det nødvendigt at få styr på hvad der skal beregnes hvornår og hvad der ikke skal beregnes. Det interessante i forbindelse med PowerForms er for hvert formularfelt at vide hvilke constraints der afhænger af det og derfor skal (gen)evalueres når feltværdien ændres. Et constraint kan afhænge direkte af et felt hvis feltets værdi indgår i en kontrollerende betingelse i et if-udtryk, men der findes også indirekte afhængigheder, hvor et constraint afhænger af et felt der har et constraint der afhænger af et andet felt. Ud over constraints der afhænger af feltets værdi, skal constraints for selve feltet naturligvis også evalueres når værdien ændrer sig.

En garvet grafteoretiker ser grafproblemer overalt og derfor også her – og almindelige dataloger vil også kunne fornemme det ...

Det er nemlig oplagt at konstruere en todelt graf ud fra PowerForms-specifikationen for at afsløre afhængigheder – både direkte og indirekte – mellem constraints og felter. I den anledning konstrueres en orienteret graf G , og denne graf har en knude for hvert constraint og for hvert (implicit defineret) felt. For overskuelighedens skyld bruges betegnelsen „constraint“ i det følgende i betydningen „knude svarende til et constraint“, og noget tilsvarende gør sig gældende for udtrykket „felt“.

Kanterne i grafen går mellem constraints og felter, og hvert constraint har en kant til de felter det umiddelbart afhænger af, dvs. dem der indgår i if-udtryk i constraintet. Hvert felt har en kant til de constraints der gælder for feltet og til de constraints der direkte afhænger af feltet. Figur 19 viser den graf som PowerForms-specifikationen i det illustrative eksempel giver anledning til.

Det er i HTML muligt at definere mere end én formular i et dokument, og man kan i den anledning overveje om det skal være muligt at specificere afhængigheder mellem felter på forskellige formularer. Der er imidlertid kun én formular der kan submittes fra et HTML-dokument, og derfor er det rimeligt at kræve at felter på forskellige formularer ikke kan være indbyrdes afhængige, og det må betragtes som en designfejl af formularer hvis der er behov for at lade indholdet af en formular afhænge af værdier på en anden formular. I og med kun én formular submittes, vil al form for serverside-validering også umuliggøres hvis der er afhængigheder på tværs af formularer.



Figur 19: Afhængighedsgrafen for det illustrative eksempel, hvor constraints er firkantede og felter er elliptiske. For overskuelighedens skyld er formularnavne udeladt.

I første omgang repræsenterer G kun direkte afhængigheder, og for at finde de indirekte laves en form for transitiv lukning af grafen. Med „en form for“ menes at det ikke er den

traditionelle transitive lukning der beregnes, men derimod en variant, hvor det – populært sagt – kun er kanter ud fra felter der „kortslyttes“.

Hvis der i G findes en simpel sti af længde tre, så har stien pr. konstruktion af G formen $f \rightarrow c' \rightarrow f' \rightarrow c$, dvs. at den går fra et felt f til et constraint, videre til et felt og slutter i et constraint c . Det turde nu være klart at en sådan sti svarer til at constraintet c afhænger indirekte af feltet f .

Et enkelt skridt i beregning af den alternative lukning af G foregår ved at lave en kant mellem et felt f og et constraint c , hvis der findes en simpel sti på førnævnte form og der ikke allerede findes en kant mellem f og c . Algoritmen til lukning af G består i at tage dette skridt så længe betingelsen er opfyldt.

Når algoritmen terminerer, findes i G en kant fra alle felter til de constraints der skal genevalueres når feltets værdi ændres. Hvorvidt algoritmen terminerer eller ej, afhænger af om den klassiske algoritme „transitiv lukning“ terminerer, men da dette heldigvis er tilfældet følger at den omtalte variant er gyldig og korrekt.

Når et HTML-dokument skal transformeres, er det ikke nødvendigvis tilfældet at alle felter der implicit defineres af PowerForms-specifikationen findes i HTML-dokumentet. Derfor laves en projektion af grafen over de felter der rent faktisk findes, og ud fra denne grafprojektion laves så den faktiske transformation.

3.7.1 Evaluering

Når et felt på den ene eller anden måde har ændret værdi, skal alle berørte constraints genevalueres med den nye værdi taget i betragtning. Den ovenfor definerede graf angiver hvilke constraints der på ny skal evalueres når et felt ændrer værdi, idet der i grafen findes en kant fra knuden svarende til feltet, til netop knuderne der repræsenterer constraints der skal evalueres.

Man kan foranlediges til at tro at den hellige grav er velforvaret hvis man, når værdien af et felt ændres, evaluerer de berørte constraints ét efter ét i en mere eller mindre veldefineret rækkefølge. Dette er dog ikke tilfældet:

Som beskrevet i afsnit 3.4.1 kan evaluering af et constraint have som sideeffekt at mængden af værdier for et felt ændres. Lad C_1, C_2, \dots, C_n være mængden af constraints der skal genevalueres når værdien af et felt f ændres. Hvis disse constraints evalueres i den nævnte rækkefølge risikerer man at C_7 ændrer værdien af et felt som C_1 afhænger af hvorfor C_1 må genevalueres, hvilket videre kan have konsekvenser for de andre constraints i rækken, så de må genevalueres!

Spørgsmålet er nu, hvordan man løser det konstaterede problem, og svaret er faktisk meget ligetil: man evaluerer alle constraints i rækkefølge så længe værdier ændrer sig!

Dette kræver en nærmere forklaring som kommer her:

En HTML-formular definerer en mængde af felter som hver især har nul eller flere værdier, og antallet af værdier et felt kan have, ligger i et bestemt interval som beskrevet i afsnit 3.6.1. Enhver udfyldning af formularen bestemmer et antal værdier for hvert felt, og videre giver en udfyldning anledning til et element i et endeligt *lattice*, der beskriver mulige udfyldninger af formularen.

Ordningen i latticet – og dermed af udfyldninger af formularen – er en punktvis ordning efter hvor mange værdier hvert felt har. Det mindste element i latticet svarer til at alle felter i formularen har deres maksimale antal værdier, og det største element svarer til at alle felter har det mindst mulige antal værdier – udfyldninger er altså ordnet „omvendt“ efter hvor mange feltværdier der er.

Evaluering af C_1, C_2, \dots, C_n i én eller anden rækkefølge kaldes en *iteration*. Som beskrevet i afsnit 3.4.1 kan evaluering af et constraint som sideeffekt have at markeringer af knapper fjernes, hvilket betyder at en iteration er en *monoton funktion* på latticet defineret af en formular. Når man har en monoton funktion over et lattice, findes et *fikspunkt* for funktionen, og et fikspunkt kan findes ved gentagen iteration af den monotone funktion.

I PowerForms-regi svarer et fikspunkt i latticet til en udfyldning af formularen hvor alle værdier har stabiliseret sig således at endnu en iteration ikke vil ændre værdier i formularen. Derfor virker den „naive“ løsning med at evaluere constraints så længe værdier ændrer sig – og den vil tilmed terminere.

Som bekendt er fikspunktet i et lattice ikke nødvendigvis entydigt, men afhænger af den monotone funktion der itereres, og i den anledning fristes man til at spørge om det fundne fikspunkt afhænger af den rækkefølge constraints bliver evalueret i. Svaret på dette spørgsmål er: „ja, det fundne fikspunkt afhænger af evalueringsrækkefølgen!“, hvilket nu vil blive demonstreret:

Som datalog kan man ikke sige „lattice“ uden at komme til at tænke på isvafler, så intet er mere nærliggende end at demonstrere betydningen af evalueringsrækkefølgen ved brug af et webbaseret iskagehus, hvor man kan bestille „gammeldaws“ isvafler. I det følgende forventes læseren at kunne abstrahere fra det fjollede i at sælge isvafler over internettet – specielt i disse smalbåndstider ...

Som bekendt har Rasmus Klumps bedstefar et iskagehus [18], og på hans hjemmeside vil en simpel formular til bestilling af en isvaffel formentlig se ud som følger:

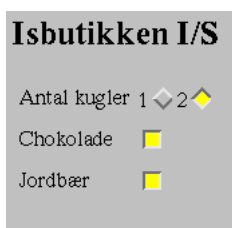
The image shows a web form for an ice cream shop. The title is "Isbutikken I/S". Below the title, there is a label "Antal kugler" followed by a radio button selected for "1" and another for "2". Below that, there are two checkboxes: "Chokolade" and "Jordbær", both of which are currently unchecked.

Man kan vælge én eller to kugler og derefter vælge hvilken smag hver enkelt kugle skal foregive at have. Hvis man kun vil have én kugle, skal det naturligvis kun være muligt at vælge én smag, hvilket svarer til at de to smagsfelter skal være gensidigt udelukkede. I PowerForms kan dette formuleres med to næsten identiske constraints for smagsfelterne. Constraint for det første smagsfelt ser ud som følger:

```
<constraint field="flavor1">
  <if>
    <equal field="num_scoops" value="1"/>
    <then>
      <if>
        <equal field="flavor2" value="yes"/>
        <then><empty/></then>
        <else><anything/></else>
      </if>
    </then>
    <else>
      <anything/>
    </else>
  </if>
</constraint>
```

og det andet constraint er magen til, bortset fra at „flavor1“ og „flavor2“ er byttet rundt.

Hvis man har spenderbukserne på og er lækkersulten, vil man gerne have to kugler i sin vaffel, og dette angives så i formularen:



Isbutikken I/S

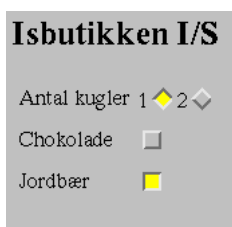
Antal kugler 1 2

Chokolade

Jordbær

En fintælling af ens kontantbeholdning kan nu afsløre at man ikke har råd til to kugler, og man derfor må angive at man kun vil have én kugle. Når antallet af kugler nedjusteres fra to til én, vil de to constraints for smagsfelter blive genevalueret og herved vil markeringen ud for „Jordbær“ og „Chokolade“ blive ændret. Spørgsmålet er bare: „Hvordan?“

Hvis en iteration først evaluerer constraintet for „jordbær“-feltet, vil markeringen ud for dette blive fjernet, og herefter vil „chokolade“-constraintet ikke forårsage yderligere forskydninger. Da feltværdier blev ændret under første iteration, laves endnu en iteration, men denne ændrer ikke nogen værdier, og følgende fikspunkt er fundet:



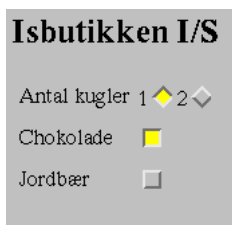
Isbutikken I/S

Antal kugler 1 2

Chokolade

Jordbær

Man kan også evaluere de to constraints i modsat rækkefølge, og så vil fikspunktet se ud som følger:



Isbutikken I/S

Antal kugler 1 2

Chokolade

Jordbær

Det fremgår tydeligt at de to fikspunkter ikke er ens, og dette viser at evalueringsrækkefølgen har betydning for hvilket fikspunkt der opnås. Når det er konstateret at det ikke er ligegyldigt i hvilken rækkefølge constraints bliver evalueret, er det på sin plads at specificere en veldefineret rækkefølge således at programmøren kan forholde sig til den.

Når der er afhængigheder mellem felter på en formular, vil det som regel være tilfældet at afhængigheder „peger bagud“, forstået derhen at et felt der afhænger af et andet felt, oftest kommer efter det felt der afhænger af. Derfor vil en iteration i PowerForms først evaluere involverede feltconstraints i den rækkefølge hvori felterne forekommer i HTML-dokumentet, hvorefter alle formularconstraints evalueres i den rækkefølge som de har i PowerForms-specifikationen. Hertil skal det bemærkes at det er det første kontrolelement for et felt der definerer feltets placering i et HTML-dokument.

3.7.2 Teknologier

PowerForms baserer sig på allerede eksisterende teknologier, hvilket medfører nogle begrænsninger på udskjelserne man kan foretage når det hele skal i sving. De følgende afsnit vil beskrive de forskellige teknologier PowerForms benytter sig af og oplyse om problemer i den anledning.

Et meget stort problem er at PowerForms skal fungere i browsere. Problemet består i at der findes mindst to forskellige browsere, og det virker som om det eneste programmørerne bag disse er enige om er at de ikke er enige om noget. Derfor er det afgørende at afprøve sin kode i et bredt udvalg af browsere, og det gælder både i forbindelse med PowerForms-genereret kode, men også for hjemmestrikket scriptkode.

