Automatisk lokalisering af dynamisk indhold på World Wide Web

Torin Finnemann Jensen - 20040813

Automatic Re-localizing of Dynamic Content on the World Wide Web
Master's Thesis

Specialeafhandling

Vejleder:
Niels Olof Bouvin

Datalogisk Institut, Aarhus Universitet
15. februar 2011

Abstract

English

This thesis investigates previous attempts at designing Location Specifiers for the World Wide Web, as could be used in annotation systems or other external applications that need to identify the same piece of content even after the containing document has changed. It evaluates the known methods' ability to handle the increasingly dynamic nature of web content. Furthermore it gives a bidding as to how to improve handling in order to ease the process of externally referencing specific parts of a web page.

Dansk

Dette speciale afdækker tidligere bud på Location Specifiers (lokationsspecifikationer) for World Wide Web, der eksempelvis kan bruges i annotationssystemer eller andre eksterne systemer, der har behov for at kunne identificere et stykke af indholdet på en webside gang på gang, selv efter siden er ændret. Specialet evaluerer de kendte metoders evne til at håndtere webindholdets stadigt mere dynamiske natur. Endvidere giver det et bud på, hvordan man bedre kan udvikle håndteringen, så muligheden for at referere specifikke dele af en webside fra eksterne kilder forbedres.

Indhold

1

Introduktion

World Wide Web (WWW) er i dag det langt mest udbredte hypermediesystem. Det er det blevet på trods af, at det på mange områder mangler muligheder og funktionaliteter, der var implementeret i tidligere hypermediesystemer. Det er særligt i forbindelse med samarbejde, at WWW har begrænsninger. En side på WWW er som udgangspunkt skrivebeskyttet for alle andre end ejeren af dokumentet. Det betyder, der ikke er nogen umiddelbart tilgængelig måde at annotere indholdet på siden på. Endvidere er WWWs links og ankre indlejrede sammen med indholdet, således at det hverken er muligt tilføje links eller linke til et specifikt element på siden, hvis ikke forfatteren har specificeret et anker der.

Der findes mange webapplikationer, der har til formål at gøre online samarbejde nemmere - for eksempel Wikipedia[1], diverse content management systemer og Google Docs[2]. Fora og blogs gør det muligt for besøgende på en webside at skrive kommentarer. Men alle disse giver kun mulighed for samarbejde på de websteder, hvor applikationerne er installeret. Man kan ikke generelt annotere websider med links, ankre eller noter.

Udbredelsen af disse applikationer betyder imidlertid, at WWW i dag er mere levende end nogensinde. Indhold kan ændres fra det ene øjeblik til det andet, der kan blive tilføjet nyt, og indholdet kan blive flyttet til en anden side eller helt slettet. I dette speciale vil begrebet dynamisk indhold dække over indhold på websider, der forandres over tid.

Brugsscenarier

De følgende tænkte brugsscenarier beskriver, hvordan annotering af web-sider kan være nyttigt. De illustrerer også nogle af de væsentligste udfordringer, der er forbundet med annotationer på det dynamiske World Wide Web.

Samarbejde

Carl Sunesen indgår som del af et team på sit arbejde. I forbindelse med arbejdet benytter teamet en stribe tredjeparts websteder, der indeholder relevante oplysninger, regler og andre informationer. De vigtigste passager på webstederne er for længst blevet markeret med virksomhedens annoteringsværktøj, så de fremstår tydeligt og hurtigt kan identificeres,

[1] `http://wikipedia.org/`, Online leksikon

[2] `http://docs.google.com`, Web baseret kontorpakke

når medarbejderne kommer forbi, ligesom konklusionen på de væsentligste afsnit er noteret i sidernes margen. Carl Sunesen sidder netop denne dag og skal bruge oplysninger fra et af webstederne, da han til sin store irritation opdager, at webstedet har gennemgået en større revidering. Indholdet ser godt nok ud til at være stort set det samme, og på nogle punkter er det forbedret. Carl Sunesens irritation skyldes, at ingen teamets af annotationer længere er korrekt placeret på siden.

Link-database

Erika Byskov er researcher for et mellemstort reklamebureau. Hendes job består i at indsamle relevante artikler fra diverse net-medier, og hvis der er tale om større tekster at udvælge de dele, der er af interesse. Hun bruger et særligt program, der gemmer hendes links direkte til de udvalgte dele af websiderne i forskellige kategorier. En dag beder en af de kreative medarbejdere hende om en kilde, der ligger inden for to forskellige kategorier. Til sin gru opdager Erika Byskov, at hun kun har et enkelt link, der matcher begge kategorier - og hvad værre er, den side, linket henviser til, har fået nyt layout. Hendes henvisning peger ikke længere på det relevante afsnit i teksten.

Desværre er der tale om en meget lang tekst, hvoraf kun det ene afsnit er relevant, så hun må selv læse hele teksten igennem og genoprette sin henvisning, før hun kan give sin kollega det efterspurgte materiale.

Som det ses af scenarierne, kan opdateringer, ændringer og flytninger af webindhold være kilde til store problemer for annotations-software og tredjeparts link-databaser, hvis ikke softwaren er robust nok til at kunne modstå ændringerne. I en ideel verden, ville Carl Sunesens annotationsværktøj selv have relokaliseret de annoterede tekststykker, så længe disse ikke var blevet helt slettet. Og Erika Byskovs linkdatabase-program havde haft en indbygget mekanisme, der automatisk kunne genfinde hendes kilde.

Eksisterende værktøjer

Der er udviklet en række værktøjer, der tilbyder nogle af de funktionaliteter, WWW ikke selv har inkorporeret. De tilbyder i forskellig grad annoteringer som noter, links, delte bogmærker, guidede tours m.m. Som eksempler kan

nævnes Arakne [2], Amaya[3], Diigo[4] og ShiftSpace[5].

Ingen af de nævnte, levende, systemer giver i udgangspunktet mulighed for at tilføje links, og et generelt problem for systemerne er håndteringen af dynamisk indhold. Hvordan håndterer man for eksempel highlight af - eller links til eller fra - et stykke tekst, der flytter sig på siden, ændrer sig, forsvinder eller flyttes til en anden side? Disse bevægelser er som nævnt hyppige på WWW, hvor blogs, sociale websteder og andre sider i stigende grad består af brugergenereret og -redigeret indhold.

Formål

På trods af potentialet i velfungerende software på området er der mig bekendt kun offentliggjort få forskningsartikler omhandlende metoder til relokalisering af dynamisk indhold. I dette speciale vil jeg evaluere de to fundne kilder, der behandler emnet, og som hver beskriver en metode til at udføre relokaliseringen. Desuden vil jeg fremstille mit eget bud på en løsning. Jeg vil:

1. opridse den videnskabelige baggrund området har i hypermedier
2. motivere nytten af og behovet for en effektiv metode til relokalisering
3. beskrive, implementere, evaluere og sammenligne eksisterende metoder
4. fremstille mit eget bud på en metode til relokalisering

Punkt et og to vil være af teoretisk karakter baseret på kilder om hypermedier, web-dynamik og brugen af annotationer. Punkt tre vil dels være en teorisk beskrivelse og analyse, dels en praktisk test udført med egne implementationer af metoderne. Mit eget bud på en metode vil også blive implementeret og sammenlignet i test med de øvrige metoder.

Til de praktiske test vil jeg udvikle en prototype på et framework, der kan bruge en eller flere forskellige metoder til relokalisering af fragmenter af dynamiske websider på samme tid. Det er forhåbningen, at sådan et framework på sigt med flere velprøvede, indbyrdes supplerende metoder kan give bedre resultater, end metoderne kan enkeltvist.

I kapitel 2 vil jeg gennemgå det relaterede arbejde, der er lavet omkring lokations-specificering i hypermedier og de bud på løsninger, der eksisterer

[3] `http://www.w3.org/Amaya/`, World Wide Wen Consortiums browser med nogen understøttelse for annotering

[4] `http://www.diigo.com/`, browser-plugin, der giver nogen understøttelse for annotering

[5] `http://www.shiftspace.org/`, tilføjer via GreaseMonkey plugin'et en vis understøttelse for annotering til Firefox

i dag, samt begrunde nyttigheden og kravene til LocSpecs i forhold til annotering af dynamisk web-indhold. Kapitel 3 præsenterer de metoder og værktøjer, jeg har brugt videre i specialet, Mozilla-platformen, det framework, jeg har udviklet, samt de test, jeg foretager. Selve implementationerne af LocSpecsne beskrives i kapitel 4, mens resultaterne af testene fremstilles i kapitel 5, hvor de også sammenlignes. Til slut samler jeg trådene i en konklusion og giver mine bud på, hvad der videre kan ske på området.

2

Teori og baggrund

I dette kapitel vil jeg motivere brugen af Location Specifiers samt beskrive noget af det arbejde, der tidligere er gjort på området med henblik på World Wide Web. Der vil først være en kort gennemgang af annotationer og hvilke krav disse stiller til at kunne lokalisere og relokalisere bidder af web-indhold. Dernæst vil jeg opsummere den historiske hypermediebaggrund for begrebet Location Specifiers samt foretage en vurdering af begrebets tilstrækkelighed i dette speciales kontekst. Slutteligt analyseres de tidligere indsatser inden for lokationsspecificering, der er mig bekendt, og som jeg senere i specialet vil implementere og teste.

2.1 Annotationer, links og web-dynamik

Der er mange både videnskabelige og ikke-videnskabelige aktiviteter, der forudsætter relokalisering af indhold i et medie. Det kan være en overstregning eller markering af en vigtig passage, så den let kan genfindes, det kan være tilføjelsen af en henvisning til relateret materiale eller en note i marginen med en kommentar til indholdet. Alle disse handlinger falder under betegnelsen annotering.

2.1.1 Annotationer

Annotering er en udbredt praksis ved læsning af akademiske tekster. Noter i margin, understregning, krydsreferencer og meget mere. Unsworth beskrev i 2000 annotering som en af syv videnskabelige grundsten, axiomer eller primitiver [13]. Syv handlinger, der danner grundlaget for videnskabeligt arbejde på tværs af fag og tid - omend han ikke udelukkede, der kunne være flere. De øvrige er *opdage, sammenligne, referere, tage prøver, illustrere* og *repræsentere*.

Som nævnt kan man skelne mellem forskellige slags annotationer. I "Toward an ecology of hypertext annotation" [9] har Marshall analyseret en større mængde annotationer foretaget på papir som en del af sin argumentation for, at annotationer hjælper til at forbedre hypermediers struktur. Hun identificerede fem typer annotationer, som hun klassificerede i hypertext-terminologi. De fem typer er: *referencer, fremhævning, kategorisering, segmentering* og *ankre*.

Referencer kan være en sammenkobling mellem en note og et stykke tekst eller mellem to stykker tekst. Fremhævning dækker over understregning eller

overstregning af vigtige passager i teksten. Kategorisering er markeringen af tekst-elementer for eksempel i forskellige farver, hvor hver farve er associeret med en kategori. Segmentering er opdelingen af teksten mindre stykker, der eksempelvis nummereres, og som eventuelt kan læses i en anden rækkefølge.

Den femte type - ankrene - er grundstenen for hver af de fire første typer annotationer. Ankre er markeringer af en bid af eller et punkt i teksten. Det punkt, hvor til eller fra en reference fører; det stykke, der er fremhævet eller kategoriseret; den bid, der udgør et segment; det punkt i teksten, hvor en note hører til. Dog understreger Marshall, at disse ankre ofte er vagt markerede, og at deres præcise placering i teksten ikke fremgår tydeligt.

"Vag" og "upræcis" går ikke ret godt hånd i hånd med digitale løsninger, hvor systemer er bygget med matematisk præcision, og hvor værdier enten er 0 eller 1, ikke cirka eller en mellemting. Derfor må ankre i digitale dokumenter specificere en nøjagtig lokation ud fra dokumentets struktur og indhold. Men der er også en anden udfordring i forbindelse med hypermedier generelt og med World Wide Web i særdeleshed i forhold til tekster på papir, nemlig at indholdet forandres over tid.

2.1.2 Dynamik på World Wide Web

Der er foretaget et væld af undersøgelser over frekvensen af ændringer eller opdateringer på websider over tid. Sigtet med undersøgelserne varierer, men for manges vedkommende er målet at understøtte web-crawleres arbejde med at holde søgemaskinernes indici opdaterede.

En sådan undersøgelse er foretaget af Adar *et al.* i 2009 [1]. De har monitoreret 55.000 websider over en periode på fem uger. Siderne blev udvalgt på baggrund af en stribe kriterier, hvor det vigtigste var, at det var sider, der var blevet besøgt mindst to gange af samme person. 66 % af siderne i undersøgelsen ændrede sig mindst en gang i løbet af de 5 uger, og gennemsnitligt ændrede en side sig for hver 123 timer. Næsten 42 % ændrede sig mindst en gang i timen. Undersøgelsen viser altså, at de websider, der bliver besøgt gentagne gange, tilsyneladende er meget dynamiske.

Koehler foretog i perioden december 1996 til februar 2001 en ugentlig monitorering af 361 tilfældigt omend repræsentativt udvalgte World Wide Web adresser [8]. I løbet af de godt fire år overvågningen forløb, overlevede cirka en tredjedel af siderne, men allerede efter halvandet år, var 50 % af siderne forsvundet. De sidste 12 måneder af forløbet var tallet af levende sider næsten konstant. Koehler undersøgte ikke, hvorvidt indholdet var flyttet til en anden adresse, men konstanterede blot, at han ikke længere fik svar, når han forsøgte at tilgå siden.

Koehlers undersøgelsen viste dog også, at der ugentligt sker ændringer på

omtrent 20 % af websiderne, omend tallet falder lidt for de ældre, sejlivede sider. At dette tal er lavere end Adar *et al.*'s tal, skyldes formentlig især udvælgelseskriterierne, men måske også i nogen grad den udvikling i retning af mere dynamisk indhold på nettet, der er sket de gennem de otte år, der er forløbet mellem undersøgelserne.

At websider for en betragtelig dels vedkommende ændrer sig over tid er altså et veletableret faktum. Hvad enten gammelt indhold fjernes til fordel for nyt som på en blog-forside, der viser de tre nyeste posts, eller det blot bliver redigeret, som for eksempel en rettelse i en wiki, så er det en udfordring for annotations-software. Skal annotationen vises, hvor den blev lavet, følge med teksten, helt fjernes eller gemmes i bunden af siden?

2.1.3 Forventninger til digitale annotationer

Brush *et al.* har i "Robust Annotation Positioning in Digital Documents" [4] undersøgt, hvordan brugerne forventer, annotationerne bliver positioneret, når et dokument er blevet redigeret. De skelner mellem fire forskellige typer ændringer: *sletning, omformulering, anker-flytning* og *paragraf-flytning*. Sletning forekommer, når det annoterede tekststykke helt slettes, og omformulering forekommer, når teksten bliver redigeret i større eller mindre grad. Anker-flytning er flytning af hele den annoterede tekst inden for den paragraf, den forekommer i - eller til en anden paragraf, mens paragraf-flytning dækker over flytning af hele den paragraf, den annoterede tekst forekommer i til et andet sted i dokumentet.

De har foretaget to undersøgelser, hvoraf kun den anden gav egentligt interessante resultater. I den interessante undersøgelse, bad man tolv brugere annotere et dokument, som efterfølgende blev udsat for en stribe ændringer. Brugerne blev så bedt om at evaluere en simpel algoritmes[1] relokalisering af deres annotationer i den nye udgave af dokumentet. Tilfredsheden var stor i de tilfælde, hvor algoritmen korrekt placerede annotationer, hvor der ingen eller kun små ændringer havde været i selve ankerteksten - altså flytnings-typerne nævnt ovenfor. Forfatterne konkluderede på baggrund af dette, at den tekst, der ligger omkring et anker, er af mindre betydning for annotationen i brugerens øjne.

Der var også stor tilfredshed, når algoritmen udelod[2] annoteringer af slettet tekst. Den største utilfredshed opstod, når algoritmen ikke genfandt

[1] Den benyttede algoritme søgte efter anker-teksten. Blev den ikke fundet, kappede algoritmen et ord af teksten i hver ende og forsøgte igen. Den fortsatte således til anker-teksten var kortere end 15 tegn. Herefter blev ankeret bestragtet som ikke fundet.

[2] Udeladte annotationer præsenteredes som "forældreløse" i en menu i venstre side af skærmen.

anker-tekst, der kun var let redigeret.

Generelt var tilfredsheden lav for komplekse ændringer. Jo mere kompleks ændringen var, jo større var forståelsen for en udeladelse, men jo mindre var tilfredsheden med de eventuelle relokaliseringer. Det sidste skyldes antageligt, at placeringen af annotationen var forkert, eller at ændringen af teksten var så stor, at annotationen ikke længere gav mening. Brush *et al.* foreslår, at ændringerne i en tekst kan nå et punkt, hvor en annotation hellere skal udelades, end knyttes til en position, der ellers med stor sandsynlighed er den korrekte.