HTML

Som nævnt har det været et mål at PowerForms skal kunne skrues ovenpå allerede eksisterende HTML-formularer, men som bekendt er anvendt HTML et meget flydende begreb, hvor folk gennem mange år har fået lov til at kalde næsten hvad som helst for HTML. Det giver nogle oplagte problemer med hensyn til parsning af input til PowerForms og har givet anledning til indførelse af et såkaldt *crap-mode*¹ som gør at PowerForms kan parse selv meget forkerte HTML-dokumenter og trylle dem om til mindre forkerte HTML-dokumenter.

I implementationen af PowerForms burde der ikke bruges tid og kræfter på krumspring for at kunne hitte rede i dokumenter der ikke er HTML, så en anden – og bedre – løsning vil være at bruge HTML Tidy til at parse et påstået HTML-dokument.

HTML Tidy [<http://www.w3.org/People/Raggett/tidy/>] er et program der kan rydde op i selv de værste såkaldte HTML-dokumenter og i stedet producere et korrekt HTML-dokument. Der findes en Java-implementation, JTidy [<http://lempinen.net/sami/jtidy/>], af HTML Tidy, og ved at bruge denne som værn mod omverdenen, vil det således være muligt at fastholde PowerForms i den naive tro at der kun findes korrekt HTML.

Den optimale løsning vil naturligvis være at alle mennesker i hele verden begynder at skrive korrekt HTML, men det ligger uden for dette speciales område og desværre nok nogle år ude i fremtiden at få dette ønske opfyldt ...

For ikke at skulle bekymre sig om hvad HTML er – og ikke er – holder PowerForms under parsning af HTML-input kun øje med ting der har relevans for PowerForms. Det betyder at formulardefinitioner og tilhørende definitioner af kontrolelementer samles op i passende strukturer, hvorimod den del af dokumentet der blot er „normalt“ HTML ikke tages i nærmere øjesyn, men blot opsamles så det kan spyttes ufordøjet ud når PowerForms har tygget på sit input. Dette gør at der kan hældes vilkårlig dårligt HTML ind i PowerForms, og at det så kommer lige så dårligt ud igen. Det er dog naturligvis tilfældet at PowerForms *ikke* gør HTML-koden mere forkert under transformationen – læsbarheden kan dog dale drastisk, fordi der indsættes en masse scriptkode og tilføjes attributter således at passende funktioner bliver kaldt når kontrolelementer ændrer værdi.

JavaScript

For at få den ønskede dynamik når brugeren udfylder en formular, benyttes den JavaScript-implementation der (forhåbentlig) er til rådighed i browseren. Da der findes mere end én forskellig browser, er her et område med glimrende muligheder for at jogge i spinaten.

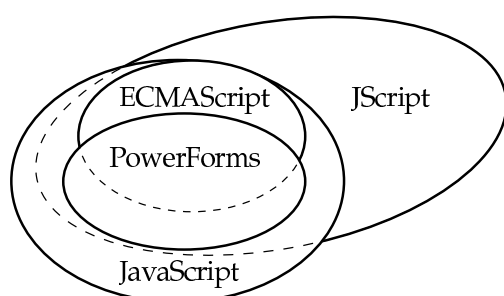
¹Tak til Mads Brøgger for betegnelsen „crap-mode“

W3C har udstukket retningslinier for hvordan det bør være muligt at tilgå og manipulere *DOM'en* [25] for det aktuelle dokument, men det er ikke alle browsere der overholder anbefalingerne. Man kan mene hvad man vil om W3C's DOM-anbefaling, men det at der findes en specifikation man kan forholde sig til, gør mange ting nemmere – hvis man ikke er den eneste der forholder sig til den ...

Én ting er hvordan man tilgår *DOM'en*, en anden er hvordan „almindeligt“ scriptkode fortolkes i browseren. Ecma har udgivet en specifikation af scriptsproget *ECMAScript* [17], der er en videreudvikling af JavaScript, som i sin tid blev opfundet af Netscape, og de fleste moderne browsere understøtter ECMAScript i henhold til specifikationen. Desværre har hver browser også sine egne udvidelser af ECMAScript, og hvis man ikke er påpasselig og ved hvad der er med i standarden, kan man hurtigt få skrevet noget script-kode der kører fint i én browser, men ikke kan køre i de andre.

Når man har konstateret at der er forskelle browsere imellem, vil et næste træk hen imod „perfekt“ scriptkode være at finde ud af hvad hver enkelt browser understøtter og hvad den ikke understøtter. Her erfarer man så at nogle browserproducenter holder kortene tættere til kroppen end andre, og uden at nævne navne kan det afsløres at nogle kun nødtigt udleverer sin dokumentation, og kun hvis man bruger en browser af fabrikatet „Microsoft Explorer“.

Det er forholdsvis nemt at holde sig indenfor den udstukne ECMAScript-specifikation [17], men det farlige er at det også er nemt at komme til „at træde udenfor“ og skrive noget kode der kun virker i én browser. Især når man vil lave ting der kræver at man roder med repræsentationen af HTML-dokumentet i browseren, er der rig mulighed for at dumme sig. Håndtag til at manipulere med selve dokumentet ligger nemlig uden for specifikationen af ECMAScript, og derfor er det nødvendigt med omhyggelige afprøvninger i diverse browsere før man kan vide sig sikker på at man har dækket de mest almindelige browsere og ikke benytter funktionalitet der kun findes i én browser.



Figur 20: Forskellige versioner af JavaScript. Som det ses snitter de forskellige versionen ikke trivielt, men trods alt heller ikke tomt ...

Figur 20 illustrerer den noget uafklarede situation i forbindelse med forskellige versioner af JavaScript. Når man vil skrive JavaScript-kode der fungerer i flest mulige browsere, består øvelsen i at holde sig indenfor snittet af de forskellige versioner af JavaScript som én's kode kan risikere at blive udsat for. Som det fremgår af figuren holder PowerForms sig i dydens smalle snit som det er defineret af browsere fra Netscape, Mozilla og Microsoft. Desværre er det meget muligt at der findes en browser „derude“ som har en lidt anden opfattelse af JavaScript og som derfor ikke kan køre den kode der genereres af PowerForms.

3.7.3 Kodegenerering

Når en HTML-side sammen med en tilhørende PowerForms-specifikation køres gennem PowerForms-oversætteren, er det en modificeret version af HTML-siden der kommer ud på den anden side. I dette afsnit vil strukturen af den genererede kode blive beskrevet.

Som nævnt tidligere forsøger PowerForms at løse sin opgave uden at skulle vide en hel masse om hvad der er lovligt HTML og hvad der ikke er. I den anledning er det rimeligt at kræve at PowerForms i det mindste ikke ødelægger noget for nogen der ved noget om HTML, dvs. at PowerForms ikke må transformere et lovligt HTML-dokument om til et ulovligt ét. Som beskrevet i afsnit 5.9 stiller PowerForms visse udvidelser af HTML til rådighed, og da disse ikke er en del af HTML, fjerner PowerForms alle spor af dem og indsætter i stedet noget korrekt HTML- og scriptkode der modsvarer udvidelserne.

Struktur

For at få sving i maskineriet genererer PowerForms én stor stump JavaScript-kode der indsættes i det transformerede HTML-dokument. Det første der gøres i den genererede kode, er at inkludere en række PowerForms-standardbiblioteker der kan håndtere forskellige generelle områder af PowerForms, blandt andet inkluderes et dfa-bibliotek der indeholder funktioner til at afgøre om en givet streng accepteres af en dfa.

Efter at alt generisk kode er inkluderet, genereres kode der er specifikt for det PowerForms- og HTML-dokument der indgår i den aktuelle transformation. Koden har følgende struktur:

Dfa-erklæringer For hvert anvendt regulært udtryk genereres en ECMAScript-version af den tilhørende minimale dfa.

Constraintfunktioner Hvert constraint giver anledning til én funktion der kan afgøre om constraintet er overholdt og som sideeffekt sørger for at tilhørende status-elementer opdateres på behørig vis ud fra resultatet af evalueringen. Hvis evalueringen af et constraint ikke er succesfuld returneres en liste af fejlbeskeder.

Formularfunktioner For hver formular på siden vil de nedenstående funktioner blive genereret hvis de er relevante for formularen.

Opdateringsfunktion En funktion der evaluerer alle constraints for selve formularen eller for felter på formularen.

Submitfunktion Hvis en formular har tilknyttet constraints eller indeholder felter der er underlagt constraints, genereres en funktion der kaldes når brugeren forsøger at submitte formularen. Funktionen undersøger om alle constraints for formularen og dens felter er overholdt, og hvis dette ikke er tilfældet vises en fejlbesked, hvori brugeren informeres om hvad der er galt og formularen bliver ikke submittet. Hvis alle constraints er overholdt, submittes formularen præcis som det foregår i „almindeligt“ HTML.

Nulstillingsfunktion Hvis en formular indeholder en reset-knap, genereres en funktion der nulstiller formularen ved at give alle felter deres initiale værdi og gen-evaluere alle tilhørende constraints.

Initialiseringsfunktion Endelig genereres én funktion til at initialisere hele PowerForms-mekanismen, hvilket blandt andet involverer at initialisere alle status-elementer og kalde opdateringsfunktionen hørende til hver formular.

Ud over at erklære de ovennævnte funktioner er det naturligvis nødvendigt at kalde dem på passende tidspunkter før det bliver interessant.

Dynamik

Den globale initialiseringsfunktion kaldes kun én gang, og det sker efter at alle formularer i HTML-dokumentet er blevet defineret hvilket er når hele dokumentet er indlæst og vises i browseren.

Når alle PowerForms-specifikke elementer i dokumentet er blevet initialiseret, skal den dynamiske del af PowerForms i sving, og i den anledning er der tre hændelser der er interessante:

- Brugeren ændrer værdien af et felt på en formular.
- Brugeren forsøger at submitte en formular.
- Brugeren nulstiller en formular.

PowerForms skal holde øje med hver af disse hændelser og sørge for at de rigtige funktioner bliver kaldt. Dette gøres ved at PowerForms tilføjer en række *eventhandlere* til forskellige elementer i HTML-dokumentet:

Alle kontrolelementer der har indflydelse på constraints, bliver udstyret med eventhåndlere således at de berørte constraints genevalueres når brugeren ændrer værdien i kontrolelementet. For at lave dynamisk validering af brugerens indtastninger sørges yderligere for at tekstfelter opdaterer constraints løbende mens brugeren skriver tekst i feltet.

Når brugeren vil submitte en formular, foregår det ved at han trykker på en submit-knap eller aktiverer et, af PowerForms introduceret, submit-element (afsnit 5.9.1). Derfor får alle disse elementer tilføjet en eventhandler der sørger for at formularens submitfunktion bliver kaldt. I særlige tilfælde kan en formular submittes ved at brugeren trykker på returtasten, og derfor instrueres en formular om at kalde sin submitfunktion hvis den forsøges submittet uden at et submit-element er blevet aktiveret.

Alle reset-knapper får tilføjet en eventhandler således at et tryk på knappen vil medføre at den til formularen hørende nulstillingsfunktion bliver kaldt.

Kapitel 4

PowerForms i den virkelige verden

Det er blevet påstået at PowerForms umiddelbart kan skrues ovenpå allerede eksisterende HTML-formularer, og for at underbygge denne påstand, indeholder dette kapitel en række eksempler på hvad PowerForms kan gøre for og ved formularer fra „den virkelige verden“.

I det følgende vil tre formularer, der er offentligt tilgængelige på webben, blive præsenteret, og disse tre formularer demonstrerer forskellige måder hvorpå validering for tiden foretages i webbaserede formularer, der rent faktisk bruges til noget. For hver formular er det på empirisk vis forsøgt afdækket hvilken form for validering der laves, hvorefter en PowerForms-specifikation af denne validering er blevet skrevet.

Det er ikke nogen hemmelighed at der findes meget forkert HTML i verden – specielt ikke når man har læst forrige kapitel – og på grund af dette har det været nødvendigt med nogle få tilrettelser af HTML-koden bag de formularer der skal have PowerForms trukket ned over sig. Rettelserne har udelukkende haft til formål at gøre det muligt at køre PowerForms på formularerne ved eksempelvis at tilføje id-attributter til formularer og navne til input-elementer. Hvis en formular i forvejen benytter validering ved brug af JavaScript, er dette slået fra, så PowerForms har frit spil til at lave sin validering.

For at kunne sammenligne indtryk af formularen henholdsvis med og uden PowerForms kan alle eksemplerne afprøves på siden <http://www.brics.dk/~ricky/powerforms/virkelighed/>.

4.1 Eksempel 1: Extensible Forms Description Language

Hvis man har sat sig for at afprøve XFDL i praksis, kan man hente en implementation til Microsoft Windows ved at åbne url'en <http://www.uwi.com/downloads/viewer/index.php> i sin browser og indtaste sin e-mail-adresse. Hvis den indtastede e-mail-adresse ikke er PureEdge bekendt, bliver man efterfølgende præsenteret for en formular som afbildet i figur 21, hvori der skal angives alverdens oplysninger om én selv og sin virksomheds gøren og laden.

The Leader in
Secure XML e-Forms

NEWS
PRODUCTS
SOLUTIONS
SERVICES
SUPPORT
PRACTICES
CUSTOMERS
PARTNERS
CORPORATE
RESOURCES
WHITE PAPERS
SAMPLE FORMS

ICS
Free
Trial
Software

PureEdge
POWERED
Affiliate
Member Site

ICS Viewer v.5.0 - Product Download

To download the ICS Viewer version 4.6, please complete and submit this *Contact Information Form*. Subsequent downloads will not require this step if you use the same email address each time.

Note: Fields with red labels are required and must be completed.

Full Name:

Title:

Department:

Organization:

Sector:

Address:

City: **State/Province:**

Zip/Postal Code:

Country:

Phone: **Fax:**

Email:

Note: You must provide a VALID e-mail address to download our products.

How did you hear about PureEdge.Com?

Name of website, publication, etc.:

What is the size of your Organization?

What is the size of the IT budget for which you are directly responsible?

Are you evaluating InternetForms for a current IT project?

If so, what is your timeframe for making IT purchases related to the project?

Does your Organization currently use electronic or Internet-based forms?

If so, what type of solution are you currently using?

Which Operating System do you currently use?

Yes, I would like to receive email containing information on PureEdge products and services.

[News](#) | [Solutions](#) | [Services](#) | [Customers](#) | [Partners](#) | [Resources](#) | [Support](#) | [Corporate](#) | [Practices](#)

[Home](#) | [Contact](#) | [Search](#)

Copyright © 1996 - 2001 PureEdge Software, Inc. All rights reserved.

Figur 21: En formular der skal udfyldes korrekt hvis man vil gøre sig forhåbninger om at hente en XFDL-processor.

Øverst i formularen bekendtgøres det at „Fields with red labels are required and must be completed.“ Kontrollementet hørende til en sådant felt er enten et tekstfelt der i så fald skal udfyldes ellers er det en liste, hvori der skal vælges en indgang forskellig fra den første.

Hvis man submitter formularen uden at have taget stilling til alle felter med et påkrævet værdi, dukker formularen op igen i let modificeret udgave: Alle felter der ikke er korrekt udfyldt har stadig røde labels, mens felter der er korrekt udfyldt har fået farvet deres label sort.

Valideringen der laves, stikker øjensynligt ikke dybere end at afgøre om der er indtastet en streng der ikke kun består af mellemrumstegn i tekstfelter eller om en indgang forskellig fra den første i en liste er valgt. Der er dog enkelte undtagelser: i e-mail-feltet kræves det at der indtastes en lovlig e-mail-adresse og i forbindelse med et par ja-nej-spørgsmål antyder teksten „if so, what ...“ at et bekræftende svar skal uddybes, hvilket dog ikke synes at være tilfældet.

Al validering foregår på serveren, som i tilfælde af fejl samtidig sørger for at opdatere formularen således at det kun er manglende værdier der er markeret med røde labels. Ud over at undersøge om der er indtastet eller valgt en værdi, er det kun værdien af e-mail-adressen der udsættes for yderligere validering så det afgøres om der faktisk er tale om en lovlig adresse.

HTML-koden bag formularen er pæn og velstruktureret og kan umiddelbart hældes gennem PowerForms og desuden falder der meget belejligt at PureEdge har nummeret indgange i lister startende fra 0, idet kravet om at der skal vælges en indgang, så kan formuleres som at værdien skal være forskellig fra „0“.

4.1.1 Constraints

Da valideringen af formularen er forholdsvis simpel, er der ingen grund til at redegøre for alle constraints der skal sammenflikkes, så i stedet vil nogle højdepunkter fra en passende PowerForms-specifikation blive beskrevet, og hele specifikationen kan ses i appendiks B.1.

Tekstfelter der skal udfyldes, skal have en „synlig værdi“, hvilket betyder at værdien ikke udelukkende må bestå af mellemrumstegn, og i den anledning defineres følgende regulære udtryk:

```
<regexp id="non-empty">
  <regexp pattern=".*[\t]+.*"/>
</regexp>
```

For alle tekstfelter der skal udfyldes, skal der så blot laves et constraint der kræver at værdien passer til dette regulære udtryk, og for at øge brugervenligheden, bør der også gives en fornuftig fejlmeddelelse. For feltet til indtastning af fornavn ser det således ud:

```
<constraint field="r_firstname">
  <match error="You must enter your first name">
    <regexp idref="non-empty"/>
  </match>
</constraint>
```

For at afgøre om der i en liste er valgt en indgang forskellig fra den første skal det, som nævnt ovenfor, blot afgøres om den valgte værdi er forskellig fra „0“, hvilket kan gøres med følgende regulære udtryk:

```
<regexp id="required">
  <complement>
    <const value="0"/>
  </complement>
</regexp>
```

Hvis dette regulære udtryk trækkes ned over lister med et feltconstraint, vil listerne som bekendt få fjernet alle indgange hvis værdi ikke passer til udtrykket, men det er ikke det der ønskes. I stedet laves derfor et formularconstraint der kræver at den valgte værdi i listen passer til udtrykket, og igen garanteres med en passende fejlbesked:

```
<constraint>
  <match field="r_title" error="Please select your title">
    <regexp idref="required"/>
  </match>
</constraint>
```

Som antyd det tidligere kan bemærkningen „if so, what ...“ foranledige den vakse bruger til at tro at et felt skal udfyldes hvis man har svaret bekræftende på et indledende spørgsmål, men PureEdge kræver ikke dette. Det bør dog ikke holde en PowerForms-skrivent tilbage fra at formulere følgende simple constraint:

```
<constraint>
  <if>
    <equal field="r_useforms" value="1"/>
    <then>
      <match field="r_currentforms" error="Please state your current form solution">
        <regexp idref="non-empty"/>
      </match>
    </then>
    <else>
      <ignore field="r_currentforms"/>
    </else>
  </if>
</constraint>
```

Effekten vil være at hvis man har bekræftet at man for tiden bruger webbaserede formulærer så skal man fortælle hvilken slags man bruger. Der er endnu et af disse „afhængigheds-spørgsmål“ på formularen, og dette giver naturligvis anledning til et tilsvarende constraint.

4.1.2 Resultat

Det er oplagt at det vil blive nemmere for brugeren at udfylde formularen hvis den ud over validering på serveren udstyres med klientsidevalidering ved brug af PowerForms. Det er ikke alle brugere der er lige tålmodige og giver sig tid til at læse instruktioner for hvordan formularen skal udfyldes. Selv om man har læst instruktionen og har udfyldt alle tekstfelter, er det nemt at overse en liste som man mangler at vælge en indgang i, og den slags „sjuskefejl“ er netop et vægtigt argument for at der skal laves validering på klienten inden serveren laver den endegyldige validering.

4.2 Eksempel 2: Java Developer Connection

For at melde sig under fanerne i Suns *Java Developer Connection* skal man udfylde formularen der findes som <http://developer.java.sun.com/servlet/RegistrationServlet>. Den fine illustration i figur 22 viser et aftryk af bemeldte formular.

The image shows a registration form for the Java Developer Connection. At the top left is the Java logo. The header reads "JAVA DEVELOPER CONNECTION" and "java.sun.com". The main heading is "Register and Get the Info You Need Today!". Below this, there are instructions and a link to learn more about the benefits of joining the JDC. The form fields include: User ID, Password (with fields for "Once" and "Twice"), Name (with fields for "First" and "Last"), Email, Country (a dropdown menu showing "UNITED STATES", "GERMANY", "CANADA", and "UNITED KINGDOM"), and City (with a "State/Province" field). There are several "required" labels in red. Below the form fields are two checkboxes: "I am interested in participating in occasional online surveys and research projects..." and "I am interested in occasionally receiving information from Sun and Sun Partners." A "Privacy Policy" link is provided. The section "The JDC Newsletters and Publications" contains a "NOTE" and a list of newsletters with checkboxes: "I want it all!", "Send me newsletters in HTML whenever possible.", "JDC Newsletter", "Java(TM) Technology Fundamentals Newsletter", "JDC Regional Bulletin", "Wireless Developer Tech Tips", "JDC Tech Tips", "Wireless Developer Bulletin", "Wireless Developer Newsletter", and "Java Learning Center". A "SUBMIT" button is at the bottom.

Figur 22: En indmeldelsesformular til Java Developer Connection.

For at melde sig skal man vælge brugernavn og et lösen, indtaste sit navn og sin e-mail-adresse og afsløre lidt om hvor i verden man bor. Der er desuden mulighed for at abonnere på en sværm af elektroniske nyhedsbreve og bulletiner som vil blive tilsendt pr. e-mail.

Hvis det første man gør når formularen dukker op i ens browser, er at submitte den får man formularen smidt tilbage i hovedet med følgende inskription tilføjet:

The following fields need to be corrected:

- **UserId is required**
- **Password1 is required**
- **Password2 is required**
- **FirstName is required**
- **LastName is required**
- **Please select a country.**

Dette afslører at validering foregår på serveren, og desuden afsløres lidt af hvad valideringen består i. På formularen findes to felter til indtastning af et lösen, og yderligere eksperimenter bekræfter mistanken om at de to løsener kræves at være ens.

Ved at granske url'en til formularen får man mistanke om at den stammer fra en web-service der er implementeret som en *servlet* [<http://java.sun.com/products/servlet>], hvilket er et Java-program der kører på serveren og håndterer interaktion med brugeren. Eftersom Sun har haft en stor finger med i udviklingen af netop servlets, er mistanken nok begrundet. Da PowerForms er implementeret i Java, er det i en *servlet* umiddelbart muligt at bruge PowerForms til validering på både klient og server. Alt der skal til, er en passende PowerForms-specifikation, og i stedet for at sende et dynamisk opbygget HTML-dokument til klienten direkte, sendes det først gennem PowerForms, hvorefter det transformerede dokument sendes til klienten. Når klienten returnerer værdierne fra den udfyldte formular tjekkes de på serveren, og hvis valideringen ikke er succesfuld, må dokumentet sendes til klienten igen så brugeren kan rette fejlene.

Det der skal foregå, er præcis som i JMWIG [afsnit 3.6.1] – bortset fra at JMWIG automatisk transformerer og validerer, mens man i en *servlet* manuelt skal gøre det. Et eksempel på hvordan PowerForms bruges i en webservice der er baseret på Java, kan findes på PowerForms-hjemmesiden [<http://www.brics.dk/~ricky/powerforms/api/example.html>].

4.2.1 Constraints

Appendiks B.3 afslører et forslag til en PowerForms-specifikation der vil stå godt til formularen, og i stedet for at gennemgå hele specifikationen her, vil der blot blive bragt nogle medrivende uddrag:

Eventuelle krav til værdierne af „UserId“ og „Password1“ ud over at de skal være ikke-tomme, kendes ikke, så der kræves blot at der skrives noget i begge felter. Det kan ikke overraske hvordan dette klares i praksis, og derfor vises i stedet et constraint som kræver at de indtastede løsener er ens:

```
<constraint id="matching-passwords">
  <equal error="Password 1 and Password 2 do not match. Try again.">
    <field name="Password1"/>
    <field name="Password2"/>
  </equal>
</constraint>
```

Dette constraint kan eventuelt suppleres med et status-element som illustreret i figur 25 i afsnit 5.9.2.

I forbindelse med muligheden for at tegne abonnement på elektroniske nyhedsbreve er der en række interessante ideer til constraints: Man kan vælge at abonnere på hele sortimentet, og i så fald bør det ikke være muligt at vælge de enkelte udgivelser for sig, hvilket giver anledning til at constraint på formen

```

<constraint field="ed_892">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

```

for hvert enkelt felt svarende til en udgivelse.