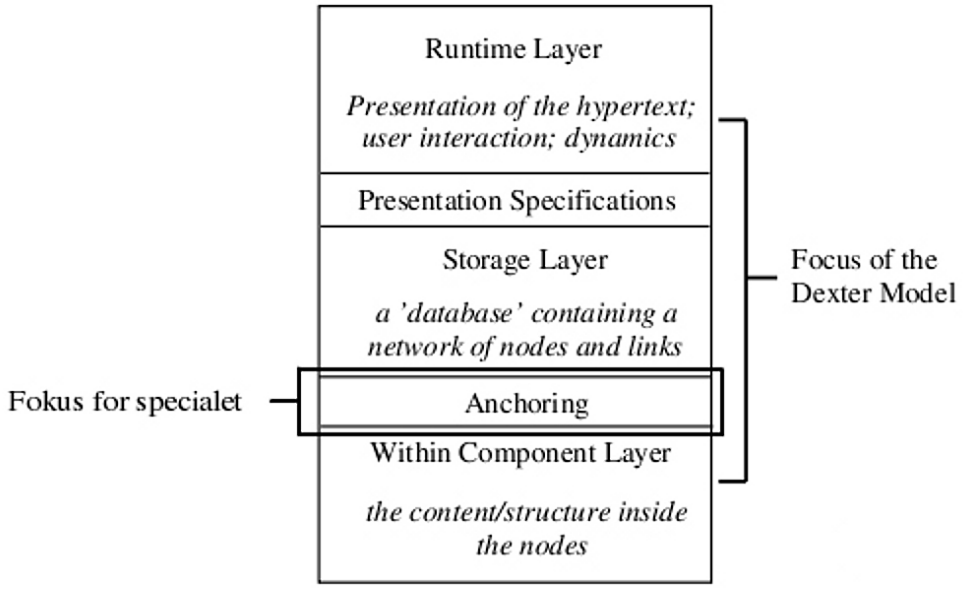
Udtalelser fra brugerne under undersøgelsen antyder, at det kan være en mulighed at kigge på brugen af nøgleord. I de tilfælde, hvor nøgleordene i en sætning er forblevet gennem en ændring, men måske er flyttet rundt, har den simple algoritme udeladt annotationen, hvor brugeren klart forventede, den burde kunne genkende sætningen. Det viste sig også, at brugerne forventede korrekte relokaliseringer af citater og navne i teksten.

Brugernes forventning er altså ikke altid at få annotationen vist, hvor den mest sandsynligt hører til. Der kan være tilfælde, hvor ændringen har en karakter, der gør annotationen irrelevant eller forældet. Sammenholdt med de øvrige problemstillinger fra dette kapitel, nemlig brugernes generelt vage specificering af annotationers start- og slutpositioner, og World Wide Webs dynamiske natur gør det ikke opgaven lettere at udvikle software til automatisk relokalisering af annotationer.

I dette speciale vil jeg dog ikke specifikt kigge på, hvornår en ændring er så stor, at en annotation bør udelades. I stedet er fokus på, hvor ofte det er muligt korrekt at relokalisere et anker. Til det fokus bidrager Brush *et al.* primært med deres pointe om brug af et sæt af nøgleord frem for en fast tekststreng samt konklusionen, at den omkringliggende tekst tilsyneladende er mindre vigtig for slutbrugeren og derfor højest bør være af sekundær betydning i en eventuel algoritme.

2.2 Hypermedier og lokationer

Annotationer, links og ankre i digitale dokumenter har deres baggrund i hypermediesystemerne, og det er herfra, terminologien omkring algoritmerne stammer. Inden jeg kigger nærmere på de eksisterende algoritmer for relokalisering, vil jeg derfor præsentere lidt af baggrunden på området. Vi starter i 1988 på Dexter's Inn i New Hampshire.



Figur 2.1: Lagene i dextermodellen, modificeret udgave af Figur 1 fra The Dexter Hypertext Reference Model [6, s. 4]

2.2.1 Dexter-modellen

Hypermediesystemer kan helt basalt defineres som systemer, hvor relaterede informationer af forskellig karakter, for eksempel tekst og billeder, forbindes og kan præsenteres i sammenhæng. Mere teknisk, men stadig helt generelt, er hypermediesystemer beskrevet i Dexter-modellen [6], hvor de deles op i tre lag: Kørsels-laget (*Runtime layer*), Lager-laget (*Storage layer*) og i-komponent-laget (*Within-component layer*) - se Figur 2.1.

Kørsels-laget beskriver, hvordan systemet præsenterer data og håndterer brugerinteraktion. Lager-laget detaljerer samspillet mellem indholdskomponenter og links i systemet. I-komponent-laget dækker over strukturen internt i komponenter. Dexter-modellen behandler primært lager-laget og henviser til, at listen af komponent-typer ikke er endelig, hvilket hindrer en generel beskrivelse som sådan. I stedet henvises i artiklen til andre referencemodeller, der kan bruges komplementerende til Dexter-modellen. Det samme gør sig i nogen grad gældende for kørsels-laget.

Det interessante for dette speciale er interfacet mellem lager-laget og i-komponent-laget samt link-systemet i lager-laget. Dexter-modellen beskriver links som opbygget af to eller flere *endpoint specifiers*, hvor en specifier er et tupel af et anker-id, en komponentspecifikation og link-retningsinformationer. Komponentspecifikationen kan være et unikt komponent-id eller et udtryk,

der kan evalueres til et komponent-id. Anker-id'et refererer unikt til et anker i den specificerede komponent. Ankeret består udover sit id af en værdi, der kan beskrive en lokation eller region i komponenten.

Ankeret ligger altså i i-komponent-laget - og beskrives som en del af komponenten sammen med for eksempel præsentationsspecifikationer, mens endpoint specifierne ligger i lager-laget sammen med linksne. En central pointe i Dexter-modellen er præcis at separere strukturen og indholdet i komponenterne fra den overordnede stuktur af komponenter og link mellem disse. I dette speciale ønsker jeg netop at kunne specificere en lokation i en komponent, der ikke nødvendigvis er markeret som anker i komponenten. Selvom en sådan specificering med fordel kan tvedeles i en komponent-specifikation og en i-komponent-specifikation, og disse to dele kan kategoriseres som værende i hver sit lag, forekommer Dexter-modellen ikke helt tilstrækkelig til at beskrive et scenarie, hvor ankrene ikke er en del af komponenten. Specifikt i forbindelse med ændring af en komponent skriver Halasz og Schwartz:

> As a component changes over time (e.g., when it is edited within the runtime layer), the within-component application will change the anchor value to reflect changes over the internal structure of the component or to reflect within component movement of the point, region or items to which the anchor is conceptually attached. (The Dexter Hypertext Reference Model [6], s. 9)

En sådan opdatering af ankre er ikke mulig i World Wide Web scenariet med eksterne, tredjeparts-ankre; at opnå fungerende, eksterne ankre uden denne manglende opdatering er essensen af problemstillingen i dette speciale.

2.2.2 Samling af indlejrede referencer og linkobjekter

Den konceptuelle løsning på problemet om eksterne, uafhængige ankre findes i en model udviklet af Grønbæk og Trigg, der i artiklen "Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects" [5] har modificeret Dexter-modellen til bedre at kunne beskrive hypermediesystemer med indlejrede links og dynamisk indhold, som for eksempel World Wide Web, men også til at understøtte tredjeparts link-databaser.

I forhold til dette speciale er udviklingen af begreberne *Location Specifier* (*LocSpec*) og *Component Location Specifier* (*CompLocSpec*) særlig relevant. En LocSpec er en beskrivelse af en lokation i en komponent, og den svarer således nogenlunde til anker-værdien i Dexter-modellen. En CompLocSpec

er en udvidelse af en LocSpec, der både angiver en komponent i hypermediesystemet og en lokation i denne komponent. På denne måde svarer en CompLocSpec nogenlunde til en endpoint-specifier i Dexter-modellen, med den forskel, at en endpoint-specifier henviser til et anker i en komponent, for at udpege i-komponent-lokationen, hvorimod CompLocSpec'en har denne information indbygget.

LocSpecs er opbygget af tre *content locator attributes*: Et *object id*, en *structure descriptor* og en *computation specification*. Disse tre behøver ikke alle være udfyldt i hver konkret LocSpec-instans, men alle udfyldte attributter skal være opfyldt, før en lokation er specificeret af LocSpec'en. I HTML-sammenhæng kan objekt-id'et eksempelvis være id-tagget på et element på siden; structure descriptor'en kan være et XPath-udtryk; og en computation specification kunne være en søgning efter en tekst-streng på siden.

CompLocSpecs er bygget op af de samme tre content locator attributes, og derudover yderligere tre *component locator attributes*, der er helt parallelle til indholds-attributterne: *Component id*, *component structure descriptor* og *component computation spec*.

Den store adskillelse fra Dexter-modellen er som nævnt, at hverken LocSpec eller CompLocSpec er en del af eller direkte knyttet til en komponent i systemet. De er selvstændige enheder, der refererer til lokationer i systemet - og kan således oprettes og gemmes eksternt af tredjeparts-software, af brugere, der ikke har skriverettigheder til det dokument eller den komponent, der refereres. Denne uafhængighed af forfatteren, sammen med mulighederne for fleksibelt at specificere lokationerne, gør, at begrebet LocSpecs beskriver præcis det værktøj, der skal bruges for at kunne specificere robuste lokationer på det dynamiske World Wide Web.

2.3 World Wide Web-standarder

World Wide Web Consortium, W3C[3], er en international organisation, der arbejder for at udvikle de World Wide Web standarder, der danner grundlaget for meget af kommunikationen på internettet i dag, for eksempel HTML, CSS og XML.[4]

[3] `http://www.w3c.org`

[4] HTML står for Hyper Text Markup Language, det sprog, der danner grundlaget for hovedparten af siderne på World Wide Web. CSS står for Cascading Style Sheet, den teknik der oftest bruges til at definere stil og udseende af elementer i HTML dokumenter. XML er en forkortelse af Extensible Markup Language, der ofte bruges til at specificere data i et format, der kan forstås af applikationer på tværs af systemer. På `http://www.w3.org/TR/`

I det følgende vil tre af W3Cs standarder kort blive præsenteret, da de spiller en central rolle i mit arbejde med implementationer af metoderne til relokalisering. De tre er DOM, XPath og XPointer.

2.3.1 DOM

DOM[5] står for Document Object Model. Det er en måde at beskrive indhold, struktur og stil af elementer i to af W3Cs andre standarder: HTML og XML. Disse dokumenttyper er strukturerede - forstået som opbygget i et hierarki af elementer med et enkelt rod-element, der kan have et eller flere under-elementer kaldet børn, der hver igen kan have børn og så fremdeles.

DOM-træet

DOM'en beskriver denne interne logiske sammenhæng i dokumenterne, der ofte illustreres som et træ. Endvidere beskriver DOM-standarden en række metoder til navigation af DOM-træet[6]. For eksempel kan man kalde parentNode, firstChild og nextSibling på et element.

I Figur 2.2 ses DOM-træet for det XML-dokument, der er vist i Oversigt 2.1. I dette eksempel er root-elementet parentNode for de to child-elementer, der har id 1 og 2, og de er henholdsvis previous- og nextSibling for hinanden. Det child-element, der har id 1.1, er firstChild for det child-element, der har id 1.

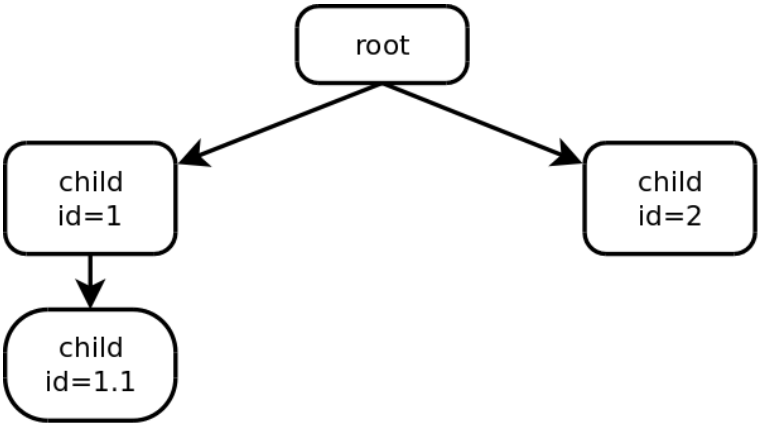
```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <root>
3   <child id='1'>
4     <child id='1.1'>
5     </child>
6   </child>
7   <child id='2'>
8   </child>
9 </root>
```

Oversigt 2.1: Eksempel på XML-dokument

findes en komplet liste over W3Cs anbefalede teknikker

[5] `http://www.w3.org/TR/DOM-Level-3-Core/` - den nyeste tilføjelse til DOM-kernen. DOM specifikationen består af en stribe specifikationer, som det ses her: `http://www.w3.org/TR/#tr_DOM`

[6] Særskilt specificeret i `http://www.w3.org/TR/ElementTraversal/`



Figur 2.2: DOM-træet for XML-dokumentet i Oversigt 2.1

Udvalgte DOM-elementer

DOM-specifikationen er temmelig omfattende, så en dybdegående kortlægning af mulighederne forbundet med brug af DOM'en er ikke mulig i denne sammenhæng. I stedet vil jeg beskrive tre af de dele, som har størst betydning for implementationerne, der bliver beskrevet i kapitel 4.

DOM-Dokument repræsenterer som nævnt et helt dokument. Det er samtidig rod-elementet i DOM-træet og dermed udgangspunktet for stier gennem træet til blad-elementer. Vigtige metoder i forhold til dette speciale er `getElementById()`, der returnerer alle elementer i dokumentet med det givne id, `createRange()`, der oprettet et DOM-range i dokumentet og `evaluate()`, der givet et XPath-udtryk (se afsnit 2.3.2) returnerer alle elementer specificeret af dette udtryk.

DOM-Range er et span i dokumentet, der strækker sig fra et offset ind i et start-element til et offset ind i et slut-element. Udover rollen som repræsentation af et stykke af dokumentet har DOM-ranget en vigtig metode kaldet `compareBoundaryPoints()`, der kan bruges til at sammenligne to ranges begyndelses- og slut-lokationer. `toString()` metoden på et DOM-range returnerer det tekststykke i dokumentet, som ranget spænder over.

DOM-Node er et interface, som alle elementer i DOM-træet implementerer. Det betyder, at man for eksempel kan sammenligne to elementer ved at kalde `elem1.isSameNode(elem2)`, hvilket er ganske nyttigt, når man skal bygge en sti fra roden til et specifikt element. Det betyder også, at de metoder til navigation af DOM-træet, der blev nævnt indledningsvist - for eksempel parentNode - findes på alle elementer, hvilket gør det noget nemmere at

iterere igennem træet. I fald et element ikke har et forældre-element, altså hvis det selv er rod-element, vil parentNode være null.

DOM i specialet

Som antydet spiller DOM-træet og navigering af det en helt central rolle for implementationerne i dette speciale. Som det bliver beskrevet i afsnit 3.2, vil jeg benytte Mozillas implementation af DOM-specifikationen.

DOM'en danner også grundlaget for XPath og dermed XPointer, som jeg beskriver herefter. Disse to sprog er designet til at specificere lokationer i DOM-træer.

2.3.2 XPath

XPath, eller XML Path Language[7], beskriver stier gennem DOM-træer. Det er interessant i forbindelse med dette speciale af to årsager. Dels er det i sig selv en metode til at specificere lokationer, dels danner det basis for en af de to metoder, jeg analyserer nærmere senere i specialet.

Som lokationsbeskrivelse i sig selv er XPath designet til at beskrive elementer i statiske dokumenter. På trods af dette har det visse træk, der gør det robust over for ændringer i dokumenterne. XPath er nemlig i høj grad uafhængig af indholdet i dokumentet, da det er tæt knyttet til dokumentets struktur. Et par eksempler er på sin plads og kan ses i Oversigt 2.2.

Alle tre XPath-udtryk beskriver et enkelt element fra Figur 2.2. Første udtryk beskriver stien via første root-element til andet child-element. Andet eksempel beskriver alle de child-elementer, der er børn af et child-element, der er barn af et root-element, der er rod i dokumentet. Det tredje udtryk er lidt anderledes. Det beskriver alle elementer i træet, der har en attribut med navnet id, og hvor værdien af attributten er 1.1.

Eksempel to og tre beskriver altså det samme element fra eksemplet i Figur 2.2 - nemlig det nederste barn. Det første eksempel beskriver child-elementet med id 2.

```
1 /root[1]/child[2]
2 /root/child/child
3 //*[id='1.1']
```

Oversigt 2.2: Eksempler på XPath-udtryk

Der er dog intet i vejen for, at et XPath-udtryk kan beskrive flere elementer i træet: Havde child-elementet med id 1.1 haft en søskende, eller

[7]Specificeret i `http://www.w3.org/TR/xpath/`

havde child-elementet med id 2 haft et barn, var begge disse også blevet beskrevet af det andet XPath-udtryk i Oversigt 2.2.

Som absolutte stier fra rod til lokation (formen set i det første XPath-eksempel), er XPath-udtryk upåvirkelige af ændringer i senere søskende og i undertræer til tidligere søskende. Kun ændringer direkte i stien eller i tidligere søskende påvirker udtrykket. Derfor er XPath et godt udgangspunkt for Paz og Díaz' arbejde i Resilient XPaths, som bliver beskrevet i afsnit 2.5.2 - og da Phelps og Wilenskys tree-Walk-metode, som beskrives i afsnit 2.5.1, også angiver absolutte stier i et træ, deler deres metode denne robusthed.

Hvad XPaths dog mangler, er specifikation af lokationer ind i elementer. Til dette formål har W3C udviklet XPointer.

2.3.3 XPointer

XPointer frameworket[8] med xpointer() scheme[9] er et værktøj designet specielt til at pin-pointe lokationer ind i tekst-elementer i DOM-træer. I dette speciale bruger jeg xpointere fra xpointer() scheme til at beskrive lokationer i statiske dokumenter. Det vil sige til at udpege de lokationer i et dokument, jeg ønsker at genfinde i et senere dokument - samt de lokationer i sådanne indsamlede, senere dokumenter, som jeg anser som korrekte relokaliseringer.