Det er i formularen muligt at angive at man gerne vil modtage nyheder i HTML, men det bør naturligvis ikke være muligt at bede om få tilsendt HTML-versioner hvis man ikke abonnerer på noget, hvilket i PowerForms formuleres som:

```

<constraint field="html_pref">
  <if>
    <or>
      <match field="wantAllSubs"><anything/></match>
      <match field="ed_892"><anything/></match>
      <match field="ed_897"><anything/></match>
      <match field="ed_896"><anything/></match>
      <match field="ed_895"><anything/></match>
      <match field="ed_894"><anything/></match>
      <match field="ed_893"><anything/></match>
      <match field="jdc_1092"><anything/></match>
      <match field="jdc_1093"><anything/></match>
    </or>
    <then><anything/></then>
    <else><empty/></else>
  </if>
</constraint>

```

4.2.2 Resultat

Bortset fra at det ikke klæder en formular at være overdrysset med ni røde labels med teksten „(required)“ og serveren skal besværes med at lave den forholdsvis simple validering der er på tale, så er formularen ganske nem at gå til. Til gengæld virker det meget lidt elegant at en fejlbesked fra et mislykket forsøg på at udfylde formularen ristes ind i toppen af formularen – en lille listig fejlmeddelelse fra PowerForms vil nok være at foretrække.

Som eksemplificeret ovenfor er der en række sammenhænge mellem knapper til at vælge dette og hint i forbindelse med abonnementer på nyhedsbreve og lignende, som ganske nemt og elegant lader sig formulere i PowerForms, men som det vil være besværligt at implementere manuelt.

4.3 Eksempel 3: Barnets bog

Hvis man har været udsat for familieførøgelse, kan man fra <http://www.sundhed.dk/servlet/cbfrontpage> oprette *barnets bog*, hvori der er mulighed for at holde styr på en række oplysninger om den lille nye. Under „personlige data“ kan formularen i figur 23 udfyldes med alverdens oplysninger om blandt andet barnets mål og vægt, og det er oplagt at lave en form for validering af sådanne værdier.

The screenshot shows a web form titled "Barnets bog" with a sub-header "Rediger". Below this, it says "Indtast ændringer i de personlige data her:". The form contains several input fields, some of which are pre-filled with "PowerForms". The fields are: Fornavn (PowerForms), Efternavn, Køn (radio buttons for Dreng and Pige), Fødselsdato (02/02/02), Fødselstidspunkt (tt:mm), Fødselssted, Fødselsvægt i kg, Fødselshøjde i cm, Moders navn, Faders navn, Søskende, Læge, Læge tlf., Sundhedsplejerske, Sundhedsplejerske tlf., Dagsinstitution, and Dagsinstitution tlf. There is also a section for "Fødsels føreløb" with a large text area. A "Gem" button is at the bottom right. On the right side, there is a sidebar with a navigation menu: 1 Oversigt, 2 Personlige data (selected), 3 Milepæle, 4 Bygdomme, 5 Vaccinationer, 6 Vækstkurve hovedindlæg, 7 Vækstkurve foto, 8 Vækstkurve vægt.

Figur 23: En formular til redigering af personlige oplysninger om et lille barn.

Det er ikke nødvendigt at udfylde alle felter, men hvis man udfylder et felt, kan der være visse krav til værdien. Validering foregår på klienten ved brug af en mængde håndskrevne JavaScript-funktioner. Eksempelvis skal et fødselstidspunkt angives på formen *tt.mm*, altså med timer og minutter adskilt af punktum og to cifre efter punktummet, og der findes så en JavaScript-funktion der kan afgøre om det angivne fødselstidspunkt opfylder dette krav.

Ved at granske JavaScript-koden afsløres det at programmøren faktisk betjener sig af de „regulære“ udtryk der findes i JavaScript, men han er tydeligvis ikke klar over det fulde potentiale i regulære udtryk i forbindelse med validering. Faktisk bruges det regulære udtryk kun til at dele en streng op i „det før punktummet“ og „det efter punktummet“.

For at spare på kræfterne benyttes den samme funktion til at tjekke værdien af både fødselsvægt i kg og fødselshøjde i cm, hvilket har det lidt bizarre resultat at man kan angive vægten af spædbarnet til 54 kg og dets højde til 4,5 cm. Dette er nogle lidt usædvanlige værdier hvis der er tale om et ungt individ af racen *homo sapiens*, men det fremgår dog ikke tydeligt af formularen hvorvidt det er et krav at det tabellagte individ er af denne race.

Ud over oplysninger om mål og vægt er der i formularen også mulighed for at notere telefonnumre på læge, sundhedsplejerske og en eventuel daginstitution, og kravene til sådanne værdier er guf for den ivrige PowerForms-skribent. Et bud på en række constraints

kan ses i appendiks B.2, og i næste afsnit vil en række uddrag herfra blive lagt for dagen.

Ud over validering på klienten synes der også at foregå lidt på serveren, idet det ikke er muligt at snige værdien „syv“ ind i fødselsvægten ved at slå udførelse af JavaScript fra på klienten. Faktisk synes det ikke at være muligt at opdatere værdier uden brug af JavaScript, idet visse værdier bliver præformateret på klienten for at kunne blive accepteret på serveren – det er sgu ikke særlig smart!

I lighed med det forrige eksempel, hvor der formentlig gemmer sig en servlet bag servicen, er der også noget der tyder på at dette er tilfældet med Barnets Bog.

4.3.1 Constraints

Som nævnt skal der kun stilles krav til værdien af et felt hvis brugeren faktisk har indtastet noget i feltet. Derfor synes det rimeligt at der kun vises statusikoner, hvis værdien i feltet (endnu) ikke er lovlig og for eksemplets skyld kan dette signaleres med en rød klat:

```
<status blank="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
na="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
green="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
yellow="http://www.brics.dk/~ricky/powerforms/misc/red-dot.gif"
red="http://www.brics.dk/~ricky/powerforms/misc/red-dot.gif"/>
```

Fødselstidspunktet skal skrives på formen *tt.mm*, og underligt nok indeholder tekstfeltet til indtastning af tidspunktet fra starten teksten „tt.mm“, men der stilles ikke noget krav om at man skal slette denne tekst og skrive et rigtigt tidspunkt i stedet. Da PowerForms vil hjælpe den opmærksomme bruger til at skrive tidspunktet på den korrekte form er der ingen grund til at lade feltet indeholde „tt.mm“, men for at være så lig den oprindelige validering som muligt tillades værdien „tt.mm“ således at et constraint for tidspunktsfeltet kan se ud som følger:

```
<constraint field="TIMEOFBIRTH">
  <match error="Du har indtastet et ugyldigt tidspunkt.
    Tidspunktet skal have følgende format tt.mm - forsøg venligst igen!">
    <union>
      <const value="tt.mm"/>
      <optional>
        <concat>
          <union>
            <interval low="0" high="23"/>
            <interval low="0" high="23" width="2"/>
          </union>
          <const value="."/>
          <interval low="0" high="59" width="2"/>
        </concat>
      </optional>
    </union>
  </match>
</constraint>
```

Bemærk brugen af width-attributten for at kræve at minuttal mindre end ti skrives med et foranstillet nul. I øvrigt kan afsnit 5.4 med fordel konsulteres for at forstå hvorfor det regulære udtryk ser ud som det gør.

Efter rådføring med en forholdsvis nybagt mor, er det konstateret at en fødselsvægt rimeligvis kan antages at ligge mellem to og ti kg med en nøjagtighed på ti gram. Vægten i kg skal altså angives som et decimaltal mellem to og ti med maksimalt to decimaler efter kommaet:

```

<constraint field="WEIGHTATBIRTH">
  <match error="Fødselsvægt skal angives som et tal med maks 2 decimaler
    - forsøg venligst igen.">
    <optional>
      <concat>
        <interval low="2" high="9"/>
        <optional>
          <concat>
            <const value=","/>
            <union>
              <interval low="0" high="99"/>
              <interval low="0" high="99" width="2"/>
            </union>
          </concat>
        </optional>
      </concat>
    </optional>
  </match>
</constraint>

```

Yderligere forespørgsler hos den nybagte mor har ledt til at fødselshøjden – som nok snarere er en længde da barnet oftest ligger ned – kan antages at ligge et sted mellem 20 og 70 cm. Nøjagtigheden i målingen er oplyst til at være én cm, og det er en smal sag at formulere dette i et PowerForms-constraint:

```

<constraint field="LENGTHATBIRTH">
  <match error="Fødselshøjde skal angives som et tal, venligst prøv igen.">
    <optional>
      <interval low="20" high="70"/>
    </optional>
  </match>
</constraint>

```

Det skal bemærkes at den ordvendte omstilling i fejlbeskeden er kopieret direkte fra den oprindelige formular.

Til sidst bør der rimeligvis knyttes constraints til felterne hvori der skal indtastes diverse telefonnumre, men da der er ikke banebrydende nyt ved disse constraints, henvises den nysgerrige læser til appendiks B.2, hvor alle detaljer kan besigtiges.

4.3.2 Resultat

Felterne til at angive fødselstidspunkt, -vægt og -højde skal indeholde værdier det er mere eller mindre trivielt at opskrive regulære udtryk for, men som det er lettere bøvlet at skrive JavaScript-funktioner der kan afgøre lovligheden af. Programmøren har derfor valgt at springe over hvor gærdet er lavest og benytter den samme funktion til at validere to forskellige felter, til trods for at deres værdier rimeligvis ikke må indeholde værdier af samme numeriske størrelse. Som demonstreret er det ved brug af PowerForms en smal sag at formulere de krav som JavaScript-funktionen tjekker, og som bonusgevinst kan det ved samme lejlighed nemt afgøres om de indtastede værdier er realistiske.

Kapitel 5

PowerForms-manual

I dette kapitel vil der blive gået i detaljer med hvordan et PowerForms-dokument opbygges, og de forskellige PowerForms-elementer og deres semantik vil blive beskrevet.

Grammatikken for PowerForms kan beskues i appendiks D, og en DSD [20] for lovlige PowerForms-dokumenter kan findes på <http://www.brics.dk/~ricky/powerforms/powerforms.dsd>.

5.1 Overblik

En brug af PowerForms involverer to dele, nemlig et PowerForms-dokument og et HTML-dokument. HTML-dokumentet er et næsten normalt HTML-dokumentet, hvor „næsten“ hentyder til at PowerForms introducerer lidt ekstra krydderi på HTML, så der er mulighed for lidt mere avancerede sider end med almindeligt HTML [afsnit 5.9].

Et PowerForms-dokument (eller en *PowerForms-specifikation*) beskriver en mængde constraints som udtrykker hvad der skal gælde for formularerne i et HTML-dokument når det til sin tid er blevet *powertransformeret*. Et PowerForms-dokument er et XML-dokument [11], hvori rodelementet – meget passende – har navnet `powerforms` og inde i dette element kan man angive constraints. Ud over elementer til definition af constraints er der visse andre elementer der kan angives i et PowerForms-dokument, og disse elementer styrer blandt andet hvordan et powertransformeret dokument tager sig ud i en browser.

For at skabe lidt overblik over strukturen af et et PowerForms-dokument bringes nedenfor en kort beskrivelse af hvilke elementer der må forekomme på øverste niveau i dokumentet:

- **regexp** (afsnit 5.4)
Global erklæring af et regulært udtryk der senere kan bruges i constraints.
- **constraint** (afsnit 5.2)
Erklæring af et felt- eller formularconstraint.
- **status** (afsnit 5.6)
Brugerkonfiguration af status-elementer som PowerForms automatisk indsætter.
- **init** (afsnit 5.5)
Initialisering af feltværdier på en formular.

- **autocomplete** (afsnit 5.7)
Muliggør automatisk fuldførelse af værdier i tekstfelter med et tilknyttet feltconstraint.
- **option** (afsnit 5.8)
Håndtag til at ændre standardindstillinger i PowerForms.

Nedenfor vil de forskellige PowerForms-elementer blive beskrevet, og på samme måde som i appendiks D angiver „*“ nul eller flere forekomster og „+“ angiver én eller flere forekomster. Hvis en attribut er omklamret af „[“ og „]“, betyder det at det er valgfrit at angive den.

5.2 Constraints

Som beskrevet i afsnit 3 findes der to typer constraints i PowerForms, og de defineres begge med et constraint-element som beskrevet nedenfor.

`<constraint [form=form] [id=id]/>`

Definerer et formularconstraint. Hvis form-attributten ikke er angivet gælder constraintet for den første formular i et HTML-dokument med mindre denne har en id-attribut. Ellers gælder constraintet for en formular hvis id-attribut er lig *form*.

`<constraint [form=form] field=field/>`

Definerer et feltconstraint. Attributten form fungerer som beskrevet ovenfor, og constraintet vil gælde for felter med navn *field* på den udpegede formular.

Ud over de viste attributter har constraint-elementet også en error- og en warning-attribut der kan anvendes til at styre hvilke fejl- og advarselsbeskeder der gives.

Som beskrevet i afsnit 3.7 er afhængigheder mellem felter på forskellige formularer ikke tilladt, og derfor vil alle referencer til felter i et constraint pege på felter i den formular som constraintet gælder for.