XPointer med xpointer() er meget alsidigt og omfattende, så ligesom for DOM-specifikationen og XPath, beskriver jeg her kun et lille subset bestående af tre funktioner:

`string-range()` kaldes med argumenterne: *location-set*, *string*, *number* og *number*. *Location-set* angiver de elementer, hvori `string-range` skal lede efter tekst-strengen *string*. De to *number*-parametre angiver henholdvis hvor mange pladser fra begyndelsen af den eventuelt fundne tekst-streng, og hvor mange bogstaver det returnerede range skal bestå af. Eksempel:

```
string-range(//child, 'barn', 1, 4)
```

Dette eksempel vil returnere et sæt af ranges, bestående af alle forekomster af ordet "barn" i child-elementer. Benyttes en tom tekst-streng, finder funktionen alle positioner i elementerne fra *location-set*.

`start-point()` kaldes med et enkelt argument af typen *location-set*. Funktionen returnerer et nyt *location-set* bestående alene af start-punkterne for hver lokation i argumentet.

[8] `http://www.w3.org/TR/xptr-framework/` - XML Pointer Language Framework
[9] `http://www.w3.org/TR/xptr-xpointer/`

```
start-point(string-range(//child, 'barn', 1, 4))
```

Dette eksempel returnerer et sæt af lokationer bestående af hvert punkt, der ligger lige før bogstavet b i barn, i alle de forekomster, som `string-range`-funtionen fra forrige eksempel fandt.

`range-to()` kaldes i forhold til en lokation - kontekst-lokationen, og med et sæt af lokationer som argument. Argument-sættet skal dog indeholde præcis ét punkt, der vil blive brugt som slut-punkt for ranget. `Range-to` vil returnere et range fra kontekst-lokationen til slut-punktet.

```
start-point(string-range(//child, 'barn', 1, 4)[1])
/range-to(end-point(string-range(//child, 'barn', 1, 4)[2]))
```

Dette eksempel vil returnere ranget fra starten af den første forekomst af ordet barn i et child-element til slutningen af den anden forekomst af ordet barn i et child-element. `End-point` fungerer efter samme princip som start-point - blot findes slut-punkter i stedet for start-punkter.

xpointer() scheme indeholder flere end disse funktioner, men de nævnte har været mest centrale i forbindelse med dette speciale.

2.3.4 Konklusion

W3C-standarderne spiller en stor rolle for World Wide Web, og alt hvad der foregår der. Enhver brug af WWW må forholde sig til disse standarder og ofte også inddrage dem. DOM, XPath og XPointer er tre af deres specifikationer, der ikke har været til at komme uden om i arbejdet med dette speciale, som det vil fremgå af de følgende afsnit og kapitler.

Afsnittene om W3Cs standarder er blot korte introduktioner til teknologierne og er på ingen måde fuldstændige. Ej heller dækker introduktionerne nødvendigvis de vigtigste elementer - derimod blot de elementer, der er vigtigst i forhold til brugen i dette speciale.

Resten af kapitlet graver ned i to metoder til lokationsspecifikation, og de har begge har relationer til W3Cs DOM og XPath.

2.4 LocSpecs og World Wide Web

I indledningen blev nævnt en lang række applikationer, der på den ene eller den anden måde lokaliserer bidder af indhold på websider. På trods af udbuddet af applikationer er det temmelig begrænset, hvor meget forskningsmateriale, der er publiceret om lokationsspecifikationer for World Wide

Web. I de følgende afsnit vil jeg præsentere de to kilder, jeg har kendskab til, på området. Det vil være de metoder, der er beskrevet i disse kilder, jeg senere i specialet implementerer og evaluerer.

2.4.1 Inter- og intra-dokument-ændringer

Hvis web-indhold bliver flyttet fra et sted på en side til et andet, eller hvis det bliver redigeret, betragtes ændringen som en *intra-dokument* ændring. Hvis det derimod bliver flyttet til en anden side, er der i stedet tale om en *inter-dokument* ændring. I tilfælde af inter-dokument ændringer vil man først skulle lokalisere ankerets nye side og efterfølgende benytte en intra-dokument metode til at lokalisere ankeret på den pågældende side.

Der er altså ikke tale om gensidigt udelukkende metoder, men om supplerende metoder, hvor man kan forestille sig, at man typisk først forsøger at lokalisere ankeret på den oprindelige side ved hjælp af en intra-dokument metode. Hvis det ikke lykkes, kan man forsøge at relokalisere ankeret ved hjælp af en inter-dokument metode kombineret med en intra-dokument metode. I dette speciale er fokus på intra-dokument-ændringer.

2.5 Intra-dokument lokalisering

Selv uden at betragte inter-dokument ændringer, er der væsentligt forskellige tilgange til problemstillingen. En kilde specificerer på element-niveau, mens en anden beskriver, hvordan man kan lokalisere helt ned på bogstav-niveau. Generelt er det sidstnævnte ønskværdigt, men hvis det første fungerer godt, kan det bruges supplerende.

2.5.1 Robuste lokationer

Phelps og Wilensky beskriver i Robust Intra-Document Locations [12] fra 2000, både forskellige konkrete lokaliseringsmetoder og en overordnet strategi for lokalisering. De opsætter desuden en række *robusthedskriterier*, der i deres øjne skal opfyldes for at have opnået en lokaliseringsalgoritme, der er robust i et dynamisk miljø som World Wide Web. Krav som *modstandsdygtighed overfor almindelige ændringer*; *baseret på dokumentindhold* (indhold og/eller struktur - i modsætning til geometrisk lokation: "noten er placeret på position x,y"); *fungere med ikke-samarbejdende servere*; *udvideligt til multimedier* og *fungere med eksisterende dokumenter og servere* giver god mening som krav, hvorimod deres kriterier om *ligefrem implementation*; *relativt lille* og

gradvist aftagende præcision ved stigende ændringer af dokumentet nok er ønskværdige egenskaber, men måske ikke er egnede som generelle krav.

Den overordnede strategi

Den overordnede strategi formulerer, hvordan man kan sammensætte forskellige metoder til et samlet system. Metoderne prioriteres og benyttes efter tur. Det forudsættes også, at metoderne udover at give et bud på en lokation giver en vægtning af, hvor sandsynligt deres resultat er. I Robust Intra-Document Locations [12] benyttes metoderne enkeltvist, og kun hvis den ene ikke finder et brugbart resultat forsøges den anden. Vægtningen bruges således ikke til at prioritere mellem flere resultater, og metoderne bruges ikke til at understøtte hinandens resultater; begge dele kunne ellers være interessant. I stedet bruges vægtningen af resultatet til at vurdere, hvordan det skal vises for brugeren af systemet, hvilket da også er en fornuftig plan, som beskrevet i afsnit 2.1.3, "Forventninger til digitale annotationer". Phelps og Wilensky har udarbejdet tre basis-metoder, men de skriver, at det er tænkt som et udgangspunkt, der både kan udvides og ændres, og at det ikke er en optimal algoritme.

De tre basismetoder består hver af en *location descriptor*-del, som er de data, der beskriver lokationen, og en algoritme-del, der udfører lokaliseringen ud fra beskrivelsen. Hver af de tre metoder kan i princippet betragtes som en selvstændig LocSpec, men det er bemærkelsesværdigt, hvordan de tre valgte metoder kan passes ind i hver sin af de tre *content locator attributes*, LocSpecs består af (se afsnit 2.2.2, Samling af indlejrede referencer og linkobjekter).

Første metode: Unikt id

Lokationsbeskrivelsen

Lokationsbeskrivelsen er her blot et unikt id for et givent element.

Eksempel:

`Ekstern_henvisning` beskriver præcis det element, hvis id er "Ekstern_henvisning".

Lokaliseringsalgoritmen

Lokaliseringsalgoritmen søger efter det angivne id i dokumentet. Hvis det findes, betragtes lokationen som fundet. Hvis det ikke findes, må der forsøges med en anden metode.

Styrker og svagheder

Den mest markante fordel er, at et unikt id ifølge forfatterne er sejlivet og oftest overlever de fleste ændringer, bortset fra sin egen sletning.

Ulemper er, at der kun kan beskrives lokationer, der falder sammen med elementer, samt at disse id'er kun forefindes, hvis dokumentets forfatter har sørget for det. Endvidere baseres brugen af id på den antagelse, at id'et er kædet sammen med elementets indhold. Det kan resultere i forkerte lokationer, hvis en dokumentforfatter ikke overholder denne antagelse, eksempelvis hvis en ny indføring i en weblog altid overtager id'et "newest-entry".

Anden metode: Tree-walk

Lokationsbeskrivelsen

Lokationsbeskrivelsen i tree-walk-metoden er en sti i noget forfatterne kalder *SGDOM*: simple, general document object model. Det er en forenkling af W3Cs DOM, hvor de interne knuder har type efter de elementer de repræsenterer, og hvor bladene er medie-elementer, for eksempel ord i tekst, frames i filmklip eller sekunder i musikstykker - elementer, der giver mening i forhold til mediet, og som kan indekseres med et tal. Objekter, der tilføjer attributter eller egenskaber til teksten fremfor at være strukturerende, sies fra. I HTML kunne sådanne objekter være `span`, `a`, `i` og lignende. Stien beregnes fra dokumentes rod-element. Sammen med hvert elements type registreres også et indeks-tal, over hvilket nummer barn elementet er. Forfatterne har valgt, at stien har rod-elementet længst til højre.

Eksempel:

`5/afdeling/2 0/<TEXT> 7/P 3/DIV 2/DIV 1/BODY 0/HTML`.

Som nævnt læses stien fra højre. Første skridt er dokumentets første barn, `HTML`. Herefter `HTML`s andet barn, `BODY`. Så `BODY`s tredje barn, `DIV` - og så fremdeles. `0/<TEXT>` angiver, at vi er nået til et medie-element af typen text, og at det er `P`s første barn. `5/afdeling/2` fortæller os, at vi leder efter det sjette ord, og at ordet er "afdeling". To-tallet specificerer, at den nøjagtige lokation, vi leder efter, er mellem f'et og d'et.

Lokaliseringsalgoritmen

Tree-walk-metodens lokaliseringsalgoritme eksekveres ved, trin for trin, at evaluere stien. Dette gøres ved at finde typen på det indekserede barn og

sammenholde den med den registrerede type i stien. Hvis alle trin matcher, betragtes lokationen som fundet i det element, hvor stien slutter. Hvis der derimod er et mismatch, vil algoritmen afprøve alternative stier, hvor der ændres på et enkelt led af gangen, efter et specifikt mønster[10].

Hvis ikke dette giver et resultat, må en anden metode prøves. Selv hvis disse ændringer giver et resultat, så vil antallet og størrelsen af ændringerne i stien, påvirke den vægtning algoritmen tilskriver resultatet. Hvis den specificerede lokation flyttes langt væk, eller der foretages en større ændring i dokumentets struktur, er der derfor en vis risiko for, at et eventuelt resultat forkastes som værende for usandsynligt.

Styrker og svagheder:

Tree-walk-metoden er i modsætning til unikt id-metoden uafhængig af dokument-forfatteren. Den er modstandsdygtig over for alle ændringer i undertræer i børn, der ikke er en direkte del af stien, samt ikke-strukturelle ændringer. Endvidere er den modstandsdygtig over for sletninger og tilføjelser af efterfølgende søskende til elementer i stien. Hvis der opstår et mismatch mellem elementets indeks og type, afprøves alternative stier.

En uheldig bivirkning ved metoden er, at der kan opnås et falskt positivt resultat, hvis en redigering af dokumentet betyder, at et element i stien erstattes af et element af samme type. Problematikken bliver dog kun udtalt, hvis det nye elements undertræ matcher resten af stien - en situation forfatterne beskriver som usandsynlig.

Tree-walk-metoden som sådan er strengt taget kun tænkt til at navigere frem til medie-elementet, hvorfra en mediespecifik metode må tage over. Måske er det årsagen til, at Phelps og Wilenskys bud på indeksering ind i tekst-medie-elementer har den svaghed, at hvis præcis det ord, lokationen ligger i, bliver ændret eller slettet, vil lokationen ikke kunne genfindes. En udbedring er dog ikke triviel, da resten af tree-walk-metoden kun vil kigge i andre undertræer efter en eventuelt flyttet lokation, hvis medie-metoden ikke accepterer det "bedste alternativ".

Sat på spidsen reducerer denne sidste svaghed tree-walk-metoden til en avanceret søgealgoritme, der leder efter præcis det ord, den oprindelige lokation befandt sig i.

[10]Først afprøves efterfølgende og foregående søskende, nærmeste først. Dernæst springes et trin over, hvilket ville svare til, der var indsat et ekstra strukturelt niveau i dokumentet. Slutteligt springes det aktuelle index/type-par over, hvilket ville svare til, der var fjernet et strukturelt niveau.

Tredje metode: Kontekst

Lokationsbeskrivelsen

Lokationsbeskrivelsen er en tekst-streng bestående af en lille bid af det foregående og efterføglende indhold, hvor delene er adskilt af et mellemrum. Mellemrum og andre specialtegn, der er en del af konteksten, kodes efter samme forskrift som i URI'er. Hvor meget kontekst, der skal gemmes, kan varieres, men Phelps og Wilensky benytter selv omkring 25 tegn.

Eksempel:

`tut+var+oprindeligt+en+af deling%2C+der+hørte+under`,

hvor %2C er kodningen af ,-tegnet, og hvor et afsluttende mellemrum er kortet af.

Lokaliseringsalgoritmen

Kontekst-metodens lokaliseringsalgoritme fungerer ved at søge efter kontekst-strengene i dokumentet. Hvis det findes flere steder, foretrækkes den lokation, der er nærmest den oprindelige position. Hvis strengene ikke findes, afkortes den forudgående kontekst-streng med det første ord, og den efterfølgende kontekst-streng afkortes med det sidste ord, hvorefter der søges igen. Der søges i første omgang kun i undertræet for den knude, hvortil stien i tree-walk-metoden passede, hvilket kan reducere den tekst, der skal gennemsøges betragteligt. Findes der intet match, forsøges samme fremgangsmåde fra knuden ovenover i træet - og så fremdeles.

Både afkortning af kontekst-strengene og forøgelse af det afsøgte undertræ bidrager til en reduceret vægtning af et eventuelt match.

Styrker og svagheder

Da beskrivelsen af lokationen med denne metode består af to stykker tekst, er metoden helt ufølsom over for strukturelle ændringer i dokumentet, så længe teksten umiddelbart omkring den gemte lokation ikke bliver delt op af ændringerne. Til gengæld er metoden følsom over for ændringer i den tekst, der bruges i beskrivelsen. Et enkelt ændret ord, for eksempel en rettelse i en stavefejl, i et ord lige op ad lokationen, vil forårsage, at metoden ikke finder lokationen, ligesom en ombytning af to ord vil betyde, lokationen ikke findes.

Ord for ord-afkortning af de gemte kontekst-bidder betyder en øget modstandsdygtighed mod sådanne ændringer - men øger risikoen for at finde

en forkert lokation. En generel svaghed ved metoden er, at den indebærer en vis risiko for at finde en kopi af teksten et andet sted i dokumentet.

Forfatterne foreslår selv tilnærmet matching som en måde at forbedre robustheden i metoden. Deltagerne i undersøgelsen beskrevet i Forventninger til digitale annotationer, afsnit 2.1.3, foreslog at bruge nøgleord fra den omkringliggende tekst i stedet for at bruge den eksakte tekst-streng, hvilket vil give fuld robusthed mod ændret rækkefølge af ordene og også mindske følsomheden over for ændringer i enkelte ord helt tæt på lokationen.

Forfatternes resultater

Phelps og Wilensky fremhæver i deres oprindelige artikel fra 2000, at deres system stadig mangler grundig testning:

> Our Multivalent client is a prototype, so such studies have not yet been undertaken. We offer the following relevant pieces of evidence that the algorithm will work well in practice, acknowledging that they are no substitute for actual meassurement. (Robust Intra-Document Locations [12], s 113)

Det har ikke været mig muligt at finde nogle senere, mere udførlige test-resultater.

De foreløbige resultater, de beskriver i artiklen, er fra tre tests: Dels fra UNIX manual-siderne (`man`), dels fra et simpelt forsøg, hvor de forsøgte at fremprovokere en situation, hvor en lokation ikke kunne genfindes, og slutteligt fra en test af en enkelt web side.

I det første tilfælde lykkedes det den samlede pakke at genfinde 742 af 754 annotationer, der var ændret, hvoraf flere af de resterende 12 var lokationer, der var blevet fjernet fra man-siden. En imponerende succesrate på over 98 %.

Det andet test-eksempel, hvor de forsøgte at lave ændringer i en side, så de kunne teste deres applikations håndtering af fejlslagne relokationer, beskriver de, hvordan de efter talrige ændringer i siden måtte opgive og i stedet ændre i lokationsbeksrivelsen.

Tredje test beskriver, hvordan annotationer på en hjemmeside overlevede flere ændringer, blandt andet at få indholdet på siden flyttet ind i tabeller, og først opgav, da siden blev konverteret til brug af frames i forbindelse med en større ombygning af siden.

Alt sammen meget positive resultater, der desværre ikke så let lader sig gentage, da de ikke er særligt præcist specificeret. Og det eneste eksempel fra WWW angiver hverken antallet af annotationer, de præcise ændringer af siden eller overlevelsesraten.

Konklusion

Phelps og Wilenskys forestillede sig som nævnt de ovenstående tre metoder i ét, samlet system. De tre metoder komplementerer hinanden, de har forskellige styrker og svagheder, men de er dog kun et udgangspunkt, og der er intet til hinder for at udvide systemet med flere andre mekanismer. Idéen om at bruge flere metoder som backup for hinanden er bestemt et lige så væsentligt bidrag, som det arbejde, de individuelle metoder repræsenterer.

I næste afsnit beskriver jeg Resilient XPaths, der er et nyere bud på en metode til lokalisering af elementer i XML-dokumenter, og som kunne være et bud på en sådan anden mekanisme, der kan passes ind i Phelps og Wilenskys ramme.

2.5.2 Resilient XPaths

Resilient - eller modstandsdygtige - XPaths er navnet på en metode til lokalisering ved hjælp af XPath (se 2.3.2), der er udviklet af Paz og Díaz og publiceret i Providing Resilient XPaths for External Adaption Engines, 2010 [11]. Faktisk beskriver artiklen to problemstillinger: Forskelle på indholdet som følge af ændring i henholdsvis tid og rum. Forskelle i rum dækker over de forskelle, to forskellige personer kan opleve, hvis de besøger en webside på samme tid. Sådanne forskelle kan skyldes profil-indstillinger på det pågældende websted, brugernes nationalitet eller deres tidligere brugsmønster på websiden. Den tidslige dimension dækker over den mere klassiske dynamik - opdatering af sidens indhold eller layout.

Ændringer i rum

For at takle den rumlige dynamik på en side udvikler Paz og Díaz et XPath-udtryk, der er baseret på flere (rumligt) forskellige inkarnationer af den pågældende side. For hver af disse inkarnationer af siden udpeger de manuelt det segment, for eksempel et profilbillede, de ønsker at kunne lokalisere, hvilket giver dem et antal forskellige, absolutte XPaths. På dette sæt af XPaths køres en induktions-algoritme, der så at sige smelter udtrykkende sammen til et enkelt, der kun består af de dele, der var fælles for alle udtryk i sættet. Der dannes således et generelt udtryk, der entydigt identificerer elementet i alle inkarnationer af siden.

Induktion

Algoritmen sammenligner to XPath-udtryk ad gangen, skridt for skridt, og danner et kombineret udtryk. Dette gentages til alle XPaths er blevet

betragtet og slået sammen i et enkelt udtryk. Paz og Díaz inddeler de mulige forskelle i tre typer og beskriver, hvordan de kombineres således:

Forskel i position:

XPath1: `/html/body[1]/div[1]`, XPath2: `/html/body[1]/div[3]`

Kombineret: `/html/body[1]/div[conds]`

Forskel i type:

XPath1: `/html/body[1]/div[1]`, XPath2: `/html/body[1]/span[1]`

Kombineret: `/html/body[1]/*[conds]`

Forskel i sti-længde:

XPath1: `/html/body[1]/div[1]`, XPath2: `/html/body[1]/div[2]/div[1]`

Kombineret: `/html/body[1]//div[conds]`

conds dækker her over karakteristika, der er fælles for de to elementer. Det kunne være en fælles `class`-attribut, `width`-attribut eller andet.

Flere af de tre typer kan være til stede på en gang. I sådan et tilfælde må man forlade sig på de fælles karakteristika.

Der er dog et potentielt problem: To elementer kan være helt uden fælles karakteristika, eller disse kan være så generelle, at et kombineret udtryk identificerer flere end det ønskede element. Forfatterne beskriver en mulig løsning, hvor der laves to løsninger for hver sådan situation. Dog erklærer de, at situationen er så tilpas sjælden, at det typisk ikke er nødvendigt at implementere denne løsning.

Som nævnt skal en bruger selv udpege den ønskede lokation i hver brugt udgave af web siden. I scenarier, som de nævnte i kapitel 1 (Introduktion), må sådan en ekstra barriere i det daglige arbejde nok forventes at reducere brugbarheden noget.

Ændringer i tid

Til at overkomme ændringer over tid, tager Paz og Díaz ligesom ved ændringer i rum udgangspunkt i et sæt af forskellige inkarnationer af samme side. Disse bruges til at efterprøve, om deres udledte XPath-udtryk er ækvivalent med udgangspunktet - altså om det nye udtryk for alle udgaver af siden identificerer præcist samme sæt af knuder. Der bliver således også her tale

om ekstra manuelt arbejde i forbindelse med udpegelsen af den lokation, der ønskes gemt.

Selve beregningen af det nye XPath-udtryk starter på samme måde som ved rumlige ændringer. Først beregnes nemlig et fælles XPath-udtryk for alle indsamlede inkarnationer af siden ved hjælp af induktionsalgoritmen beskrevet ovenfor.

Herefter bruges *simulated annealing*, som er en randomiseret algoritme til beregning af lokale optima i et optimeringsproblem. Paz og Díaz' artikel beskriver således, hvordan det at kunne udvikle et XPath-udtryk, der er modstandsdygtigt mod fremtidige, ukendte ændringer, kan oversættes til et optimeringsproblem og løses ved hjælp af simulated annealing.

Simulated annealing

Grundprincippet i simulated annealing er lånt fra den fysiske proces kaldet annealing - eller anløbning på dansk, der bruges inden for for eksempel stål-industrien. Metal-substansen opvarmes her til smeltestadiet og afkøles så langsomt. Under den høje varme i det smeltede metal vil krystallerne bevæge sig relativt frit rundt i massen, og så under den gradvise afkøling vil de falde mere og mere til ro i de positioner, der kræver mindst energi. De vil så at sige søge mod lokale optima. Annealing, både fysisk og simuleret præsenteres i Optimization by Simulated Annealing af Kirkpatrick *et al.* [7], mens algoritmen trin for trin er beskrevet i Optimization Using Simulated Annealing af Brooks og Morgan [3, s. 243].

Som algoritme simuleres anløbningen iterativt. I hver iteration udvælges én af en mængde mulige transformationer af massen svarende til krystallernes mulige "hop" rundt i metal-substansen. Hver tilstand af massen har en mængde energi tilknyttet, og man kan derfor beregne det krævede energi-forbrug for en transformation ved at finde forskellen på energien af massen før og efter transformationen.

Hvorvidt transformationen udføres, afhænger af energi-forbruget samt systemets overordnede temperatur. Hvis transformationen frigiver energi - altså hvis energien efter transformationen er mindre end før - vil den altid blive gennemført. Hvis transformationen kræver energi, udføres den med en sandsynlighed, der vokser, jo mindre energi, der kræves, og jo varmere systemet er. Systemets temperatur falder iteration for iteration, indtil en vis slut-temperatur er opnået.

Det vil altså sige, at tidligt, hvor systemet har høj energi, er tilfældigheden høj, og massen kan udvikle sig i stort set alle retninger - dog med en svag overvægt mod mindre energier. Efterhånden som temperaturen sænkes, vil de optimerende transformationer blive foretrukket, hvorfor massen vil nærme

sig et lokalt optimum.

I en given implementation af simulated annealing, skal man således specificere et sæt af mulige transformationer og en energi-beregnings-funktion. Herefter forløber algoritmen simpelt ved i hver iteration at: udvælge en tilfældig transformation, beregne energi-tilvæksten, udføre et sandsynlighedstjek baseret på energi-tilvækst og temperatur, eventuelt udføre transformationen og slutteligt reducere systemets temperatur.

Resilient XPaths som simulated annealing

I Resilient XPaths består systemets masse af et XPath-udtryk. Målet er at gøre det så modstandsdygtigt som muligt - altså skal XPath-udtrykkets energi blive mindre, jo bedre udtrykket er til formålet. Det mest interessante i Paz og Díaz' arbejde er i den forbindelse deres energi-beregnings-funktion, der led for led i XPath-udtrykket skal tildele lave værdier til de mest modstandsdygtige bidder, mens de sårbare dele skal give høje værdier.

Transformationerne af XPath-udtrykkene virker umiddelbart mere ligetil - de består af forskellige muligheder for at generalisere eller specialisere udtrykket. Her er udfordringen til gengæld, at det transformerede udtryk skal være ækvivalent med det oprindelige udtryk.

Transformationer for Resilient XPaths

Paz og Díaz beskriver i alt seks transformationer, der parvist er hinandens inverse. Altså tre transformationer, der reducerer energien i udtrykket - generaliserer det: t1,t2 og t3, og tre transformationer, der øger energien i udtrykket - specialiserer det, herunder kaldet t1_inv, t2_inv, t3_inv. En oversigt findes i Tabel 2.1.

Det er selvfølgelig ikke muligt til enhver tid at foretage en hvilken som helst transformation på et hvilket som helst skridt i XPath-udtrykket. Wildcards kan for eksempel kun fjernes, når og hvor de findes. I Providing Resilient XPaths for external adaption engines [11] beskrives udførslen som om, alle naboer beregnes - altså at resultatet af alle mulige transformationer findes - og en enkelt udvælges tilfældigt. Den omvendte rækkefølge er dog også en mulighed: Udvælg en tilfældig transformation og beregn naboen, hvilket beregningsmæssigt er noget billigere. De kan have haft andre, ubeskrevne, årsager, til at ville beregne alle naboer.

Det vigtigste at holde sig for øje er, at transformationen udvælges tilfældigt, samt at det XPath-udtryk, den danner, skal være ækvivalent med det oprindelige XPath-udtryk. Sidstnævnte kunne på papiret ligne en tung opgave, da den, som forfatterne nævner, er coNP-komplet, hvis det

Transformationer.			
Type	*Beskrivelse*	*Før*	*Efter*
t1	Fjern wildcard	`/html/*/p`	`/html//p`
t2	Omdan til Wildcard	`/html/div/p`	`/html/*/p`
t3	Fjern attribut	`/html/div[id='42']/p`	`/html/div/p`
t1_inv	Tilføj wildcard	`/html//p`	`/html/*/p`
t2_inv	Gendan navn	`/html/*/p`	`/html/div/p`
t3_inv	Gendan attribut	`/html/div/p`	`/html/div[id='42']/p`

Tabel 2.1: Paz og Díaz' transformationer

skal gøres generelt. Derfor forenkles opgaven til blot at skulle udføres for et afgrænset sæt af sider, nemlig sættet af oprindeligt indsamlede inkarnationer af siden.

Når en tilfældig, ækvivalent transformation er fundet, er næste skridt at bestemme energien af det XPath-udtryk, den vil danne.

Energi-beregnings-funktionen for Resilient XPaths

Energien skal som nævnt reflektere XPath-udtrykkets modstandsdygtighed mod ændringer - jo lavere energi, jo mere solidt et udtryk har vi fundet. Til at vurdere modstandsdygtigheden benytter Paz og Dáz to faktorer: *Change likelihood* og *condition singularity*. Change likelihood dækker over sandsynligheden for ændring, og condition singularity dækker over, i hvor høj grad en attribut kan forventes at være unik for et element.

Energi som funktion af change likelihood og singularity.			
	Change likelihood	*Condition singularity*	*Energy*
Style condition	High	Low	Medium
Layout condition	Medium	Low	Medium
Descriptive condition	Low	High	Low
Content condition	Medium	High	Low
Normal step	High	Low	V. high
Wildcard step	Medium	V. low	High

Tabel 2.2: Let modificeret udgave af Table 1, Providing Resilient XPaths for External Adaption Engines [11, s. 74].

For et givet XPath-udtryk kan det generelt siges, at det er mere modstandsdygtigt, jo mindre sandsynligheden for ændringer i udtrykket er (altså

foretrækkes lav change likelihood). Og jo større chance, der er for at de brugte attributter er unikke, jo større chance er der, for at XPath-udtrykket kun beskriver en enkelt lokation (altså foretrækkes høj condition singularity).

Skønsmæssigt og baseret på erfaringer opdeler og kategoriserer Paz og Díaz de forskellige mulige bestandele i XPath-udtryk, efter change likelihood og condition singularity og tildeler dem energier baseret herpå. Denne inddeling kan ses i Tabel 2.2.

Forfatterne anbefaler, at man prøver sig frem til de enkelte energier for hver klassificering baseret på det konkrete forhåndenværende tilfælde, da siders opbygning og brug af attributter er meget forskellig.

De opstiller følgende funktion for beregningen af energien:

$$\begin{aligned} E(XPath) = & \, a * (numberofnormalstep) \\ & + b * (numberofwildcardstep) \\ & + c * (numberofstyle + layoutconditions) \\ & + d * (numberofdescription + contentconditions) \end{aligned}$$

og de opfordrer til, at man prøver sig frem til de værdier for a, b, c og d, der giver de bedste resultater. Bemærk at // ikke koster nogen energi i Paz og Díaz' opstilling. Det skyldes formentlig at et sådant skridt er "usårligt" i forhold til ændringer, dog kunne man alligevel overveje at lade det koste lidt, da det samtidig har meget lav singularity. I øvrigt beretter Paz og Díaz, at de i et forsøg har haft gode resultater med værdierne: a = 10, b = 4, c = 1 og d = 0.

Iterationer og udvælgelse

Når en transformation er valgt, og energien af XPath-udtrykket før og efter transformationen er fundet, beregnes, hvorvidt transformationen gennemføres. Som allerede nævnt vil transformationer, der frigiver energi - og dermed forbedrer udtrykket - altid blive gennemført. Men der er også mulighed for at forværre udtrykket - for at give muligheden for hop fra et lokalt minimum over i et andet. Dette sker med aftagende sandsynlighed.

Paz og Díaz er ikke særlig eksplicitte omkring, hvordan de beregner denne sandsynlighed. De skriver blot, i tråd med hvordan simulated annealing traditionelt udføres, at: „... a random value depending on the actual temperature and on the energy difference between the both solutions decides whether the transformation to the new neighbour is performed or not." (Providing Resilient XPaths For Third Party Adaption Engines [11], s. 74). Tilgengæld fremhæver de, at disse annealing-parametre er mindre væsentlige end energi-omkostningerne beskrevet i forrige afsnit.

Alt dette sker i hver iteration af algoritmen. Desuden reduceres temperaturen en smule således, at sandsynligheden for at foretage en forværrende transformation falder løbende. De beskriver ikke brug af "equilibrium"-tilstande for hver temperatur, sådan som det er beskrevet i kilderne om simulated annealing [7], [3]. Når temperaturen når et specificeret frysepunkt, stopper algoritmen, og det aktuelle XPath-udtryk returneres.

Relokalisering

Metoden Resilient XPaths udfører sit arbejde på det tidspunkt, lokationen specificeres, i modsætning til Phelps og Wilenskys metoder, der først laver reelt arbejde, når lokationen skal genfindes. Derfor er Resilient XPaths arbejde på relokations-tidspunktet også en enkel opgave: Det er blot at evaluere XPath-udtrykket i forhold til den nye version af dokumentet.

Hvis XPath-udtrykket specificerer mere end en lokation - eller slet ingen - har metoden ingen måde at vælge eller finde alternativer. Det er selvfølgelig tanken, at det indledende arbejde i tilstrækkelig grad bearbejder positionen til, at den fremover kun vil give netop den ene lokation. Om det lykkes, må afprøvningen af metoden vise.

Styrker og svagheder

Da jeg i dette speciale ikke behandler rumlige forskelle særskilt, evalueres her de to dele af Resilient XPaths (induktions-algoritmen og simulated annealing) samlet, da disse begge benyttes ved forskelle over tid. Jeg har allerede nævnt et par gange, at en svaghed ved systemet er den øgede manuelle involvering. Dette er dog kun en svaghed set fra en brugbarhedsvinkel, og det ikke noget, der forringer metodens evne til at genfinde lokationer i et dokument.

Resilient XPaths forsøger der imod at drage fordel af denne udvidede startviden, der stammer fra kendskabet til flere versioner. Antaget at forskellene mellem disse versioner er nogenlunde repræsentative for typerne af fremtidige ændringer, må det da også forventes, at det lykkes. I hvert fald er de gemte karakteristika for lokationen unikke nok til entydigt at specificere den over dette sæt af sider.

Metoden har nogle ting tilfælles med Phelps og Wilenskys tree-walk-metode, da begge grundlæggende bygger på stier i DOM-træet. Men hvor Phelps og Wilensky gemmer den præcise sti og lidt til, der gøres stien i Resilient XPaths så "upræcis", som det er muligt, uden at den bliver tvetydig i det oprindelige sidesæt. Dette betyder, at der er færre ændringer, der kan påvirke en Resilient XPath. Til gengæld er den meget sårbar, hvis

en sådan ændring forekommer, da der så ikke er nogen mulighed beskrevet for reparation af udtrykket.

For eksempel er udtrykket `//p[19]` modstandsdygtigt over for samtlige ændringer i et dokument - med undtagelse af to: Enten at der et andet sted i siden dukker et element af typen `p` op med 19 eller flere børn, eller at det angivne `p` får eller mister en tidligere søskende af samme type - altså for eksempel, hvis der bliver slettet eller tilføjet en paragraf kort inden den angivne paragraf. Hvis en af de to ting sker, vil det være umuligt at afgøre, hvor den korrekte lokation er.

Resilient XPaths er altså meget afhængig af det sæt af sider, der tages udgangspunkt i, da der ikke gemmes ret meget information om lokationen, og der derfor ikke ret godt kan foretages nogen reparation på et senere tidspunkt.

Forfatternes resultater

Paz og Díaz har hentet data til deres tests fra "the way back machine" på archive.org[11]. De har hentet versioner af Yahoos forside fra 1996 til 2007 og El Mundos forside fra 2000 til 2009 og brugt disse til afprøvningen. Testen forløb ved, at de skrev et XPath-udtryk, der pegede på et element på hver af siderne, og brugte 2-3 sider fra en periode på et halvt år, til at danne det "Resilient XPath"-udtryk, der skulle bruges efterfølgende.