Hvis et constraint ikke er overholdt på et tidspunkt hvor dette kræves, vil en passende besked blive vist til brugeren, og indholdet af denne besked kan angives ved brug af netop error- eller en warning-attributten. Hvis constraintet er „måske overholdt“ og der er angivet en advarselsbesked, vil denne blive vist og ellers vil fejlbeskeden blive vist. I tilfælde af at der skal gives en advarselsbesked, men en sådan ikke er angivet, vil en eventuelt angivet fejlbesked blive vist i stedet. Hvis et constraint ikke specificerer beskeder, vil brugeren blive præsenteret for passende standardbeskeder i tilfælde af fejl.

Alle udtryk beskrevet i afsnit 5.3 har tilsvarende error- og en warning-attributter. Når et udtryk resulterer i en fejl vil den „nærmeste“ besked der findes ved at gå fra udtrykket op igennem constraint-træet indtil en besked er fundet, blive vist. For overskuelighedens skyld er error- og warning-attributter udeladt i nedenstående oversigt over de udtryk der findes i PowerForms.

`<if> expression <then> constraint </then> <else> constraint </else> </if>`

Udtrykket *expression* evalueres, og hvis dets boolske værdi er sand, evalueres constraintet i then. Ellers evalueres det andet constraint.

`<ignore/>`

Dette constraint signalerer at værdien af feltet det gælder for, ikke er relevant og derfor kan være „hvad som helst“. Hvis der er knyttet et status-element med type css til feltet, vil det gøres inaktivt (hvis browseren understøtter dette).

5.3 Udtryk

Nedenfor er en kort beskrivelse af semantikken for de udtryk der findes i PowerForms.

Som det fremgår af afsnit 3.3 vil udtrykkene så vidt muligt, altså når det giver mening og kan lade sig gøre, blive evalueret i det beskrevne til én af de tre beskrevne tilstande.

De fleste udtryk har en field-attribut, men hvis udtrykket forekommer i et feltconstraint må denne ikke angives med mindre andet udtrykkeligt er nævnt. Når et udtryk bruges i et feltconstraint svarer det til at udtrykkets field-attribut refererer til feltet som constraintet gælder for.

Som et levn fra gamle dage er det tilladt at bruge et regulært udtryk som et udtryk, hvilket svarer til at pakke det regulære udtryk ind i et match-element. Det anbefales altid at bruge match-elementet for at øge læsbarheden og så der kan gives bedre fejl- og advarselsbeske-der.

`<count field=field number=number />`

Afgør om feltet *field* har præcis *number* værdier.

`<count field=field min=min max=max />`

Afgør om feltet *field* har mindst *min* og højst *max* værdier.

`<equal field=field [value=value] />`

Afgør om feltet *field* har mindst én værdi der er lig med *value*. Hvis udtrykket bruges i et feltconstraint og field-attributter er til stede afgøres lighed mellem feltet constraintet gælder for og det angivne felt som beskrevet nedenfor.

`<equal> field+ </equal>`

Hvis equal-elementet bruges i et formularconstraint eller som boolsk udtryk skal der angives præcist to field-elementer. I et feltconstraint må der kun angives ét field-element, idet feltet som constraintet gælder for fungerer som det andet. Udtrykket afgør om de to angivne felter har mindst én værdi til fælles.

`<less-than field=field [value=value] [type=type] />`

Afgør om feltet *field* har mindst én værdi der er mindre end *value*. Værdien af type-attributten angiver hvordan værdierne af *field* skal fortolkes. Lovlige typer er string og integer. Hvis udtrykket bruges i et feltconstraint og field-attributten er til stede afgøres det om det felt som constraintet gælder for er mindre end det angivne felt på.

`<less-than [type=type]> field field+ </less-than>`

Hvis less-than-elementet bruges i et feltconstraint, men *ikke* som boolsk udtryk, må der kun angives ét field-element og det afgøres så om det felt constraintet gælder for er mindre end feltet angivet i field-elementet. Ellers skal der angives to field-elementer og det afgøres så om feltet angivet i det første field-element har mindst én værdi der er mindre end en værdi af feltet angivet af det andet field-element. Attributten type fungerer som beskrevet ovenfor.

`<match field=field> regexp </match>`

Afgør om mindst én værdi af feltet *field* er indeholdt i sproget defineret af *regexp*.

`<not> expression* </not>`

Udtrykkene i *expression* evalueres som boolske udtryk ét efter ét og hvis ét eller flere af dem evaluerer til sand er værdien falsk. Ellers er værdien sand.

Må kun bruges som boolsk udtryk.

`<and> expression* </and>`

Udtrykkene i *expression* evalueres som boolske udtryk ét efter ét, og hvis ét eller flere af dem evaluerer til falsk, er værdien falsk. Ellers er værdien sand.

Må kun bruges som boolsk udtryk.

`<or> expression* </or>`

Udtrykkene i *expression* evalueres som boolske udtryk ét efter ét, og hvis ét eller flere af dem evaluerer til sand, er værdien sand. Ellers er værdien falsk.

Må kun bruges som boolsk udtryk.

5.4 Regulære udtryk

I kraft af at den aktuelle implementation af PowerForms er skrevet i programmeringssproget Java, definerer de regulære udtryk sprog bestående af strenge over unicode-alfabetet [<http://www.unicode.org>]. Følgende konstruktioner er til rådighed for opbygning af regulære udtryk.

`<empty/>`

Definerer det tomme sprog. Bemærk at dette *ikke* er sproget bestående af den tomme streng.

`<anychar/>`

Definerer sproget bestående af alle strenge af længde én.

`<anything/>`

Definerer sproget bestående af alle strenge.

`<const value=value/>`

Definerer sproget bestående af strengen *value*.

`<charset value=value/>`

Definerer sproget bestående af alle strenge af længde én der består af et tegn fra *value*.

`<charrange low=low high=high/>`

Definerer sproget bestående af alle strenge af længde én der leksikografisk ligger mellem *low* og *high*, begge inklusive. Værdierne af *low* og *high* skal begge være af længde én.

`<interval low=low high=high [width=width] [radix=radix]/>`

Definerer sproget bestående af alle strengrepræsentationer af tal mellem *low* og *high*, begge inklusive. Som standard opfattes *low* og *high* som tal i 10-talssystemet, men dette kan ændres ved at angive en anden base som værdi af *radix*-attributten. Cifrene der benyttes i strengrepræsentationerne er

0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w x y z

hvor *a* svarer til 10, *b* til 11 og så videre ...

Hvis *width* er angivet vil strengene få foranstillet nuller („0“) så de er af længde (mindst) *width* med et eventuelt foranstillet ‘-’ fraregnet.

Eksempel: `<interval low='100' high='1000' width='4'/>` definerer sproget

`{-0100, -0099, ..., -0001, 0000, 0001, 0002, ..., 0998, 0999, 1000}`

og `<interval low='0' high='fff' radix='16' width='3'/>` giver anledning til sproget

`{000, 001, ..., ffe, fff}`.

`<repeat count=count> regexp </repeat>`

Definerer sproget der består af *count* sammensætninger af sproget defineret af *regexp*.

`<repeat [min=min] [max=max]> regexp </repeat>`

Definerer sproget der består af mindst *min* og højst *max* sammensætninger af sproget defineret af *regexp*.

`<complement> regexp </complement>`

Definerer komplementet til sproget defineret af *regexp*.

`<optional> regexp </optional>`

Definerer sproget bestående af den tomme streng plus strengene i sproget defineret af *regexp*.

`<plus> regexp </plus>`

Definerer sproget bestående af én eller flere sammensætninger af sproget defineret af *regexp*.

`<intersection> regexp+ </intersection>`

Definerer sproget bestående af fællesmængden af strengene i sprogene defineret af de regulære udtryk, *regexp⁺*.

`<union> regexp+ </union>`

Definerer sproget bestående af foreningsmængden af strengene i sprogene defineret af de regulære udtryk, *regexp⁺*.

`<concat> regexp+ </concat>`

Definerer sproget bestående af sammensætninger af strenge fra sprogene defineret af de regulære udtryk, *regexp⁺*.

`<regexp pattern=pattern/>`

Definerer et regulært sprog ud fra *pattern*, der er en streng der parses i henhold til grammatikken defineret i `dk.brics.automaton.RegExp` [12] med alle flag slået til.

`<regexp url=url/>`

Henter et regulært udtryk fra *url*'en, *url*, som skal pege på en fil der indeholder en serialiseret instans af klassen `dk.brics.automaton.Automaton` [12]. Se PowerForms-hjemmesiden [<http://www.brics.dk/~ricky/powerforms/tool/>] for en beskrivelse af hvordan filer til senere inkludering kan laves.

`<regexp id=id> regexp </regexp>`

Definerer et navngivet regulært udtryk, med navn *id*, der senere kan refereres til ved brug af `<regexp idref=id/>`.

`<regexp idref=id/>`

Reference til et regulært udtryk defineret med `<regexp id=id/>`.

5.5 Initialisering

Hvis man har prøvet at rode med (dynamiske) HTML-formularer, vil man vide at det ofte er nødvendigt at levere en formular der er helt eller delvist udfyldt på forhånd. Dette kan forekomme i forbindelse med opdatering af en købers adresseoplysninger ved varekøb i en virtuel butik på internettet.

Hvis den aktuelle formular opbygges dynamisk, skal man under opbygningen holde øje med hvornår man indsætter et formularfelt der skal være udfyldt, og så skal der genereres passende kode der kan sætte værdien. Til trods for at denne tilgangsmåde ikke er videre besværlig, er der rige muligheder for at lave fejl, og koden i et CGI-program bliver ikke lettere at læse af at være overrislet med kode til at præudfylde en formular. Noget mere besværligt bliver det hvis det er en statisk formular der skal præudfyldes, idet man så først skal finde ud af hvordan dens struktur er og derefter „bryde ind“ i HTML-koden og tilføje et par stumper kode hist og her.

Opildnet af dette faktum indføres init-elementet i PowerForms:

```
<init [form=form] field=field value=value/>
```

form

Navn på den formular der indeholder feltet der skal initialiseres.

field

Navnet på det felt der skal initialiseres.

value

Den værdi det initialiserede felt skal have.

En brug af init-elementet resulterer i at det angivne felt har værdien *value* når formularen vises. I praksis betyder dette at alle tekst-felter med navn *field* indeholder teksten bestemt af *value*, at radioknapper og afkrydsningsfelter med værdi *value* er valgte og at indgange med værdi *value* i select-elementer er valgt.

5.6 Status

Som redegjort for tidligere signaleres den aktuelle tilstand af et constraint løbende til brugeren, men hvordan signalering præcist foregår styres med et status-element. Afsnit 5.9.2 beskriver hvordan man kan knytte et status-element direkte til et tekstfelt eller et formularconstraint ved at bruge status-elementet i et HTML-dokument.

Som standard indsætter PowerForms automatisk status-elementer for tekstfelter der er underlagt constraints, men som ikke eksplicit er tilknyttet et status-element, og ved at angive et status-element i et PowerForms-dokument, kan man styre hvordan de indsatte status-elementer skal give sig til kende.

I et PowerForms-dokument fungerer status-elementet på samme måde som det tilsvarende element i et HTML-dokument [afsnit 5.9.2], bortset fra at field- og constraint-attributterne ikke er tilladt. Til gengæld er der tilføjet en ekstra *type*, nemlig none der resulterer i at der ikke automatisk indsættes status-elementer.

```
<status [type=type] [red=red] [yellow=yellow] [green=green] [na=na] [blank=blank]/>
```


Eksempel: ofte ser man HTML-formularer hvor der øverst står „felter markeret med * skal udfyldes“. En blæret udgave af dette kan laves således at *'erne forsvinder når feltet er udfyldt korrekt, eller ikke er relevant ved hjælp af følgende PowerForms-besværgelse:

```
<status type="image" red="star.png" yellow="star.png"
green="blank.png" na="blank.png"/>
```

Bemærk at alle relative url'er fortolkes i forhold til *miscurl* [afsnit 5.8].

5.7 Automatisk fuldførelse

Som beskrevet i afsnit 3.4.2 understøtter PowerForms automatisk fuldførelse af værdier i tekst-felter, og dette aktiveres ved brug af autocomplete-elementet.

```
<autocomplete constraint=id [type=type] [image=url] [na=url]/>
```

constraint

Angiver et feltconstraint der skal kunne laves automatisk fuldførelse af. Efterfølgende vil automatisk fuldførelse være mulig for alle tekst-felter som constraintet gælder for.

type

Angiver hvilken slags automatisk fuldførelse der skal laves. Lovlige værdier er unique, prefix og first og semantikken af disse er beskrevet i afsnit 3.4.2. Standardværdien er unique.

image og na

Url'erne i disse attributter angiver et billede der skal vises hvis automatisk fuldførelse er mulig henholdsvis ikke mulig. Begge url'er fortolkes relativt til *miscurl*, og standardværdierne er *autocomplete.gif* og *nocompletion.gif*.