I løbet af de tidsrum, de havde sider fra, ændrede Yahoos side sig 15 gange og El Mundos otte gange. Paz og Díaz rapporterer, at deres XPath-udtryk overlevede 90 % af de mindre ændringer, svarende til æstetiske ændringer og små tilføjelser af data, og 10 % af de større ændringer, hvilket dækker over for eksempel nyt layout på siden. Dette omregner de til et gennemsnit på 62 %, hvorved de vægter små og store ændringer lige højt.

Konklusion

De rumlige forskelle på en side er ikke særskilt betragtet af Phelps og Wilensky, men det betyder ikke, at deres metoder ikke kan bruges over for rumlige ændringer. Forskellen på rumlige og tidslige ændringer ligger i højere grad i muligheden for at tage udgangspunkt i flere versioner af en side, når en lokation gemmes, således at man allerede på forhånd har en viden om, hvilke slags ændringer, der er typiske. Den viden medfører dog den omkostning, at den manuelle indblanding øges, idet lokationen, der skal gemmes, må udpeges for hver side.

[11] `http://archive.org` er et digitalt bibliotek over internetsider og andre kulturstykker i digital form

Paz og Díaz takler altså lokaliseringsproblemet i en modsat rækkefølge af Phelps og Wilensky. Hvor de sidstnævnte gemmer præcise oplysninger om lokationen og siden slækker på præcisionen, hvis lokationen ikke kan genfindes, forsøger Paz og Díaz at abstrahere så mange detaljer væk fra deres lokations-beskrivelse som muligt, inden denne gemmes, for at få så generelt et udtryk som muligt til senere lokalisering. Fordelene ved dette er, at man kan skære meget fra og stadig vide, at man rammer den samme lokation i det originale dokument, samt at man kun udfører regnearbejdet én gang - når man gemmer - og siden blot skal evaluere det gemte XPath-udtryk.

Paz og Díaz giver til gengæld ikke noget bud på, hvad man kan gøre, når deres metode fejler. Givet Resilient XPaths og Phelps og Wilenskys tree-walk-metodes ensartede natur - de gemmer begge stier i DOM-træer - ville det være nærliggende at tilføje tree-walk-metodens fremgangsmåde, når den ikke længere identificerer den ønskede lokation, til Resilient XPaths - men her kan et problem være, at man simpelthen ikke har tilstrækkelige data gemt til overhovedet at vide, hvor man skal begynde, og hvad man skal kigge efter.

En anden ulempe eller mangel ved Resilient XPaths er, at de kun specificerer på knude eller element-niveau, ligesom det var tilfældet med Phelps og Wilenskys unikt-ID-metode. Det er altså ikke muligt at specificere lokationer inde i medie-indhold som tekst, lyd eller video. Hertil må der i givet fald bygges en udvidelse - en nærliggende løsning kunne være XPointer (se 2.3.3), der netop er en udvidelse af XPath med det formål.

En styrke ved Resilient XPaths er netop brugen af XPath, der er en W3C-standard og er understøttet i de fleste større browsere, hvilket fjerner nogle af de forhindringer, man kunne støde på i forbindelse med implementation.

Resilient XPaths er altså en standard-baseret tilgang til lokalisering, der har givet opløftende resultater. Men det er også en metode, der kræver en vis manuel tilpasning, når lokationer gemmes, og som stadig mangler lidt med hensyn til specifikation under element-niveau og fallback-metoder, når det ønskede element ikke længere genfindes.

2.6 Opsummering

I afsnit 2.1 har jeg dokumenteret nytten af og behovet for velfungerende relokaliseringsmetoder. Adskillige kilder dokumenterer World Wide Webs foranderlige natur, mens kilder som Unsworth [13], Brush [4] og Marshall [9] har beskrevet typen af, behovet for og forventningen til annotationer i digitale dokumenter.

Videre har jeg gennemgået den hypermedie-historiske baggrund for begreber som ankre og Location Specifiers (lokationsbeskrivelser) (afsnit 2.2), der har sin rod i Dextermodellen [6] og Grønbæk og Triggs bearbejdning af samme [5]. Jeg har påpeget, hvorledes denne model tager hensyn til særlige karakteristika ved World Wide Web, som for eksempel at man ikke har mulighed for at redigere og indsætte ankre i de sider, man gerne vil specificere lokationer i.

I World Wide Web-standarder, afsnit 2.3, er et par af W3Cs standarder blevet præsenteret, da de er centrale i implementationerne og afprøvningen af de udvalgte lokationsbeskrivelser. For eksempel er Resilient XPaths baseret direkte på XPath og DOM-træer, mens jeg blandt andet benytter XPointere til at specificere facitlisten for de lokationer som algoritmerne skal finde.

I den resterende del af kapitlet har jeg først kort forklaret forskellen på intra- og inter-dokumentændringer, hvorefter jeg i detaljer har gennemgået algoritmer og karakteristika for to udvalgte metoder til intra-dokument-relokalisering: Robust Intra-Document Locations [12] og Providing Resilient XPaths For External Adaption Engines [11]. De to metoder bliver i den resterende del af specialet evalueret gennem implementatition og test.

3

Metode og værktøjer

Dette kapitel beskriver de værktøjer og den fremgangsmåde, jeg har benyttet i arbejdet med at implementere og teste de metoder til relokalisering af web-indhold, der er beskrevet i kapitel 2.

Blandt andet vil jeg beskrive Mozilla-frameworket, der måske er bedst kendt som fundamentet for Firefox-browseren, men som er meget alsidigt og fleksibelt, ligesom jeg vil redegøre for det framework for LocSpecs, jeg har udviklet, de planlagte test og den test-suite, jeg benytter til at udføre testene.

3.1 Formål

Jeg har tidligere beskrevet to metoder til at relokalisere fragmenter af web-indhold: Robust Intra-Document Locations (2.5.1) og Resilient XPaths (2.5.2). Målet med specialet er videre at evaluere og sammenligne metoderne gennem implementation og test. Ophavsmændene til de forskellige metoder har hver især implementeret og testet deres egne metoder, men resultaterne er ikke umiddelbart sammenlignelige, da de udførte test er meget forskellige. Derfor har jeg udviklet et framework og en række test-cases, der kan håndtere implementationer af alle metoderne ensartet og give sammenlignelige resultater.

Det ønskede output er for hver test-case, en angivelse af, om den afprøvede metode korrekt eller delvist korrekt har lokaliseret det gemte fragment i et ændret web dokument. Hvis ikke det er tilfældet, skal det outputtes, om metoden helt har opgivet lokaliseringen, eller om fragmentet er forkert lokaliseret, da dette har afgørende betydning i et brugbarhedsøjemed. (Se afsnit 2.1.3 om forventninger til digitale annoterings-værktøjer). Givet et vist antal test-cases kan metoderne således sammenlignes på succesrate og fejl-type.

Metoderne er ikke helt ensartede i deres forudsætninger til input data. For eksempel kræver Paz og Díaz' Resilient XPaths et sæt af dokumenter, der tidsligt repræsenterer den samme side, men som er rumligt forskellige, hvorimod Phelps og Wilenskys Robust Locations blot kræver det aktuelle dokument, der indeholder den lokation, der skal gemmes. Derfor må både test-cases og framework have en vis fleksibilitet, der vil blive beskrevet nærmere i de følgende afsnit.

Første skal valget af platform beskrives og motiveres.

3.2 Mozilla-platformen

Ved at bruge Mozilla-platformen[1], får man adgang til et stort bibliotek af allerede implementerede komponenter, der bruges internt i programmer som Thunderbird og Firefox - blandt andet komponenter til håndtering af HTTP-requests, navigation af DOM-træer og styring af sikkerhed. Disse komponenter er en del af XPCOM[2], der gør det muligt at lave interfaces eller bindinger mellem forskellige programmeringssprog, således at objekter skabt af kode i et sprog kan bruges af kode i et andet sprog. Koblingen mellem de to sprog sker via XPConnect og fungerer usynligt for udvikleren.

Mozilla-platformen har en anden fordel, nemlig at man får foræret en smutvej til en grafisk brugerflade i form af add-ons til Firefox. Under udviklingen af testcasesne har det været værdifuldt smertefrit at kunne hente websider, markere bidder af dem og få det range, der repræsenterer det markerede, serveret på et sølvfad i form af et XPointer-udtryk. At bruge XPCOM-komponenterne i et add-on kræver ikke meget kode takket være XPConnect og XPIDL.

3.2.1 XPCOM, XPConnect og XPIDL

XPCOM er ikke bare en objektmodel, der tillader kode i forskellige sprog at tilgå de samme objekter, det er også en samling af en enorm mængde komponenter, der danner grunden for Mozilla-organisationens mange applikationer på tværs af platforme og operativsystemer. Komponenterne i XPCOM er således tilgængelige og brugbare på alle væsentlige platforme [10, Kap. 16]. Platform-kompatibilitet er dog ikke et fokusområde for dette speciale, hvorfor XPCOMs funktion som objekt-håndteringssystem er mere interessant.

Et almindeligt scenarie er for eksempel at benytte en af de eksisterende komponenter udviklet i C eller C++ i en add-on, der er kodet i JavaScript. Via et kig i komponentens XPIDL-fil[3] og et par enkelte linjers kode, kan en add-on udvikler således bruge komponenten, som var den et JavaScript-objekt.

I Oversigt 3.1 anføres et eksempel på simpel fil-håndtering i JavaScript via en XPCOM-komponent, der formentlig er skrevet i C++. I eksemplet

[1]Kernen i Mozilla-platformen hedder egentlig XULRunner. Information om download af XULRunner findes her: `https://developer.mozilla.org/en/XULRunner`

[2]Cross Platform Component Object Model, Mozilla-organisationens model for håndtering af objekter mellem kode i forskellige sprog på tværs af platforme.

[3]Cross Platform Interface Definition Language, det sprog i hvilket interfaces defineres i XPCOM

oprettes en fil med navnet domain.dat i add-on'ens installationsmappe:

```
1 var file = Components.classes['@mozilla.org/file/local;1'].
      createInstance();
2 file.QueryInterface(Components.interfaces.nsILocalFile);
3 file.initWithPath(extension.extPath);
4 file.append('domain.dat');
5 if( !file.exists() || !file.isFile() ) {
6     file.createUnique(Components.interfaces.nsIFile.
          NORMAL_FILE_TYPE, 0666);
7 }
```

Oversigt 3.1: JavaScript eksempel på brug af XPConnect

Variablen `file` oprettets som en instans af '`@mozilla.org/file/local;1`'-klassen, hvorefter interfacet `nsILocalFile` hentes og giver adgang til de metoder, der er defineret deri. På samme vis fås adgang til hundreder af andre komponenter fra JavaScript. '`@mozilla.org/file/local;1`' er et kontrakt- eller klasse-id, der unikt angiver Mozilla-organisationens implementation af interfacet. Den samme komponent kan være kodet til at overholde forskellige interfaces, og afhængigt af hvordan den skal bruges, kan man kalde QueryInterface på objektet og specificere via hvilket understøttet interface, man ønsker at tilgå det.

Teknikken XPConnect er bindingen mellem XPCOM og JavaScript, og som det ses i Oversigt 3.1, er det helt usynligt for JavaScript-udvikleren, hvilket sprog `nsILocalFile` er implementeret i. Det eneste, der kræves, er, at komponenten er implementeret i et sprog, for hvilket der er udviklet en binding til XPCOM, og at komponenten og dennes interface registreres via Mozilla-platformen.

Udvikler man selv en komponent skal man således også overholde nogle enkle formalia. Der skal specificeres et interface i XPIDL, som kompileres til en XPType-fil, og komponenten skal implementeres i et understøttet sprog - det være sig C, C++, JavaScript eller andre. Både interface og implementation skal have et unikt id, og komponenten skal indeholde noget standard-kode, der bruges af Mozilla, til at registrere den korrekt i XPCOM-databasen. Komponenten og type-filen skal placeres i et bibliotek, der scannes af Mozilla-platformen - hvis komponenten skal bruges i en Firefox add-on, kan der oprettes et bibliotek med navnet *components* i add-on'ens rod-bibliotek til formålet.

XPCOM med de derunderhørende sprog og teknikker XPIDL og XPConnect er altså en meget alsidig universalkniv, der er relativt let at tilgå og benytte. Hvad enten man laver en selvstændig applikation, udbygger Mozilla-platformens komponenter eller laver add-ons til nogle af de mange programmer, der er baseret på platformen, får man en masse hjælp serveret

uden at skulle tænke over sprogvalg eller kompleksitet ved integrationen.

3.2.2 XPCOM-komponenterne

Ovenfor viste jeg et glimt af `nsILocalFile`, og hvordan en instans af denne initialiseres. I afsnit 2.3.1 introducerede jeg nogle dele af W3Cs Document Object Model. Håndteringen af disse er implementeret som komponenter i XPCOM, blandt andet findes `nsIDOMDocument`, `nsIDOMRange` og `nsIDOMNode`, der repræsenterer henholdsvis et helt dokument, et range i et dokument og et enkelt element i et dokument. Disse komponenter overholder W3Cs DOM-specifikation, så der er ikke grund til yderligere uddybning her.

I stedet vil jeg fremhæve XPointerService[4], der ikke er en del af standard XPCOM-pakken. Det er en selvstændig komponent, der udvider XPCOM med en service til konvertering af DOM-Ranges til og fra XPointer-udtryk. Dette har vist sig at være en værdifuld funktion i forbindelse med konstruktion af test-casene, da både "facitlisten" for relokaliseringen og specifikationen af lokationen i de oprindelige dokumenter således har kunnet udtrykkes med XPointer-udtryk.

3.2.3 xpcshell

Til selve test-suiten har jeg brugt xpcshell, der kan afvikle JavaScripts fra en tekstshell med XPCOM understøttelse således, at man ikke behøver indlæse en grafisk brugerflade eller udvikle en hel applikation. Det er tilsyneladende ikke muligt at tilføje XPCOM-interfaces på script-niveau med xpcshell, så det har været nødvendigt at kopiere egne XPType-filer til Mozilla-platformens komponent-bibliotek - hvilket ikke er et problem udover, at det besværliggør opsætning af test-stuiten en smule.[5]

3.2.4 Konklusion

Mozilla-platformen har tilbudt standardiserede løsninger til stort set alle de behov, jeg har haft i forbindelse med håndtering af DOM-objekter i specialet. Brugen af platformen fra JavaScript og implementation af nye komponenter i JavaScript har været enkel og smertefri, efter den umiddelbare forståelse for brugen af XPCOM var opnået.

[4] `http://xpointerlib.mozdev.org/`

[5] Information om brug af xpcshell findes her: `https://developer.mozilla.org/en/xpcshell`

Om andre platforme kunne have givet tilsvarende funktionalitet, har jeg ikke undersøgt, da Mozilla ret hurtigt i forløbet virkede som en ideel løsning til mit formål.

3.3 Frameworket: DynamicAnchors

Til brug for testene har jeg udviklet et framework, kaldet DynamicAnchors, baseret på Mozilla-platformen. Det er således muligt at benytte frameworket i alle Mozilla-baserede applikationer, hvor det kan være interessant at gemme lokationer i foranderlige DOM-træer.

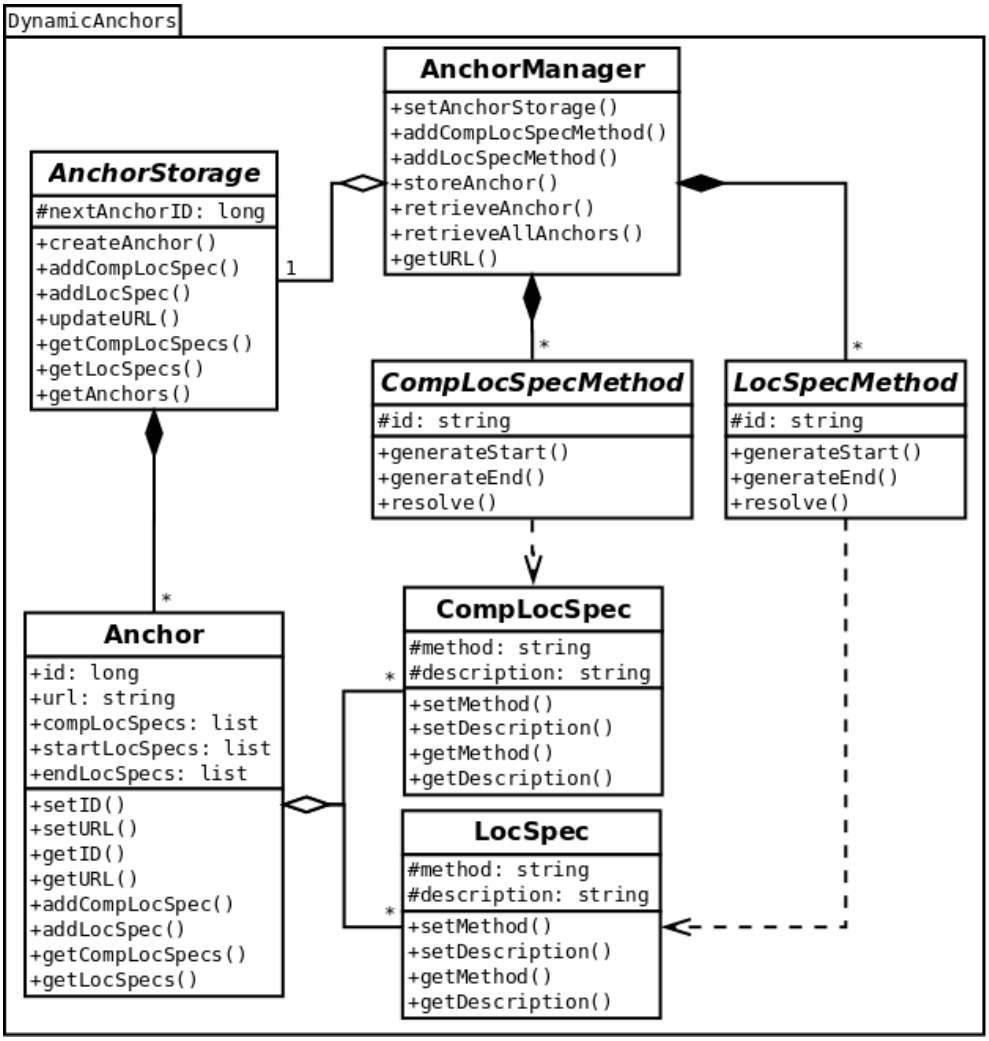
Interaktion med frameworket er tænkt til at ske gennem AnchorManager-klassen, men de øvrige komponenter kan i en vis udstrækning også bruges selvstændigt. Frameworket består af i alt syv interfaces: DAnchorManager, DAnchorStorage, DAnchor, CompLocSpecMethod og LocSpecMethod samt CompLocSpec og LocSpec. Frameworkets struktur kan ses i diagrammet i figur 3.1.

Dele af frameworket er ikke endeligt implementeret, da de ikke har været aktuelle i forbindelse med dette speciale. Det drejer sig i særdeleshed om brugen af CompLocSpecs, der kun er relevante ved inter-dokument-ændringer, men også idéer til andre AnchorStorage-implementationer og en udvidet AnchorManager er blevet nedprioriteret, da de ikke direkte har relevans for specialet.

3.3.1 AnchorManager

AnchorManager er som nævnt det primære ansigt udadtil fra frameworket, selvom hver enkelt komponent kan instantieres og bruges selvstændigt. Den initialiseres med et AnchorStorage, der husker de gemte ankre i frameworket, og der kan tilføjes et antal CompLocSpecMethods og LocSpecMethods, som herefter benyttes til at generere CompLocSpecs og LocSpecs, når der gemmes ankre.

Ankre gemmes via metoden `storeAnchor`(*url*, *doms*, *ranges*). *Url* er adressen på det indeholdende dokument, mens *doms* og *ranges* er lister af sammenhørende DOM-Dokumenter og DOM-Ranges - hvor hvert range angiver ankerets position i det tilhørende dokument. I den nuværende implementation af AnchorManageren består arbejdet ved `storeAnchor` blot i at uddelegere til de LocSpec-metoder, der er tilføjet - som producerer deres respektive LocSpecs - og herefter at overdrage disse LocSpecs til AnchorStorage. Til brugeren af AnchorManageren returneres et id for det gemte anker, der kan bruges til senere opslag.



Figur 3.1: Klassediagram over frameworket

Et anker hentes ud af frameworket ved at kalde metoden `retrieveAnchor`(*url*, *dom*, *anchor*, *finalRating*). *finalRating* er her et out-parameter, der bruges til at returnere den rangering, som det bedste fundne anker vurderes at have. *Anchor* er det id, som ankeret er gemt under i frameworket. *Url* og *dom* er henholdsvis adressen på og DOM-Dokumentet for den side, som ankeret ønskes fundet i. *Url* bruges dog ikke i den nuværende implementation af frameworket, hvor metoden forløber ved, at de gemte LocSpecs for det givne anker-id hentes i AnchorStorage og fodres til deres tilhørende LocSpec-metoder, som beregner et range og en rangering for hver på baggrund af lokationsbeskrivelsen og DOM-træet.

Der er også metoder til at hente en URL for et anker-id og at hente en liste af anker-id'er for en given URL. Disse benyttes dog ikke i specialet, men kan være nyttige i en brugssituation, hvor man ikke ved hvilke ankre, der findes på en side.

3.3.2 AnchorStorage

AnchorStorage er lagerområdet i frameworket. Det er adskilt fra manageren, da man kan forestille sig flere typer lager. Til dette speciale har jeg implementeret en simpel version, der ikke lagrer ankrene permanent - de huskes med andre ord kun i en liste i hukommelsen. I brugssituationer kunne man dog forestille sig et permanent lager i en database, på en webserver eller som en peer-to-peer-løsning.

Når der skal gemmes et nyt anker, kaldes først `createAnchor`, som opretter ankeret i lageret og associerer det til en URL, hvorefter et id for ankeret returneres. Dette id benyttes, når der tilføjes LocSpecs til ankeret. Id'et benyttes også, når LocSpecsne skal hentes ud af lageret med `retrieveAnchor`.

I den nuværende implementation foregår der som sagt ikke ret meget andet end indsættelse i og hentning fra lister. Der er metoder til at hente alle anker-id'er for en URL og til at hente en URL for et givent anker-id - disse metoder understøtter de tilsvarende metoder i AnchorManageren, og finder ligesom dem ikke nogen anvendelse i dette speciale.

3.3.3 LocSpecMethods

I frameworket skelnes mellem LocSpecMethods - der bruges til intra-dokument-lokationer - og CompLocSpecMethods - der bruges til inter-dokument-lokationer. I specialet benyttes CompLocSpecs dog ikke, hvorfor interfacet for disse ikke beskrives her.

LocSpecMethod-interfacet er til gengæld centralt, da det definerer brugen af de to metoder, Robust Intra-Document Locations og Resilient XPaths, der bliver testet i specialet. Det specificerer blot tre metoder: `generateStart`, `generateEnd` og `resolve`.

`GenerateStart` og `generateEnd` tager begge en liste af sammenhørende DOM-dokumenter og DOM-ranges som parametre. Forløbet i de to metoder vil også typisk være ens, da de dybest set foretager samme arbejde: At beskrive en lokation. De beskriver henholdsvis start- og slutlokationen for det anker, der skal gemmes, og returnerer begge et LocSpec-objekt. Når inputtet til metoderne er lister af dokumenter og ranges, og ikke bare et enkelt dokument og range, skyldes det, at for eksempel Resilient XPaths benytter et sæt af dokumenter til at udlede lokationsbeskrivelsen.

`Resolve`-metoden får som input et DOM-dokument, to LocSpec-objekter (en start- og en slut-lokation) - og returnerer et DOM-range og en rangering mellem 0 og 100. 0 svarer til, at det ønskede anker ikke kunne findes i dokumentet, mens 100 svarer til, at metoden selv vurderer at have ramt ankeret helt præcist. Hvis der har været ændringer i dokumentet, kan resolve-metoden være nødt til at droppe nogle af detaljerne i lokationsbeskrivelserne, og det bør medføre en lavere rangering.

Disse rangeringer vil ikke kunne garanteres at være korrekte, og en metode, der giver en rangering på 50, kan have fundet en "mere korrekt" løsning, end en anden metode, der returnerer 60. Især i et brugsscenarie, hvor flere udviklere har tilføjet hver deres implementationer af LocSpecMethods, vil rangeringerne være svære at stole på. Dog benyttes de i frameworket i AnchorManager, til at vurdere hvilket DOM-range, der returneres til brugeren af frameworket, hvorfor balancering af rangeringerne mellem metoderne har været vigtig.

Et alternativ kunne være at returnere samtlige fundne ranges til brugeren eller blot at returnere den først fundne løsning, som Phelps og Wilensky beskriver i deres overordnede strategi (se afsnit 2.5.1). Her benyttes alle tilføjede metoder altså til at lokalisere et range, og det bedst rangerede range returneres i tillid til, at balanceringen i metoderne er nogenlunde tilfredsstillende.

De egentlige implementationer af LocSpecMethods beskrives i kapitel 4, LocSpecMethod-implementationerne.

3.3.4 Anchors og LocSpecs

Et Anchor i frameworket er et objekt, der repræsenterer et anker. Det indeholder beskrivelsen af et anker i form af lokationsbeskrivelser for ankerets startpunkt og slutpunkt, samt en komponentlokationsbeskrivelse. Ankeret

identificeres i frameworket via et id og en URL.

I frameworket er Anchor-interfacet blot implementeret som tre lister: *startLocSpecs*, *endLocSpecs* og *compLocSpecs*.

De grund-elementer, som ankrenes lister er opbygget af, er LocSpec- og CompLocSpec-objekterne. Disse har blot et id-felt, der identificerer den metode, der har genereret dem og et description-felt, der indeholder den tekstuelle beskrivelse af lokationen, som er det reelle resultat af LocSpec-Methods `generateStart` og `-End`-metoder.

3.3.5 Opsummering

Frameworket er opbygget efter den grundtanke, at det skal kunne finde anvendelse som anker-database i applikationer fra det virkelige liv. Dog har der af tidsmæssige årsager været mere fokus på at udvikle de dele, der konkret har skullet bruges i forbindelse med specialet. Frameworket har derfor visse svagheder - blandt andet brugen af rangeringer og håndteringen af komponent-lokationsbeskrivelser - som gør den nuværende version uegnet til egentlig brug.

Til brug for dette speciale fungerer frameworket dog efter hensigten. Test-suiten behøver blot at oprette en AnchorManager, et AnchorStorage og angive de LocSpecMethods, der ønskes testet. Herefter er brugen reduceret til at kalde `storeAnchor` og `retrieveAnchor` på manager-objektet.

3.4 Test

Før jeg beskriver implementationerne af LocSpecMethodsne i kapitel 4, præsenteres her de test, de skal udføre. Som nævnt indledningsvist i dette kapitel (afsnit 3.1) er formålet med testene at opnå sammenlignelige resultater af metoderne. Derfor udføres den samme serie test med hver enkelt metode, mens testene hver især repræsenterer enten cases indsamlet fra nettet eller skræddersyede cases, der skal demonstrere stærke og svage sider ved metoderne.

Test-processen afvikles af et JavaScript, start_test.js, som håndterer indlæsning og initialisering af frameworket og test-data, samt output af de umiddelbare resultater. Når jeg i det efterfølgende omtaler "test-suiten", er det set-up'et med dette script og test-data, jeg refererer til. Brugen af test-suiten er dokumenteret i Bilag B.

3.4.1 Typer

I afsnit 2.1.2 præsenterede jeg en række kilder, der viste, hvordan få sider på WWW er helt statiske, og mange ændrer sig hyppigt, mens kilderne i afsnit 2.1.3 blandt andet beskrev de typer af ændringer, man kan støde på: *sletning, omformulering, anker-flytning* og *paragraf-flytning*.

For at kunne afgøre LocSpec-metodernes styrker og svagheder, er det derfor nødvendigt med mange forskellige test-cases, således at de bliver afprøvet bredt på alle disse typer af ændringer. I det følgende beskriver jeg, hvordan testene er opbygget, hvordan de forløber, og hvilken type resultat de giver, så de kan opfylde dette mål om afprøvning og give resultater, der lever op til ønsket om sammenlignelighed.

3.4.2 Opbygning

Som udgangspunkt for hovedparten test-casene benyttes revisions-historien af forskellige wikipedia-artikler[6], hvori der findes utallige eksempler på alle fire typer ændringer. I andre test-cases vil en enkelt side bliv udvalgt, og jeg vil da foretage redigeringer i nogle kopier af siden, for at teste præcist på specifikke ændringstyper.

For hvert test-sæt specificeres:

- Et sæt af originale sider.
- Et sæt af originale lokationer.
- Et sæt af senere sider.
- Et sæt af senere lokationer.

De originale sider repræsenterer egentlig en enkelt side i rumligt forskellige inkarnationer. For wiki-siderne er dog brugt tidsligt beslægtede sider i stedet, da det ikke umiddelbart er muligt at indhente rumligt forskellige sider. Siderne er gemt i XHTML format, hvilket er nødvendigt for den parser, jeg bruger.

Der kan være én eller flere lokationer specificeret for de originale sider; for hver lokation er lokationens position i hver af siderne angivet med et xpointer-udtryk. Lokationerne gemmes i XML-format og et eksempel, der illustrerer idéen, kan ses i Bilag A.

Sættet af senere sider repræsenterer sidens udvikling. Der kan eventuelt være tale om en enkelt senere side, men oftest vil det være en serie af sider,

[6]Revisions-historien for en wikipedia-artikkel tilgås ved at klikke "Se historik" (dansk) eller "View history" (engelsk) øverst til højre på siden. Eksempel: `http://da.wikipedia.org/w/index.php?title=Datalogisk_Institut_%28Aarhus_Universitet%29&action=history`

der repræsenterer en længere udvikling - eller som nævnt en modifikation af den oprindelige side for at simulere et særligt tilfælde.

De senere lokationer er specificeret i en XML-fil efter samme model som de originale lokationer. Hver senere lokation repræsenterer en af de originale lokationer og har tilknyttet en xpointer for hver senere side, der findes. Disse lokationer er mit bud på, hvor de oprindelige lokationer kan genfindes i de senere sider efter diverse ændringer - en art facitliste.

3.4.3 Forløb

En test initialiseres ved, at den eller de valgte LocSpecMethod-implementeringer indlæses i frameworket. Herefter indlæses og parses de originale sider og originale lokationer til DOM-repræsentationer. For hvert (lokation, inkarnation)-par bestemmes et DOMRange. DOM-ranget spænder over de knuder i DOM-træet for inkarnationen, som identificeres af det xpointer-udtryk, der er specificeret for netop det (lokation, inkarnation)-par. Oversættelsen fra xpointer til DOM-Range sker ved hjælp af XPointerService.

Dette arbejde resulterer i en liste af DOM-Ranges og DOM-Dokumenter, der for hver lokation angiver placeringen af lokationen i hver inkarnation af siden. Disse to lister gives som input til frameworket, der på baggrund heraf - via de indlæste LocSpec-metoder - gemmer en eller flere LocSpecs under et fælles id[7], således at hver lokation får sit eget id.

Når alle lokationer er gemt, starter relokaliseringen i de senere sider. De senere lokationer og sider indlæses og parses til DOM-repræsentationer. Hver lokation har som nævnt et id-nummer. Udfra dette nummer og et DOM-træ for en af de senere sider, bliver frameworket bedt om at returnere et DOM-Range, svarende til den pågældende lokation i det pågældende DOM-træ[8].

De returnerede ranges, der altså er beregnet af LocSpec-metoderne ud fra de gemte LocSpecs, sammenlignes på flere måder med de "facit-ranges", som de manuelt udvalgte xpointer-udtryk evalueres til af XPointerService.

3.4.4 Resultat

Det beregnede range sammenlignes med det forventede, "facit-range", ved at kigge på de knuder eller elementer, de spænder over, samt den tekst, de repræsenterer. Test-suiten skriver resultaterne til en fil, der siden hen parses og bruges til genereringen af de tabeller, der findes i kapitel 5.

[7] `nsIDAnchorManager.store()`

[8] `nsIDAnchorManager.retrieve()`

Outputtet beskriver resultatet af hvert enkelt forsøg på relokalisering, og det indeholder dels nogle nøgletal, der kan bruges til at identificere hvilken lokation i hvilken fil, der er tale om og dels de reelle resultater.

De data, der beregnes er følgende:

- Om metoden fandt en lokation
- Om facitlisten angav en lokation
- Metodens interne rangering af lokationen
- Om metoden korrekt udelod lokationen
- Om metoden forkert udelod lokationen
- Om metoden fandt den korrekte lokation
- Om metoden fandt en delvist korrekt lokation
- Et kvalitets-estimat for en evt. delvist korrekt lokation
- Om metoden fandt en forkert lokation
- Et afstands-estimat fra en evt. forkert lokation til den korrekte

De tre første resultater er ganske enkelt tjek af, hvad metoderne returnerer. Om metoden korrekt eller forkert udelod en lokation kan beregnes ud fra de første to resultater i listen ovenfor.

Den korrekte lokation antages at være fundet, hvis ét af to tilfælde er gældende: Enten skal facitlistens DOM-Range og det DOM-Range metoden har fundet have præcis samme start- og slutpunkter, fundet ved hjælp af DOM-Range-objektets indbyggede `compareBoundaryPoints`-funktion. Eller også skal teksterne, de to ranges spænder over, være identiske, samt de to ranges skal overlappe. Dette andet, slækkede, kriterie, at de to ranges ikke behøver være identiske, er introduceret for at undgå, resultater bliver afvist af strukturelle årsager. Når de to ranges overlapper og teksten er ens, må det også være den samme tekst.

En delvist korrekt lokation er fundet, hvis de to ranges overlapper, uden at der er tale om en helt korrekt lokation. Når der er tale om en delvist korrekt lokation, beregnes også en grovkornet vurdering af kvaliteten af lokationen. Den beregnes som hundrede gange tekstlængden af overlappet delt med den største af tekstlængderne af facit og løsning. Hvis det korrekte anker er 20 tegn langt, den fundne løsning er 30 tegn lang og de overlapper på 10 tegn, vil kvaliteten være $100 * 10/30 = 33$ %.

Og slutteligt, hvis metoden fandt en lokation, der hverken er helt eller delvist korrekt, regnes den som forkert. I så fald beregnes den tekstuelle afstand fra den fundne til den korrekte lokation i forhold til dokumentets længde. Hvis den fundne lokation er 200 tegn fra en korrekt lokation i et dokument med længden 2000, vil afstanden således være 10 %. Hvis der ikke findes en korrekt lokation, vil afstanden være 100 %.