5.8 Indstillinger

Det er i et PowerForms-dokument muligt at ændre visse indstillinger som PowerForms bruger i forbindelse med en transformation af et dokument, og det foregår ved at angive et eller flere option-elementer. Som beskrevet nedenfor er der for tiden to egenskaber man kan sætte:

```
<option name=name value=value/>
```

miscurl

En url hvorfra alle JavaScript-biblioteker der bruges af PowerForms hentes og med mindre andet er angivet, bliver billeder til statusikoner også hentet fra *miscurl*. Url'en skal angive et bibliotek, hvilket vil sige at den skal ende med „/" og i øvrigt være en lovlig http-url [2].

Standardværdien af *miscurl* er <http://www.brics.dk/~ricky/powerforms/>

errortitle

Angiver den tekst der skal stå øverst i det vindue der dukker op og informerer om fejl og mangler i udfyldningen af en formular hvis brugeren forsøger at submitte den før alle tilhørende constraints er opfyldt.

5.9 Udvidelser af HTML

Ud over de ovennævnte konstruktioner til validering introducerer PowerForms en række praktiske udvidelser til HTML-formularer som man kan bruge i et HTML-dokument. Når man transformerer dokumentet, vil PowerForms fjerne alle de PowerForms-specifikke elementer og i stedet indsætte almindeligt HTML-kode [22] som implementerer udvidelserne.

5.9.1 Submit

I HTML findes der to ting der kan submitte en formular, nemlig input-felter med type-attributten sat til submit eller image. Disse input-felter vil give anledning til henholdsvis en knap og et billede, og når man trykker på én af delene submittes formularen. Det synes at være tilfældet at der gennem tiden er brugt ikke så få arbejdsdage (og -nætter) på at lave submit-dimser med et andet udseende, og dette bliver typisk opnået ved at skrive en stump JavaScript-kode der kan submitte en formular, sammen, og så snøre et event på et anker eller en knap op til at aktivere det skrevne kode. Koden der skal skrives, er ikke avanceret, men da den er skrevet mere end én gang (og formentlig vil blive skrevet igen) er der al mulig grund til at sætte det i system. Endnu en gang komme PowerForms den hårdtprøvede webprogrammør til undsætning, og det kan ikke overraske den årvågne læser at det sker med submit-elementet.

Submit-elementer fungerer på samme måde som submit-knapper i HTML, idet det kun er det submit-element der aktiveres' navn og værdi der sendes med tilbage til serveren.

```
<submit [type=type] name=name value=value [ignoreconstraints=ignoreconstraints]/>
```

type

Angiver hvordan submit-elementet skal give sig til kende i det transformerede HTML-dokument. Lovlige værdier er anchor (standard), button og submit. Når typen er anchor eller button, kan elementet indeholde den tekst der skal stå i ankeret eller på knappen, og hvis elementet er tomt, vil værdien af value blive brugt. Elementet skal være tomt når typen er submit.

name

Navnet der skal submittes.

value

Værdien der skal submittes.

ignoreconstraints

Hvis værdien af denne attribut er „yes“, vil alle felt- og formularconstraints blive ignoreret når elementet aktiveres, og formularen vil altid kunne submittes.

Traditionelle submit-knapper i HTML (<input type="submit" .../>) har fået tilføjet en ignoreconstraints-attribut med samme effekt som netop beskrevet.

5.9.2 Status

Som bekendt indsætter PowerForms automatisk status-elementer for tekstfelter der er underlagt et feltconstraint. Ved at bruge et status-element i et HTML-dokument er det muligt at styre præcist hvor status-elementet skal indsættes, og hvordan det skal se ud. Det er også muligt at knytte et status-element til et formularconstraint.

Når et tekstfelt er underlagt et feltconstraint og samtidig har tilknyttet et status-element, vil status-elementet løbende afspejle tilstanden af constraintet evalueret ud fra tekstfeltets

værdi. Tilsvarende vil et status-element for et formularconstraint vise den aktuelle tilstand af constraintet ud fra feltværdierne på den relevante formular.

Der findes tre forskellige måder hvorpå et status-element kan give sig til kende på en HTML-side, og den første måde er ved brug af billeder som beskrevet tidligere. Den anden måde er ved brug af CSS, hvorved tilstanden af et constraint for et tekstfelt vil afspejles i værdien af tekstfeltet class-attribut der styrer feltets udseende. Endelig kan constraint-tilstanden vises med tekstbeskeder der opdateres på passende vis. De sidste to måder virker kun i moderne browsere, dvs. uden for akademiske kredse, mens „den med billeder“ virker i formentlig alle browsere.

```
<status constraint=id [type=type] [red=red] [yellow=yellow] [green=green]
  [na=na] [blank=blank]/>
```

```
<status field=id [type=type] [red=red] [yellow=yellow] [green=green]
  [na=na] [blank=blank]/>
```

constraint og field

Knytter status-elementet til et constraint eller et tekstfelt med en id-attribut svarende til værdien af id.

type

Denne attribut styrer hvordan status-elementet skal vises på HTML-siden og lovlige værdier er image, css og text, hvor image er standard.

red, yellow, green og na

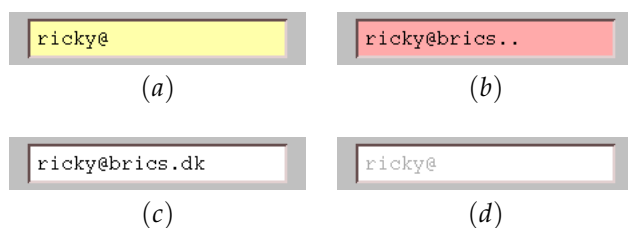
Lovlige værdier for disse attributter afhænger af værdien af type-attributten. Hvis typen er image, fortolkes attributværdierne som url'er der udpeger de billeder der skal vise det underliggende constraints tilstand. Alle billeder bør have samme størrelse.

Når typen er css, bliver værdierne opfattet som navne på allerede definerede CSS-klasser [5] der vil blive brugt via et tekstelements class-attribut som illustreret i figur 24.

Endelig kan typen være text, og i dette tilfælde vil status-elementet blive realiseret som et span-element [22, afsnit 7.5.4] hvis tekstuelle indhold angiver tilstanden af constraintet. Figur 25 viser et eksempel på dette.

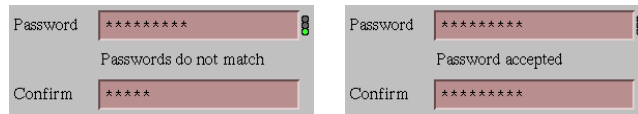
blank

Denne attribut er kun relevant hvis typen er image, og værdien skal i så fald udpege et billede der vises når status-elementet defineres i en browser. Billedets eneste funktion er at lave plads til status-elementet, og det bør således have samme størrelse som billederne angivet af red, yellow, green og na.



Figur 24: Et tekstfelt hvor tilstanden i et tilknyttet constraint vises ved brug af CSS. I (a) er værdien endnu ikke accepteret, i (b) er den ulovlig og i (c) er den accepteret. I (d) er tekstfeltet gjort inaktivt, idet det tilhørende feltconstraint evaluerer til ignore.

```
<status constraint="matching-passwords"  
  type="text"  
  green="Password accepted"  
  yellow="Passwords do not match!"  
  red="Passwords do not match!"/>
```



Figur 25: Et status-element med type text knyttet til et formularconstraint. Nederst ses status-elements manifestation i en browser, hvor det også bemærkes at det øverste tekstfelt er underlagt et feltconstraint.

5.9.3 Andet

En variant af init-elementet er tilføjes af en value-attribut til select-elementet i HTML. Hvis denne attribut er til stede, vil de indgange i listen hvis værdi er lig værdien af attributten, være valgt når formularen dukker op i browseren.

Som nævnt i afsnit 5.9.1 er der tilføjet en ignoreconstraints-attribut til input-elementer med type submit.

Kapitel 6

Afslutning

Dette speciale har beskrevet en række forskellige forslag til, hvordan webbaserede formularer kan moderniseres så de svarer til de krav der kan stilles. XForms og XFDL har begge taget hele formularbegrebet under behandling, men fra forskellige tilgangsvinkler og derfor er resultaterne også meget forskellige. Naturligvis indeholder begge forslag mulighed for at validere værdier i formularer, men som beskrevet er der visse oplagte valideringskrav som ikke kan indfanges af valideringsmekanismerne. Yderligere kræves der visse steder et vist kendskab til traditionel programmering i et scriptsprog for at få det fulde udbytte af valideringen.

I PowerForms er det kun en lille del af formularbegrebet der søges løst, nemlig validering af værdier af felter på formularen. Den præsenterede løsning er simpel og elegant og giver mulighed for at udtrykke mange interessante – og realistiske – valideringskrav.

Både XForms og XFDL lider under at det er meget svært at skaffe sig praktiske erfaringer med dem og deres anvendelighed – eller mangel på samme – hvilket til dels skyldes at de begge baserer sig på nye teknologier som hverken programmør eller bruger umiddelbart er i besiddelse af.

Derimod baserer PowerForms sig på teknologier som er blevet hvermands eje, og dermed kan PowerForms umiddelbart prøves i praksis, så man kan danne sig et fornuftigt indtryk af hvordan det rent faktisk virker. Selv om HTML-formularer har mange år på bagen og ikke helt lever op til de forventninger man i dag har til elektroniske formular, findes der mange af dem som fungerer ganske udemærket, og hvis disse formularer, ved brug af PowerForms, kan blive nemmere at bruge for både programmør og bruger, er det oplagt at PowerForms har sin berettigelse.

Det er klart at PowerForms kun løser en lille del af hvad der skal til for at gøre webbaserede formularer tidssvarende. Især dynamiske formularer, hvor felter kun dukker op hvis de skal bruges til noget, synes at være noget der savnes, og både XForms og XFDL understøtter derfor dette. Et andet problem ved eksisterende formularer, altså hovedsageligt HTML-formularer, er at de værdier der returneres når en formular submittes, blot er en række par af navne og værdier, hvilket heller ikke lever op til nutidens krav om hvilke data en elektroniske formular skal kunne manipulere. En oplagt forbedring er at lade en formular redigere en XML-struktur som i XForms, men også XFDLs løsning hvor der ikke er adskillelse mellem en formular og dens data er elegant.

Uanset hvordan fremtiden former sig for webbaserede formularer, står det forhåbentlig klart at PowerForms er et godt bud på hvordan man kan lave validering, og at PowerForms gør det nemmere for både programmør og bruger at anvende *automatisk validering af webbaserede formularer*.

Appendiks A

Det illustrative eksempel

I det følgende findes den komplette kildekode til det illustrative eksempel formuleret i XForms, XFDL og PowerForms og for at skabe lidt overblik opridses nedenfor hvilke dokumenter der hører sammen.

For ikke at besvære læseren med unødigt bladren bringes XFDL-versionen af typografiske årsager før XForms-versionen.

XFDL i New York

I XFDL er det som bekendt kun nødvendigt med ét dokument:

- NYC.xfdl [side 88]

XForms i New York

XForms-versionen består af to dokumenter:

- NYC.xhtml [side 90]: selve definitionen af formularen i XForms og XHTML.
- NYC.xsd [side 92]: et skema for instansdata.

PowerForms i New York

Når eksemplet formuleres i PowerForms kræves både et HTML-dokument og en samling constraints i et PowerForms-dokument:

- NYC.htm [side 93]: selve formularen opbygget i HTML.
- NYC.pwf [side 94]: en passende PowerForms-specifikation.

Alle eksemplerne kan findes i elektronisk form på <http://www.brics.dk/~ricky/speciale/>.