På baggrund af disse data produceres de tabeller, der ses i kapitel 5 - Resultater. Tabellerne samt betydningen af resultaterne bliver diskuteret i afsnit 5.1.

3.5 Opsummering

I dette kapitel har jeg præsenteret de værktøjer, jeg har brugt til at udføre test-casesne. Jeg har bygget mit framework til lagring af ankre i dynamisk webindhold som XPCOM-komponenter til Mozillas framework. Frameworket er for sit vedkommende brugt i den test-suite, jeg har konstrueret til at udføre de forskellige test.

Jeg har også beskrevet hvilke typer test, jeg udfører. Testenes opbygning er kort introduceret, mens forløbet og resultaterne af en test er præsenteret mere uddybende.

I næste kapitel beskriver jeg implementationerne af de LocSpecMethods, jeg har brugt til testene.

4

LocSpecMethod-implementationerne

I kapitlet Teori og baggrund, afsnit 2.5.1 og afsnit 2.5.2, gav jeg en beskrivelse og en umiddelbar vurdering af to metoder til lokations-bestemmelse - henholdsvis Robust Intra-Document Locations og Resilient XPaths. For at kunne uddybe disse vurderinger og foretage en reel sammenligning af metodernes styrker har jeg implementeret dem. I dette kapitel beskriver jeg implementationerne.

Desværre er ingen af metoderne fuldstændigt specificeret, så det har været nødvendigt at gøre nogle antagelser. Endvidere er der af forskellige årsager små forskelle mellem implementationerne og metodernes beskrivelser i kilderne. Disse forskelle bliver påpeget, hvor de er gældende, og opsummeret sidst i afsnittet for den pågældende metode.

Ud over at implementere metoderne fra litteraturen har jeg også udviklet min egen metode. Idéen bag metoden er sammen med detaljer om implementationen beskrevet i afsnit 4.3.

Implementationerne overholder LocSpecMethod-interfacet beskrevet i afsnit 3.3.3.

4.1 Robust Locations

Robust Intra-Document Locations er titlen på Phelps og Wilenskys pakkeløsning til relokalisering af indhold fra web-sider. Pakken består af hele tre metoder, der enkeltvist kan betegnes som LocSpecMethods, hvorfor de også er implementeret som tre selvstændige klasser. At opnå den effekt, at den ene tager over, hvor den anden fejler, som er en del af Phelps og Wilenskys strategi, overlades i stedet til frameworket (afsnit 3.3), men baseres på de rangeringer eller vægtninger, som beskrives i kilden.

4.1.1 Unikt id

Unikt id-metoden er beskrevet side 18, afsnit 2.5.1.

Lokationsspecificering

Dette er den simpleste af metoderne, da den blot gemmer et eventuelt id på det element, der indeholder lokationen. Ofte vil lokationen i DOM-træet ligge i et såkaldt `#text` element, hvilket er et slags usynligt lag, der indkapsler tekst-data. For eksempel vil `<p>data</p>` bestå af et `p`-element, med et

barn af typen `#text`, der så vil indeholde teksten "data". `#text`-elementer kan ikke have et id, da de ikke er specificeret i dokumentets kode, men kun introduceres i DOM-træet. Det kan `p`-elementet godt, og derfor vil det tit være nødvendigt at kigge i forældre-elementet, for at finde id.

Alternativt, hvis heller ikke den nærmeste forældre har et id, kan et element højere oppe i træet have et id. Jeg har valgt, at et sådant id kan accepteres for lokationen, hvis lokationen er en startlokation og ligger i en lige linje af firstChilds, eller det er en slut-lokation og ligger i en lige linje af lastChilds, fra det element, der har et id.

Dette er ikke en del af specifikationen, men da unikt id i forvejen ikke specificerer mere præcist end på element-basis, vil denne forskel ikke have nogen reel betydning, da der ikke er en tekstuel forskel på startpunktet af et element og startpunktet på dette elements firstChild.

Koden, der udtrækker id'et for en startlokation er skitseret i Algoritme 1, hvor det antages, at koden kaldes rekursivt med `Recurse()`.

Algoritme 1 Unikt id - find id

if *node.hasAttributes* AND *node.id* $\neq$ null **then**
  return *node.id*
else if (*type* == 'start' AND *node* == *node.parentNode.firstChild*)
OR (*type* == 'end' AND *node* == *node.parentNode.lastChild*) **then**
  return *Recurse*(*node.parentNode*, *type*)
end if

Relokalisering

Phelps og Wilenskys specifikationen fortæller blot, at hvis id'et for en lokation findes i dokumentet, antages lokationen at være fundet. Da mit framework arbejder med ranges og ikke kun enkelt-punkter, har jeg nuanceret denne definition en smule og introduceret rangering for denne metode, hvilket ellers var udeladt af Phelps og Wilensky.

For at opnå en rangering på 100 skal både start-id og slut-id være specificeret og fundet, og det returnerede range strækker sig fra starten af start-elementet til slutningen af slut-elementet. Hvis kun ét af id'erne er specificeret og findes i dokumentet, returneres et range fra starten til slutningen af dette element. Her gives en rangering på 50.

Hvis der hverken kan findes et start- eller slutelement, returneres et null-range og en rangering på 0.

For at finde et eventuelt element bruges DOM-Dokument-metoden `getElementById()`:

```
startNode = dom.getElementById(start.getDescription());
```

hvor *dom* er DOM-dokument-objektet for den side lokationen ønskes fundet i, og *start* er det LocSpec-objekt, der har beskrivelsen af start-lokationen.

Afvigelser fra specifikationen

Udover eventuelle forskelle, der er opstået på grund af manglende detaljer i beskrivelsen, er der i unikt id-metoden to reelle forskelle mellem implementationen og specifikationen.

Dels tillader jeg brug af forfædres id, så længe lokationen ligger i det første blad-element i forfaderens undertræ for start-lokationer og det sidste blad-element for slut-lokationer. Dette er ikke beskrevet af Phelps og Wilensky, men giver ikke nogen reel forskel for lokationen, det øger blot chancen for at finde et id.

Dels har jeg indbygget en rangering af eventuelt fundne lokationer. Phelps og Wilensky nævner slet ikke rangeringer for unikt id-metoden. Dette skyldes, at de kun opererer med "fundet" og "ikke fundet" og kun fortsætter med næste metode, hvis en lokation ikke er fundet. I mit framework benyttes alle metoder, og den "bedste" lokation udvælges på baggrund af rangeringerne. Rangeringen på 100 for en funden start- og slut-lokation reflekterer dog Phelps og Wilenskys strategi, idet resultatet fra de øvrige metoder - tree-walk og kontekst - da kun vil komme i overvejelse, såfremt de også rammer en rangering på 100.

Kun den første af disse to forskelle har betydning for metodens operation og resultat, men det bør i de fleste tilfælde være en forbedring.

4.1.2 Tree-walk

Tree-walk-metoden er beskrevet side 19, afsnit 2.5.1.

Lokationsspecificering

Kilden baserer sine tree-walk-stier på en reduceret udgave af W3Cs DOM, som kaldes SGDOM - Simple General Document Object Model. SGDOM udelader i forhold til DOM visse stil- og beskrivelses-elementer som `a`, `b` og `span` elementer og inkluderer kun strukturelle elementer. Denne reduktion af DOM'en har jeg ikke implementeret, hvorfor de stier, jeg genererer, i nogen grad bliver mere afhængige af sidens visuelle fremtræden end tiltænkt.

Konstruktionen af stien tager sit udgangspunkt i det element, der indeholder lokationen, og itererer herfra op gennem DOM-træet til rodelementet nås.

For hvert skridt i stien skal elementets navn samt nummer i sit forældre-elements børnerække bruges. Derfor itereres forældre-elementets børn, indtil det aktuelle elementet er fundet, hvorefter nummeret og element-navnet føjes til stien. Dette foregår i funktionen `rootToThis`, der er gengivet i pseudo-kode i Algoritme 2.

Algoritme 2 Tree-Walk - generer sti

```
current ← end
parent ← current.parentNode
while parent.parentNode ≠null do
  if parent.hasChildNodes then
    for i = 0 to parent.childElementCount do
      if parent.childNodes[i] = current then
        path.append(i +′ /′ + current.nodeName)
        break;
      end if
    end for
  end if
  current ← parent
  parent ← current.parentNode
end while
```

Den sti, som `rootToThis` genererer, er første del af lokationsbeskrivelsen for en tree-walk-LocSpec, som altså beskriver stien gennem træet til det element, der indeholder lokationen.

Men tree-walk-beskrivelsen indeholder også en beskrivelse af lokationen i selve medieindholdet i elementet. I dette speciale har jeg kun kigget på muligheden for dette, hvor der er tale om tekst - ligesom det er gældende for kilden. Denne beskrivelse opbygges for ankerets startlokation ved at nummerere ordene fra starten af `range.startContainer` frem til, at den samlede længde af ordene overstiger `range.startOffset`. Da kan beskrivelsen konstrueres som beskrevet i afsnit 2.5.1: $< indeks > / < ord > / < offset >$, hvor *indeks* er ordets nummer i teksten, og *offset* er antallet af tegn inde i det ord, hvor lokationen findes.

Slutteligt sættes beskrivelsen for LocSpec-objektet til "medie-lokations-beskrivelsen + element-stien".

Relokalisering

Relokalisering er trivielt, hvis der ingen ændringer i dokumentet har været. Stien opdeles i skridt, der igen opdeles i et indeks-tal og et element-navn. Så længe elementet med det indeks i børne-rækken har det rette navn, fortsættes til næste skridt ned i træet. Når sidste skridt nås, har vi det element, vi søger. Herefter mangler bare offsettet, som opnås ved at løbe gennem ordene indtil ord-indekset er nået, hvorefter summen af længden af de tidligere ord og offsettet ind i det sidste ord udgør resultatet.

Lykkes det ikke at finde et barn af den rette type med det rette indeks i et elements børnerække, træder en serie af korrigeringer i kraft.

Først søges efter et element af den korrekte type i børnerækken - korrigering 1. Der søges ud fra det korrekte indeks, så der findes det nærmeste element herpå. Hvis et element af den rette type findes, fortsætter algoritmen ned i træet fra dette element med den resterende del af stien. Denne metode vil korrigere stien således, at der tages højde for, at der i dokumentet er indsat eller fjernet en tidligere søskende til det søgte element.

Hvis det ikke lykkes at finde et matchende undertræ ved hjælp af korrigering 1, forsøges i stedet at søge i alle børn efter det samme (indeks, type)-par, som der blev brugt ved dette skridt - korrigering 2. Dette svarer til at ignorere det aktuelle niveau i træet og vil tage højde for et ekstra element indsat på dette sted i stien.

Lykkes det ej heller at finde et matchende undertræ ved korrigering 2, er sidste forsøg at søge gennem børnerækken efter et sted, hvor resten af stien passer på undertræet - korrigering 3. Denne korrigering vil rette stien, hvis et element er skåret ud af stien, eller hvis det er ændret til en anden type.

Relokaliserings-algoritmen kaldes rekursivt, hvilket vil sige, at der på hvert trin i stien kan foretages en korrigering, så algoritmen i yderste konsekvens kan bevæge sig temmelig langt væk fra den oprindelige sti. Dette er meningen, men Phelps og Wilensky har indbygget en begrænsning: For hver korrigering reduceres løsningens rangering en smule. Hvis rangeringen når under et vist niveau, afbrydes søgningen dybere i træet fra det punkt. De har dog ikke angivet, hvor denne grænse går, og de har heller ikke sat nogle tal på, hvor meget en afvigelse fra stien skal koste. Det har i implementationen derfor været nødvendigt selv at opsætte nogle kriterier. De er som følger:

- En helt korrekt sti vil give en rangering på 100
- En korrigering 1 koster: 3 for hver position, der korrigeres
- En korrigering 2 koster: 4 + 3 * max antal korrigering 1
- En korrigering 3 koster: 4 + 3 * max antal korrigering 1

- Det tilføjes en omkostning på 4, for hvert korrigeret skridt i stien
- Hvis rangeringen når under 15, vil stier i undertræet fra dette punkt blive forkastet

Disse kriterier betyder, at der skal ske korrigeringer på en del skridt i stien, før den forkastes, samt at en enkelt korrigering alene vil give en relativt god rangering. Korrigeringer af type 2 og 3 vil være noget dyrere end korrigeringer af type 1. Dette svarer til beskrivelsen i Robust Intra-Document Locations [12, s. 110], men er som nævnt sandsynligvis ikke de samme værdier, som forfatterne har tiltænkt.

Relokaliseringen foretages både for start- og slut-lokationsbeskrivelsen og på baggrund af de eventuelt fundne elementer dannes et range, der repræsenterer det fundne anker. Rangeringen for ranget er 100 fratrukket summen af straffene opnået under relokationen af start og slut. Hvis der findes både en start- og slutlokation, returneres et null-range med rangering 0.

Afvigelser fra specifikationen

Indledningsvist nævnte jeg, at jeg ikke reducerer DOM'en til Phelps og Wilenskys SGDOM. Dette er vel nok den væsentligste forskel, da man kan argumentere for, at elementer som `b`, `i` og `u`, som alene beskriver tekstens formatering, oftere kan blive indsat og fjernet end de rent strukturelle elementer. Omvendt benyttes et element som `span` også ofte i et strukturelt øjemed uagtet, at det måske ikke er semantisk korrekt. Uanset om det styrker eller svækker de stier, algoritmen laver, er det ikke forhindringer, som metodens reparations-strategier ikke kan overkomme.

På et par punkter er kilden ikke helt præcis, hvad angår den nøjagtige implementation. Et sådant eksempel er udregningen af rangeringer for korrigeringer, der foretages under relokaliseringen, som beskrevet ovenfor.

Slutteligt kan algoritmen, som den er skruet sammen, risikere at returnere en lokation med en lavere rangering end den bedst mulige. Det kan ske, hvis en korrigering af type 1 finder en lokation - efter adskillige senere korrigeringer af stien - der har en rangering på for eksempel 25, hvor en korrigering af type 2 måske kan finde et eksakt match på resten af undertræet og derfor ville have givet en lokation med en rangering omkring måske 80. Dog kræver sådan en situation en vis lighed i dele af undertræerne - men det er uklart, hvor stor lighed, der er nødvendig for at undgå at falde for rangeringsgrænsen. Dette er dog ikke en egentlig afvigelse fra Phelps og Wilenskys version, da de kun forsøger at finde en lokation så tæt på den

oprindelige som muligt - og det er som beskrevet ikke garanteret, at det er den, der giver den bedste rangering.

Pånær inklusionen af formatterings-elementer i DOM'en, kan det altså antages, at min implementation ligger meget tæt på den originale version.

4.1.3 Kontekst

Kontekst-metoden er beskrevet side 21, afsnit 2.5.1.

Lokationsspecificering

Der skal i alt laves fire tekst-strenge for et enkelt range - en pre-kontekst- og en post-konstekst-streng for hver af start- og slutlokationerne. Konstruktionen af de fire strenge foreløber parallelt, bortset fra variation i enten udgangspunkt eller retning. I det følgende beskrives, hvordan pre-kontekst-strengen for start-lokationen findes.

Her tages udgangspunkt i lokationen, der er "`range.startOffset` tegn" inde i elementet `range.startContainer`. Herfra tælles bagud i tegnene i elementets tekst-indhold, og hvert tegn lægges til pre-kontekst-strengen. Hvis det første elements tekst-indhold er for kort til at opnå en specificeret kontekst-længde, fortsættes i det foregående blad-element i træet. Algoritmen er skitseret i Algoritme 3.

Det foregående element i træet findes ved først at kigge efter en foregående søskende. Hvis en sådan haves, bruges den. Hvis ikke, undersøges det, om forældre-elementet har en foregående søskende. Hvis ikke, fortsættes søgningen opad i træet efter en foregående søskende ind til en sådan findes. Når den er fundet, findes elementets sidste blad ved at følge lastChild ned gennem DOM-træet, til bladet er nået.

Hvis der ikke kan findes et foregående element, må pre-konteksten forblive i den længde, den har, fordi lokationen i så fald ligger så tæt på dokumentets start, at der ganske enkelt ikke er mere forudgående tekst.

Algoritme 3 Kontekst - find kontekst

while $missingContext$ **do**
  $element \leftarrow getPreviousElement(element)$
  $addLength \leftarrow MIN(element.textContent.length, missingContext)$
  $context \leftarrow element.textContent.subtring(end - addLength, end)$
  $missingContext- = addLength$
end while

Efter pre- og postkontekst er fundet, konverteres de med JavaScripts `escape`-funktion, hvorefter lokationsbeskrivelsen udgøres af konkatenationen af de to separeret af et ikke-escapet mellemrum.

Relokalisering

Relokalisering foregår ved at søge i dokumentet efter pre- og post-kontekst-strengene. Søgningen tager i kilden ikke sit udgangspunkt i starten af dokumentet, men derfra hvor tree-walk-metodens sti maksimalt kunne nå. Da jeg har implementeret metoderne som selvstændige LocSpecMethods, kan et sådant samarbejde på tværs af metoderne ikke lade sig gøre. I stedet søges i min implementation blot fra starten af dokumentet. Det giver det handicap, at hvis teksten findes flere steder i dokumentet, findes den første lokation i stedet for det sted, der er tættest på den oprindelige lokation.

En forbedring i forhold til den problemstilling kunne være at indskrive et dokument-offset i lokationsbeskrivelsen, der skulle angive startstedet for søgningen, men det er ikke implementeret.