A.1 XFDL i New York

NYC.xfdl

```
<XFDL version="4.5.0">
  <page sid="NYC">
    <field sid="name">
      <label>Name</label>
    </field>

    <field sid="e_mail">
      <label>E-mail</label>
      <format content="array">
        <ae>string</ae>
        <ae>mandatory</ae>
        <message>Invalid e-mail address</message>
      </format>
    </field>

    <popup sid="country">
      <label>Country</label>
      <group>country-group</group>
    </popup>

    <cell sid="country_dk">
      <label>Denmark</label>
      <value>dk</value>
      <group>country-group</group>
    </cell>

    <cell sid="country_usa">
      <label>USA</label>
      <value>usa</value>
      <group>country-group</group>
    </cell>

    <field sid="zip_code">
      <label>Zip code</label>
      <format content="array">
        <ae>string</ae>
        <ae content="compute">
          <compute>
            NYC.country.value == "country_usa" ? "mandatory" : "optional"
          </compute>
        </ae>
        <template content="array">
          <ae>#####</ae>
        </template>
        <message>Invalid zip code</message>
      </format>
      <visible content="compute">
        <compute>NYC.country.value == "country_usa" ? "on" : "off"</compute>
      </visible>
    </field>
```



```

<field sid="phone">
  <label>Phone</label>
  <format content="array">
    <ae>string</ae>
    <template content="array">
      <ae>#####</ae>
      <ae>###-###-####</ae>
    </template>
    <message>Invalid phone number</message>
  </format>
</field>

<check sid="visit">
  <label>Request visit from NYC office</label>
  <visible content="compute">
    <compute>
      (NYC.country.value == "country_usa"
      & & ((strmatch("212*", NYC.phone.value) == "1")
      || (strmatch("347*", NYC.phone.value) == "1")
      || (strmatch("646*", NYC.phone.value) == "1")
      || (strmatch("718*", NYC.phone.value) == "1")
      || (strmatch("917*", NYC.phone.value) == "1"))) ? value : "off"
    </compute>
  </visible>
</check>

<button sid="ok">
  <value>OK</value>
  <type>submit</type>
  <url content="array">
    <ae>http://example.com/submit</ae>
  </url>
</button>

<button sid="cancel">
  <value>Cancel</value>
  <type>cancel</type>
  <itemlocation content="array">
    <ae content="array">
      <ae>after</ae>
      <ae>ok</ae>
    </ae>
  </itemlocation>
</button>
</page>
</XFDL>

```

A.2 XForms i New York

NYC.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xfm="http://www.w3.org/2002/01/xforms"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <head>
    <title>XForms - NYC</title>

    <xfm:model id="NYC">
      <xfm:instance xmlns="">
        <problem>
          <personal>
            <name/>
            <e-mail/>
            <country>dk</country>
            <zip-code/>
            <phone/>
          </personal>
          <visit>>false</visit>
        </problem>
      </xfm:instance>

      <xfm:schema href="NYC.xsd"/>

      <xfm:submitInfo id="ok" method="post" action="http://example.com/submit"/>

      <xfm:submitInfo id="cancel" method="post" ref=""
        action="http://example.com/submit"/>

      <xfm:bind id="bind-zip-code" ref="/problem/personal/zip-code"
        type="zip-code"
        relevant="/problem/personal/country = 'usa'"/>

      <xfm:bind id="bind-phone" ref="/problem/personal/phone"/>

      <xfm:bind id="bind-visit" ref="/problem/visit"
        relevant="(/problem/personal/country = 'usa')
          and ((substring(/problem/personal/phone, 1, 3) = '212')
            or (substring(/problem/personal/phone, 1, 3) = '347')
            or (substring(/problem/personal/phone, 1, 3) = '646')
            or (substring(/problem/personal/phone, 1, 3) = '718')
            or (substring(/problem/personal/phone, 1, 3) = '917'))"/>
    </xfm:model>
  </head>

  <body>
    <p>
      <xfm:input ref="/problem/personal/name" style="width: 8cm">
        <xfm:caption>Name</xfm:caption>
      </xfm:input>
    </p>
  </body>
</html>
```

```

<p>
  <xfm:input ref="/problem/personal/e-mail" style="width: 8cm">
    <xfm:caption>E-mail</xfm:caption>
    <xfm:alert>Invalid e-mail address</xfm:alert>
  </xfm:input>
</p>

<p>
  <xfm:selectOne ref="/problem/personal/country">
    <xfm:caption>Country</xfm:caption>
    <xfm:choices>
      <xfm:item>
        <xfm:caption>Danmark</xfm:caption>
        <xfm:value>dk</xfm:value>
      </xfm:item>
      <xfm:item>
        <xfm:caption>USA</xfm:caption>
        <xfm:value>usa</xfm:value>
      </xfm:item>
    </xfm:choices>
  </xfm:selectOne>
</p>

<p>
  <xfm:input ref="/problem/personal/zip-code" style="width: 8cm">
    <xfm:caption>Zip code</xfm:caption>
    <xfm:alert>Invalid zip code</xfm:alert>
  </xfm:input>
</p>

<p>
  <xfm:input ref="/problem/personal/phone" style="width: 8cm">
    <xfm:caption>Phone</xfm:caption>
    <xfm:alert>Invalid phone number</xfm:alert>
  </xfm:input>
</p>

<p>
  <xfm:selectBoolean ref="/problem/visit">
    <xfm:caption>Request visit from NYC office</xfm:caption>
  </xfm:selectBoolean>
</p>

<p style="text-align: right">
  <xfm:submit submitInfo="ok">
    <xfm:caption>OK</xfm:caption>
  </xfm:submit>
  <xfm:submit submitInfo="cancel">
    <xfm:caption>Cancel</xfm:caption>
  </xfm:submit>
</p>
</body>
</html>

```

NYC.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="e-mail">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-z]+@([a-z]+\.)+[a-z]+" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="zip-code">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([0-9]{5})?" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="phone">
    <xsd:union>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{3}-?[0-9]{3}-?[0-9]{4}" />
      </xsd:restriction>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{8}" />
      </xsd:restriction>
    </xsd:union>
  </xsd:simpleType>

  <xsd:simpleType name="boolean">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="true" />
      <xsd:enumeration value="false" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="problem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="personal" />
        <xsd:element name="visit" type="boolean" minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="personal">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" />
        <xsd:element name="e-mail" type="e-mail" />
        <xsd:element name="country" />
        <xsd:element name="zip-code" type="zip-code" minOccurs="0" />
        <xsd:element name="phone" type="phone" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

A.3 PowerForms i New York

NYC.htm

```
<html>
  <head><title>NYC</title></head>
  <body>
    <form>
      <h1>Personal Information</h1>

      <table>
        <tr>
          <td>Name:</td>
          <td><input type="text" name="name" size="20"/></td>
        </tr>
        <tr>
          <td>E-mail:</td>
          <td><input type="text" name="e-mail" size="20"/></td>
        </tr>
        <tr>
          <td>Country:</td>
          <td>
            <select name="country">
              <option value="dk">Denmark</option>
              <option value="usa">USA</option>
            </select>
          </td>
        </tr>
        <tr>
          <td>Zip code:</td>
          <td><input type="text" name="zip-code" size="20"/></td>
        </tr>
        <tr>
          <td>Phone:</td>
          <td><input type="text" name="phone" size="20"/></td>
        </tr>
        <tr>
          <td colspan="2">
            Request visit from NYC office
            <input type="checkbox" name="visit" value="true"/>
          </td>
        </tr>
        <tr>
          <td colspan="2" align="right">
            <input type="submit" name="Submit" value="OK"/>
            <input type="submit" name="Submit" value="Cancel"
              ignoreconstraints="yes"/>
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

NYC.pwf

```
<powerforms>
  <constraint field="e-mail">
    <match warning="Incomplete e-mail address"
      error="Invalid e-mail address">
      <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/e-mail.dfa"/>
    </match>
  </constraint>

  <constraint field="zip-code">
    <if>
      <equal field="country" value="usa"/>
      <then>
        <match>
          <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/zip-code.dfa"/>
        </match>
      </then>
      <else>
        <ignore/>
      </else>
    </if>
  </constraint>

  <regexp-def id="sep">
    <optional><charset value="-"/></optional>
  </regexp-def>

  <regexp-def id="digit">
    <charset value="0123456789"/>
  </regexp-def>
```

```

<constraint field="phone" error="Invalid phone number">
  <if>
    <equal field="country" value="usa"/>
    <then>
      <concat>
        <repeat count="3">
          <regexp idref="digit"/>
        </repeat>
        <regexp idref="sep"/>
        <repeat count="3">
          <regexp idref="digit"/>
        </repeat>
        <regexp idref="sep"/>
        <repeat count="4">
          <regexp idref="digit"/>
        </repeat>
      </concat>
    </then>
    <else>
      <concat>
        <repeat count="8">
          <regexp idref="digit"/>
        </repeat>
      </concat>
    </else>
  </if>
</constraint>

<constraint field="visit">
  <if>
    <and>
      <equal field="country" value="usa"/>
      <match field="phone">
        <concat>
          <union>
            <const value="212"/><const value="347"/><const value="646"/>
            <const value="718"/><const value="917"/>
          </union>
          <anything/>
        </concat>
      </match>
    </and>
    <then><match><anything/></match></then>
    <else><match><empty/></match></else>
  </if>
</constraint>
</powerforms>

```

Appendiks B

PowerForms i den virkelige verden

I dette appendiks findes PowerForms-specifikationer hørende til eksemplerne fra den virkelige verden [afsnit 4]. Af typografiske årsager er det lykkedes for „Barnets bog“ at snige om foran „Java Developer Connection“.

Eksemplerne kan prøves på <http://www.brics.dk/~ricky/powerforms/virkelighed/>, hvor specifikationerne også kan findes i elektronisk form.

B.1 Extensible Forms Description Language

ICS.pwf

```
<powerforms>
  <regexp id="non-empty">
    <regexp pattern=".*[\t].*" />
  </regexp>

  <regexp id="required">
    <complement>
      <const value="0" />
    </complement>
  </regexp>

  <constraint field="r_firstname">
    <match error="You must enter your first name">
      <regexp idref="non-empty" />
    </match>
  </constraint>

  <constraint field="r_lastname">
    <match error="You must enter your last name">
      <regexp idref="non-empty" />
    </match>
  </constraint>

  <constraint>
    <match field="r_title" error="Please select your title">
      <regexp idref="required" />
    </match>
  </constraint>

  <constraint field="r_organization">
    <match error="You must enter your organization">
      <regexp idref="non-empty" />
    </match>
  </constraint>

  <constraint field="r_address1">
    <match error="You must enter your address">
      <regexp idref="non-empty" />
    </match>
  </constraint>

  <constraint field="r_city">
    <match error="You must enter your city">
      <regexp idref="non-empty" />
    </match>
  </constraint>
```

```

<constraint field="r_state">
  <match error="You must enter your state or province">
    <regexp idref="non-empty"/>
  </match>
</constraint>

<constraint field="r_postalcode">
  <match error="You must enter your postal code">
    <regexp idref="non-empty"/>
  </match>
</constraint>

<constraint>
  <match field="r_country" error="Please select your country">
    <regexp idref="required"/>
  </match>
</constraint>

<constraint field="r_phone">
  <match error="You must enter your phone number">
    <regexp idref="non-empty"/>
  </match>
</constraint>

<constraint field="r_email">
  <match error="You must enter your e-mail address">
    <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/e-mail.dfa"/>
  </match>
</constraint>

<constraint>
  <match field="r_referral" error="Please state how you heard about PureEdge.com">
    <regexp idref="required"/>
  </match>
</constraint>

<constraint>
  <match field="r_orgsize" error="Please state the size of your organization">
    <regexp idref="required"/>
  </match>
</constraint>

<constraint>
  <match field="r_budgetsize" error="Please state the size of your budget">
    <regexp idref="required"/>
  </match>
</constraint>

```

```

<constraint>
  <match field="r_project" error="Please state whether you're evaluating
    InternetForms for a current project">
    <regexp idref="required"/>
  </match>
</constraint>

<constraint>
  <if>
    <equal field="r_project" value="1"/>
    <then>
      <match field="r_timeframe" error="Please state your timeframe">
        <regexp idref="required"/>
      </match>
    </then>
    <else>
      <ignore field="r_timeframe"/>
    </else>
  </if>
</constraint>

<constraint>
  <match field="r_useforms" error="Please state whether your organization
    is using Internet-based forms">
    <regexp idref="required"/>
  </match>
</constraint>

<constraint>
  <if>
    <equal field="r_useforms" value="1"/>
    <then>
      <match field="r_currentforms" error="Please state your current form solution">
        <regexp idref="non-empty"/>
      </match>
    </then>
    <else>
      <ignore field="r_currentforms"/>
    </else>
  </if>
</constraint>

<constraint>
  <match field="r_operatingsystem" error="Please select your operating system">
    <regexp idref="required"/>
  </match>
</constraint>
</powerforms>

```

B.2 Barnets bog

BB.pwf

```
<powerforms>
  <status blank="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
    na="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
    green="http://www.brics.dk/~ricky/powerforms/misc/blank.gif"
    yellow="http://www.brics.dk/~ricky/powerforms/misc/red-dot.gif"
    red="http://www.brics.dk/~ricky/powerforms/misc/red-dot.gif" />

  <option name="errortitle" value="Fejl i indtastninger" />

  <regexp id="phone-dk">
    <optional>
      <repeat count="8">
        <charset value="0123456789" />
      </repeat>
    </optional>
  </regexp>

  <constraint form="f0" field="DATEOFBIRTH">
    <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/date.dfa" />
  </constraint>

  <constraint field="TIMEOFBIRTH">
    <match error="Du har indtastet et ugyldigt tidspunkt.
      Tidspunktet skal have følgende format tt.mm - forsøg venligst igen!">
      <union>
        <const value="tt.mm" />
        <optional>
          <concat>
            <union>
              <interval low="0" high="23" />
              <interval low="0" high="23" width="2" />
            </union>
            <const value="." />
            <interval low="0" high="59" width="2" />
          </concat>
        </optional>
      </union>
    </match>
  </constraint>
```

```

<constraint field="WEIGHTATBIRTH">
  <match error="Fødselsvægt skal angives som et tal med maks 2 decimaler
    - forsøg venligst igen.">
    <optional>
      <concat>
        <interval low="2" high="9"/>
        <optional>
          <concat>
            <const value=","/>
            <union>
              <interval low="0" high="99"/>
              <interval low="0" high="99" width="2"/>
            </union>
          </concat>
        </optional>
      </concat>
    </optional>
  </match>
</constraint>

<constraint field="LENGTHATBIRTH">
  <match error="Fødselshøjde skal angives som et tal, venligst prøv igen.">
    <optional>
      <interval low="20" high="70"/>
    </optional>
  </match>
</constraint>

<constraint field="PHONEDOCTOR">
  <match error="Ugyldigt telefonnummer til læge">
    <regexp idref="phone-dk"/>
  </match>
</constraint>

<constraint field="PHONENURSE">
  <match error="Ugyldigt telefonnummer til sundhedsplejerske">
    <regexp idref="phone-dk"/>
  </match>
</constraint>

<constraint field="PHONENURSERY">
  <match error="Ugyldigt telefonnummer til dagsinstitution">
    <regexp idref="phone-dk"/>
  </match>
</constraint>
</powerforms>

```

B.3 Java Developer Connection

JDC.pwf

```
<powerforms>
  <regexp id="something">
    <complement>
      <const value="" />
    </complement>
  </regexp>

  <constraint field="UserId">
    <match error="UserId is required">
      <regexp idref="something"/>
    </match>
  </constraint>

  <constraint field="Password1">
    <match error="Password1 is required">
      <regexp idref="something"/>
    </match>
  </constraint>

  <constraint id="matching-passwords">
    <equal error="Password 1 and Password 2 do not match. Try again.">
      <field name="Password1"/>
      <field name="Password2"/>
    </equal>
  </constraint>

  <constraint field="Email">
    <regexp url="http://www.brics.dk/~ricky/powerforms/regexp/e-mail.dfa"/>
  </constraint>

  <constraint field="FirstName">
    <match error="FirstName is required">
      <regexp idref="something"/>
    </match>
  </constraint>

  <constraint field="LastName">
    <match error="LastName is required">
      <regexp idref="something"/>
    </match>
  </constraint>

  <constraint>
    <match field="Country" error="Please select a country">
      <regexp pattern="[A-Z].*" />
    </match>
  </constraint>
```

```

<constraint field="html_pref">
  <if>
    <or>
      <match field="wantAllSubs"><anything/></match>
      <match field="ed_892"><anything/></match>
      <match field="ed_897"><anything/></match>
      <match field="ed_896"><anything/></match>
      <match field="ed_895"><anything/></match>
      <match field="ed_894"><anything/></match>
      <match field="ed_893"><anything/></match>
      <match field="jdc_1092"><anything/></match>
      <match field="jdc_1093"><anything/></match>
    </or>
    <then><anything/></then>
    <else><empty/></else>
  </if>
</constraint>

```

```

<constraint field="ed_892">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

```

```

<constraint field="ed_897">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

```

```

<constraint field="ed_896">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

```

```

<constraint field="ed_895">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

```

```
<constraint field="ed_894">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

<constraint field="ed_893">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

<constraint field="jdc_1092">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>

<constraint field="jdc_1093">
  <if>
    <match field="wantAllSubs"><anything/></match>
    <then><empty/></then>
    <else><anything/></else>
  </if>
</constraint>
</powerforms>
```


Appendiks C

PowerForms i JWIG

På bagsiden af dette ark er hele kildeteksten til den JWIG-service der i uddrag er vist i afsnit 3.6 at læse.

En elektronisk udgave af kildeteksten og et link til den kørende service kan findes på websiden <http://www.brics.dk/~amoeller/WWW/powerforms/pwfjwig.html>.

PowerFreebie.jwig

```
import dk.brics.jwig.runtime.*;
import java.util.*;
```

```
public class PowerFreebie extends Service {
    public class HowMany extends Session {
        static final int MAX = 5;
```

```
        XML templateAsk = [[ <html>
                                <body>
                                    <form>
                                        <[msg]>
                                        <p>
                                            How many free T-shirts do you want?
                                            <input name="amount" type="text"/>
                                            <input name="continue" type="submit"/>
                                        </p>
                                    </form>
                                </body>
                            </html> ]];
```

```
        XML templateReply = [[ <html>
                                <body>
                                    You will receive <[amount]> k00l T-shirts any day now...
                                </body>
                            </html> ]];
```

```
        XML format = [[ <powerforms>
                            <constraint field="amount">
                                <match>
                                    <interval low="1" high=[high]/>
                                </match>
                            </constraint>
                        </powerforms> ]];
```

```
    public void main() {
        String msg = "";
        int amount;
        boolean ok = false;
        while (!ok) {
            ok = true;
            try {
                show templateAsk<[msg=msg] powerforms format<[high=MAX];
            } catch (PowerFormsValidateException e) {
                ok = false;
                msg = e.getMessage();
            }
        }
        amount = Integer.parseInt(receive amount);
        exit templateReply<[amount=amount];
    }
}
```

Appendiks D

PowerForms-grammatik

På de næste par sider findes en grammatik for PowerForms-dokumenter i bnf-notation. På traditionel vis angiver ornamentet "*" nul eller flere forekomster, "+" angiver én eller flere forekomster og "|" angiver valg. Attributter omgivet af "[" og "]" kan udelades.

Kapitel 5 giver en uddybende forklaring af semantikken af de forskellige konstruktioner.

<i>powerforms</i>	→	<pre> <powerforms> (regexp-def constraint status initialization autocomplete)* </powerforms> </pre>
<i>regexp-def</i>	→	<pre> <regexp id=id> regexp </regexp> </pre>
<i>constraint</i>	→	<pre> <constraint [form=form] [field=field] [id=id]> constraint-body </constraint> </pre>
<i>status</i>	→	<pre> <status [type=type] [red=red] [yellow=yellow] [green=green] [na=na]/> </pre>
<i>initialization</i>	→	<pre> <init [form=form] field=field value=string/> </pre>
<i>autocomplete</i>	→	<pre> <autocomplete constraint=id [type=type] [image=url] [na=url]/> </pre>
<i>constraint-body</i>	→	<pre> <if> expression <then> constraint-body </then> <else> constraint-body </else> </if> <ignore/> regexp expression </pre>
<i>expression</i>	→	<pre> <count field=field number=int/> <count min=int max=int/> <equal field=field value=string/> <equal> field+ </equal> <less-than field=field value=value [type=type]/> <less-than [type=type]> field+ </less-than> <match field=field> regexp </match> <not> expression* </not> <and> expression* </and> <or> expression* </or> </pre>
<i>field</i>	→	<pre> <field name=name/> </pre>

regexp

→ <empty/>
| <anychar/>
| <anything/>
| <const value=*string*/>
| <charset value=*string*/>
| <charrange low=*char* high=*char*/>
| <interval low=*int* high=*int*
| [width=*int*] [radix=*int*]/>
| <repeat count=*int*>
| *regexp*
| </repeat>
| <repeat [min=*int*] [max=*int*]>
| *regexp*
| </repeat>
| <complement> *regexp* </complement>
| <optional> *regexp* </optional>
| <plus> *regexp* </plus>
| <intersection> *regexp*⁺ </intersection>
| <union> *regexp*⁺ </union>
| <concat> *regexp*⁺ </concat>
| <regexp url=*url*/>
| <regexp pattern=*pattern*/>
| <regexp idref=*idref*/>

Litteratur

- [1] T. Berners-Lee og D. Connolly (red.). Hypertext Markup Language - 2.0. November 1995.
<http://www.ietf.org/rfc/rfc1866.txt>
- [2] T. Berners-Lee, R. Fielding, U.C. Irvine og L. Masinter (red.). Uniform Resource Identifiers (URI): Generic Syntax. august 1998.
<http://www.ietf.org/rfc/rfc2396.txt>
- [3] Paul V. Biron og Ashok Malhotra (red.). XML Path Language (XPath) Version 1.0. W3C Recommendation, november 1999.
<http://www.w3.org/TR/xpath/>
- [4] Paul V. Biron og Ashok Malhotra (red.). XML Schema Part 2: Datatypes. W3C Recommendation, maj 2001.
<http://www.w3.org/TR/xmlschema-2/>
- [5] Bert Bos, Håkon Wium Lie, Chris Lilley og Ian Jacobs (red.). Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation, maj 1998.
<http://www.w3.org/TR/REC-CSS2/>
- [6] John Boyer, Tim Bray og Maureen Gordon (red.). XFDL Specification v4.5. Oktober 2001.
http://docs.pureedge.com/xfdldocs/pdfdocs/XFDL_45.pdf
- [7] Claus Brabrand (red.). Wig Projects. 1998.
<http://www.brics.dk/bigwig/wigprojects/wigprojects.ps>
- [8] Claus Brabrand, Anders Møller, Mikkel Ricky og Michael I. Schwartzbach. PowerForms: Declarative Client-side Form Field Validation. *World Wide Web*, 3(4), 2000.
- [9] Claus Brabrand, Anders Møller og Michael I. Schwartzbach. The <bigwig> project. *ACM Transactions on Internet Technology*, 2002.
- [10] Tim Bray, Dave Hollander og Andrew Layman (red.). Namespaces in XML. W3C Recommendation, januar 1999.
<http://www.w3.org/TR/REC-xml-names/>
- [11] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen og Eve Maler (red.). Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, oktober 2000.
<http://www.w3.org/TR/REC-xml/>
- [12] Anders Møller. [dk.brics.automaton.RegExp](http://www.brics.dk/~amoeller/automaton/doc/dk/brics/automaton/RegExp.html). Marts 2002.
- [13] Aske Simon Christensen, Anders Møller og Michael I. Schwartzbach. Extending Java for High-Level Web Service Construction. 2002.

- [14] David H. Crocker (red.). Standard for ARPA Internet Text Messages. August 1982.
<http://www.ietf.org/rfc/rfc822.txt>
- [15] Helle Markmann, Mikkel Ricky, Jens Erik Vestergaard og Lone Haudrum Olesen. Power Forms – en del af **W3C**-imperiet. 1998.
<http://www.brics.dk/bigwig/wigprojects/powerforms.ps>
- [16] Micah Dubinko, Josef Dietl, Jr. Leigh L. Klotz, Roland Merrick og T. V. Raman (red.). XForms 1.0. W3C Working Draft, januar 2002.
<http://www.w3.org/TR/xforms/>
- [17] ECMAScript Language Specification. Tredje udgave. December 1999.
<http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>
- [18] Carla & Vilh. Hansen. *Rasmus Klump får brev fra Bedstefar*. Forlaget Carlsen, 1979.
- [19] John C. Martin. *Introduction to languages and the theory of computation*. McGraw Hill, Inc., 1991.
- [20] Anders Møller. Document Structure Description 2.0. In preparation, 2002.
- [21] Cassandra Greer. *Mozquito Matrix: XHTML-FML Reference Guide*. 2001.
<http://www.mozquito.org/sources/xhtmlfmlreference.pdf>
- [22] Dave Raggett, Arnaud Le Hors og Ian Jacobs (red.). HTML 4.01 Specification. December 1999.
<http://www.w3.org/TR/html4/>
- [23] Mikkel Ricky. *Automatisk validering af webbaserede formularer*. Speciale, Daimi, Aarhus Universitet, maj 2002.
- [24] Henry S. Thompson, David Beech, Murray Maloney og Noah Mendelsohn (red.). XML Schema Part 1: Structures. W3C Recommendation, maj 2001.
<http://www.w3.org/TR/xmlschema-1/>
- [25] Document Object Model (DOM).
<http://www.w3.org/DOM/>
- [26] HTML. November 1992.
<http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html>