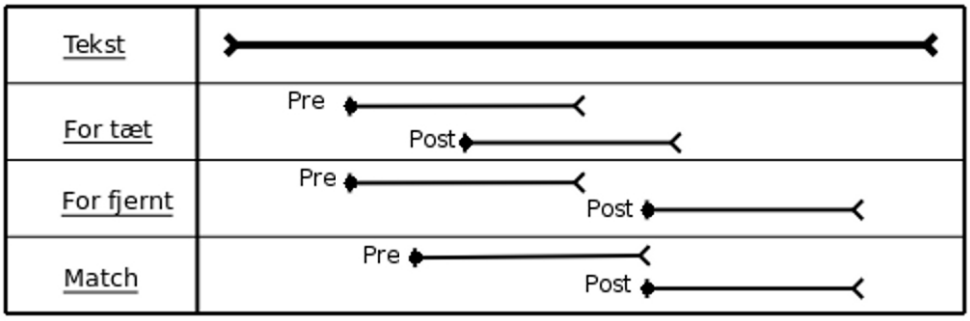
Selve søgningen foregår i en konkatenering af tekst-indholdet af alle blad-elementerne i træet ved hjælp af JavaScript-funktionen `indexOf`. Der søges efter både pre-kontekst og post-kontekst. Hvis en eller begge ikke findes, afkortes begge kontekst-strenge med et ord, og søgningen gentages. Dette fortsætter, ind til kontekststrengene bliver kortere end et fastsat minimum.

Hvis begge kontekst-strenge findes, tjekkes det, at afstanden mellem deres start-indici er præcist længden af pre-kontekst-strengen. Ellers er de enten for tæt på hinanden eller for langt fra hinanden. Hvis de ligger for tæt, søges efter en senere forekomst af post-kontekst-strengen. Hvis de ligger for langt fra hinanden, søges efter en senere forekomst for pre-kontekst-strengen. Situationerne ses i Figur 4.1.

Ovenstående søgning gentages som nævnt, enten til et korrekt indeks er fundet, eller til konteksten bliver for kort. I sidste fald returnerer kontekst-metoden et null-range og en rangering på 0.

Hvis et indeks i sidens tekst findes, skal dette oversættes til et element og et offset ind i dette element. Dette gøres ved at opbygge et map, hvor hver nøgle er en indeks-værdi, og værdien peger på det element, der slutter ved det indeks. Mappets indgange itereres, og når nøglens værdi overskrider det søgte indeks haves elementet. Offsettet findes da ved at trække forskellen på det søgte indeks og nøgle-værdien fra længden på elementets tekst-indhold. Et eksempel på dette ses i Tabel 4.1.

Ovenstående udføres for både start-lokation og slut-lokation, så det endelige resultat bliver to elementer og et offset ind i hver - hvoraf der konstrueres et DOM-Range.



Figur 4.1: Kontekstsøgning. Hvis kontekststrengene overlapper søges efter en senere ostkontekststreng. Hvis der er luft mellem dem, søges der efter en senere prekontekst. Når prekonteskten slutter i samme punkt, som postkonteksten starter, er der fundet et match.

Nøgle	*Element*	*Tekst-indhold*
...	...	...
196	`p`	Tidlig tekst
200	`span`	“Hej ”
205	`b`	“du er”
209	`i`	“ god”
217	`a`	“ med ord”
Indeks: 208		
Første nøgle over indeks: 209		
Element for denne nøgle: `i`		
Offset = 4 - (209 - 208) = 3		

Tabel 4.1: Kontekst indeks-map. Der søges efter det element, der indeholder indeks 208. `i`-elementet starter i indeks 206 og slutter i 209 og udvælges derfor. Det har en længde på fire, så indeks 209 svarer til et offset på tre tegn ind i `i`-elementet.

Der er to variable, der skal fastsættes for metoden, nemlig start-længden på kontekst-strengene og smertegrænsen for hvor korte, de må blive. Phelps og Wilensky foreslår en startlængde på 25 tegn, hvilket jeg også har brugt i implementationen. En større startlængde giver øget præcision i tekster med mange gentagelser, men vil samtidig betyde lavere rangering ved ændringer, da der sandsynligvis skal skæres flere ord fra. Forfatterne angiver ikke et bud på en smertegrænse, så her har jeg valgt en værdi på 5. Jo lavere denne er, jo større er chancen for at finde en lokation - men jo større bliver risikoen for at finde en forkert lokation også.

Kilden beskriver ikke nøjagtigt, hvordan rangering af resultatet bør foretages. Der er to faktorer, der reducerer rangeringen: Dels mindskes den, jo større en del af DOM-træet, der søges i, dels mindskes den, jo mere, der skæres af konteksten. Den første reduktion giver ikke mening i min implementation, da der søges i hele træet fra start. Den anden reduktion er ikke fuldt ud specificeret, idet det ikke oplyses, hvordan der reduceres.

I implementationen reduceres en oprindelig rangering på 100 med 5, for hvert ord, der skæres fra for henholdsvis start- og slutlokationen. Hvis en lokation ikke findes, sættes rangeringen til 0.

Afvigelser fra specifikationen

Den mest signifikante forskel fra specifikationen er, at der ikke tages udgangspunkt i tree-walk-metodens arbejde. Dette skyldes som nævnt, at metoderne er implementeret som selvstændige klasser, der kan bruges uafhængigt af hinanden. Dette påvirker både søgningen - der søges fra starten af dokumentet - og rangeringen, der i kildens version afhænger af, hvor langt fra den oprindelige position, der ledes.

Derudover er der blot nogle små usikkerheder i forhold til for eksempel smertegrænse for kontekst-længden og reduktionen af rangeringen i forbindelse med kontekst-afkortningen. Det vigtigste i forhold til rangeringen er dog at opnå balance i forhold til de øvrige metoder. Den valgte værdi er fundet efter nogle mindre, indledende tests.

Alt i alt vurderer jeg, at min implementation ligger tæt nok op ad kildens version til at resultaterne er repræsentative.

4.1.4 Det samlede system

Selvom der på implementations-niveau er tale om tre individuelle LocSpec-Methods, vil de blive testet som et samlet system, da de er beskrevet som sådant af Phelps og Wilensky. De er designet til at virke på forskellige akser - unikt id på en forfatter-baseret, semantisk akse, tree-walk på en strukturel

akse og kontekst på en indholds-akse. Dette giver dem en samlet styrke, der er svær at opnå ved blot at benytte en enkelt akse.

I frameworket vil alle tre metoder blive bedt om deres bud på en relokalisering, og den højest-rangerede vil blive valgt. Det har derfor været vigtigt at få rangeringen til at virke nogenlunde balanceret. Dog skal det gentages, at i Phelps og Wilenskys system vil den først fundne lokation altid blive valgt.

Jeg har valgt deres model fra, da jeg implementerede frameworket, idet målet med frameworket i vid udstrækning var, at det skulle kunne give de bedst mulige resultater i en senere, virkelig anvendelse. Jeg mener, min fremgangsmåde i frameworket er mere hensigtsmæssig i et setup, hvor der måske anvendes mange flere LocSpecMethods, da alle så kommer til orde. Dog anerkender jeg, at der kan blive et balancerings-problem, hvis metoderne konstrueres uafhængigt af forskellige udviklere. En forbedring i frameworket kunne være et analyse-modul, der, givet en række ranges, returnerer det enkelte range, der mest præcist dækker de øvrige. Det var en oprindelig plan i frameworket, men den er endnu ikke realiseret.

I mange tilfælde vil frameworket give samme resultat som Phelps og Wilenskys version. Kun hvis en senere metode giver en løsning med højere rangering end en tidligere ikke-nul-løsning, vil resultatet afvige. Da unikt id ofte vil give 0 eller 100 i rangering, vil denne løsning som regel enten blive brugt eller blive skippet også af Phelps og Wilensky. Afvigelserne vil derfor typisk ske, hvis både tree-walk- og kontekst-metoderne finder en løsning, og kontekst-metodens løsning er bedre end tree-walk-metodens. Som allerede nævnt sker denne afvigelse altså efter et bevidst valg, da den bedste løsning ønskes udvalgt.

Hvis ingen af de tre metoder finder en løsning, returneres et null-range med rangering 0. Ellers returneres som sagt det bedste range med dettes rangering.

4.2 Resilient XPaths

Resilient XPaths er beskrevet side 23, afsnit 2.5.2.

Resilient XPaths er navnet på den metode, Paz og Díaz præsenterede i Providing Resilient XPaths For External Adaption Engines [11]. Metoden fungerer ved at bearbejde XPaths for samme lokation i en række eksisterende kopier af et dokument for at opnå et generelt, modstandsdygtigt udtryk. Det er forhåbningen, at det nye udtryk er kogt ind til de essentielle dele, så det efter fremtidige ændringer utvetydigt udpeger den samme lokation.

I kilden opdeles metoden efter to brugsscenarier - rumlige og tidslige forskelle. Den tidslige metode, som er mest relevant for sammenligning med Robust Locations, baseres dog på resultatet af den rumlige metode. Derfor har jeg valgt at implementere dem som én samlet metode. Bearbejdningen af de oprindelige XPath-udtryk gennemløber derfor to faser: Induktionen og simulated annealing.

4.2.1 XPath-konstruktion

Før bearbejdningen af XPath-udtrykkene kan starte, skal de indkommende DOM-Ranges, der hver for hver sit DOM-Dokument repræsenterer det anker, der skal gemmes, først oversættes til XPath-udtryk. I stedet for at arbejde på tekst-repræsentationer af XPath-udtrykkene, repræsenterer jeg dem, indtil de skal gemmes som lokationsbeskrivelser i form af arrays.

Udover at spare en masse streng-operationer giver det den fordel, at arrayet kan konstrueres, så det husker informationer om et skridt i stien; informationer der ellers er generaliseret ud. Dette er nødvendigt under simulated annealing, da der i visse typer transformationer re-introduceres tidligere fjernede informationer. I arrayet huskes således skridtets type, elementets position i børnerækken samt eventuelle attributter.

Konstruktionen forløber i store træk efter algoritmen fra tree-walk-metoden - se Algoritme 2, s. 49. Blot dannes der altså ikke en tekstuel sti, men et to-dimensionelt array, med skridt-nummeret på x-aksen og skridt-informationerne på y-aksen.

4.2.2 Induktionsalgoritmen

Første relle fase i metoden er kørslen af induktionsalgoritmen, hvis opgave det er at smelte de dannede XPath-udtryk sammen til et enkelt således, at alle fællestræk fra de enkelte udtryk bevares, mens forskelle smides væk. Det overordnede forløb af induktionen beskrives af Algoritme 4.

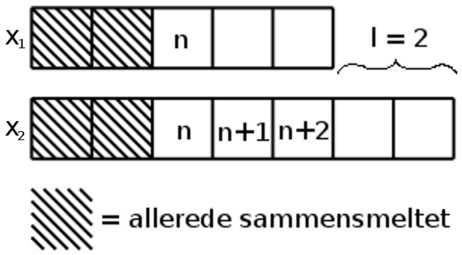
Den egentlige sammensmeltning sker i `merge`-funktionen, der skridt for skridt sammenligner de to XPaths, der skal slåes sammen. Efter forskriften fra afsnit 2.5.2 - Ændringer i rum, foretages de nødvendige modifikationer, og resultatet returneres.

Sammensmeltningen

Det er ikke en helt triviel opgave at smelte to udtryk sammen, da længderne på stierne kan være forskellige. En sådan forskel betyder, at der et sted i

Algoritme 4 Resilient XPaths - Induktion

```
current ← XPaths.shift()
if XPaths.length > 1 then
  next ← Recurse(XPaths)
else
  next ← XPaths.shift()
end if
merged ← merge(current, next)
return merged
```



Figur 4.2: To XPaths, x_1 og x_2, under induktionen. De er ens ind til skridt n, hvorfra algoritmen kigger l skridt frem i den længste af stierne.

stien skal introduceres et "skip" (// i et XPath-udtryk), men at afgøre hvor og hvor mange skridt, der skal skippes, er en kompleks opgave.

I den nuværende løsning, indsættes et skip på den første position, hvor der er en forskel i stierne - hvis længden af stierne er forskellige. Længden af skippet - altså hvor mange skridt fra den længste sti, der udelades - afgøres ved at kigge lidt fremad i stien.

Antag første forskel mellem sti x_1 og sti x_2 er i skridt n, og at x_2 er l skridt længere end x_1. Da vil skridtene i x_2 fra $n+1$ til $n+l$ blive sammenlignet med skridt n i x_1. Hvis der findes et match med skridt m i x_2, skippes skridtene frem til m, så skridt n i resultat-stien bliver //, mens $n+1$ i resultatet bliver lig med n i x_1 og m i x_2. Situationen er skitseret i Figur 4.2.

Hvis der ikke findes et match i de afsøgte skridt, indsættes i stedet et wildcard-skridt i resultatet, og skridt n fra både x_1 og x_2 udelades, såfremt der var tale om forskellige typer skridt. Hvis typerne derimod var ens, og forskellen blot var på positionen, da droppes positions-kriteriet, og i stedet for wildcard-skridtet indsættes et skridt af den pågældende type i resultatet.

Denne implementerede håndtering af stier med forskellige længder er formentlig ikke ufejlbarlig, men den løser almindelige tilfælde med blot et enkelt ekstra indsat lag og ingen andre betydende forskelle fornuftigt.

I kilden beskrives ikke, hvordan disse centrale dele er realiseret, så derfor kan der naturligvis forekomme forskelle i de producerede XPaths fra forfatternes system til min implementation. Sandsynligvis er forskellene ikke af afgørende karakter.

Terminering

Efter induktions-algoritmen er nået gennem alle XPaths, er der opnået et samlet XPath-udtryk, som kan sendes til videre behandling af simulated annealing-algoritmen.

4.2.3 Simulated annealing

Simulated annealing er en randomiseret algoritme, der kan bruges til at finde lokale minimums-energi-tilstande for et objekt i systemet. For resilient XPaths er objektet et XPath-udtryk, og energien skal være lav, når udtrykket er reduceret til få, udtryksfulde dele. Som beskrevet i afsnit 2.5.2 om simulated annealing, fungerer en sådan algoritme ved at udføre transformationer med en vis sandsynlighed baseret på en energi fundet ved en energi-beregnings-funktion.

Med implementationsspecifikke detaljer abstraheret ud ser annealing-algoritmen ud som Algoritme 5.

Algoritme 5 Resilient XPaths - Annealing

while $(energy > FREEZE\_LEVEL)$ AND $(time < ANNEAL\_TIME)$ **do**
  $newState \leftarrow transform(currentState)$
  $newEnergy \leftarrow getEnergy(newState)$
  if $P(energy - newEnergy, temperature(time) > Random[0..1])$ **then**
    $currentState \leftarrow newState$
    $energy \leftarrow newEnergy$
  end if
  $time \leftarrow time + 1$
end while

At få transformationerne til at fungere korrekt er den største udfordring, men med dem på plads, er den helt essentielle del af algoritmen `getEnergy`-funktionen, da det langt hen ad vejen er den, der vælger og vrager mellem de mulige løsninger.

Transformationer

Paz og Díaz nævner på side 74 [11], at deres version beregner samtlige naboer - eller transformationer. Dette virker som en lidt voldsom løsning i mine øjne. I stedet har jeg valgt kun at foretage transformationer, til jeg har et enkelt gyldigt XPath-udtryk. Gyldigt vil sige, at det specificerer præcis det element i alle de sider, der er udgangspunktet for lokationen.

Algoritmen foretager fire operationer:

1. Vælg et tilfældigt skridt i stien
2. Vælg en tilfældig, mulig transformation af det skridt
3. Udfør transformationen
4. Tjek om resultatet er gyldigt i forhold til side-sættet

Der er seks typer transformationer, som beskrevet i Tabel 2.1, side 27, og udførslen af dem forløber rimelig enkelt, da konstruktionen af XPath-udtrykket involverede en liste over mulige attributter og type-navne for hvert skridt.

Det sidste tjek af gyldigheden foregår ved at oversætte array-XPath'en til sin tekstuelle pendant og evaluere udtrykket ved hjælp af DOM'ens `evaluate`-funktion. Hvis der findes mere end et, ingen eller et forkert element i forhold til det forventede, er udtrykket ugyldigt.

Energi-funktionen

Som nævnt er denne funktion essentiel for Resilient XPaths. Paz og Díaz har beskrevet den omhyggeligt, men mangler alligevel en smule detaljer, for at det er muligt at lave en præcis kopi - til gengæld opfordrer de til, at man skræddersyr disse detaljer af metoden til den situation, den skal bruges i.

De inddeler attributter i fire grupper: Stil, størrelser, beskrivelser og indhold. Disse inddeles igen i to hovedgrupper: Stil+Størrelse og Beskrivelse+Indhold. Men at afgøre for eksempel hvilken gruppe en `class`-attribut hører under, overlades til læseren. Den beskriver måske nok elementet, omvendt bruges den oftest til at identificere elementer fra CSS, der beskriver stil og størrelse.

Da det ikke præcist er angivet hvordan i kilden, og da kilden opfordrer til, at man justerer vægtningen, er inddelingen af attributter i kategorierne i min implementation sket på baggrund af nogle få indledende tests og en vurdering af hver enkelt attribut i forhold til grupperne.

Jeg benytter følgende energier:

- Normalt skridt: 20

- Wildcard-skridt: 5
- Skip-skridt: 3
- Stil- og layoutattributter: 6
- Beskrivende og indholdsattributter: 3
- Brug af `text()`-funktionen: 6

Af listen ses, at jeg har lidt flere kategorier end Paz og Díaz. De ekstra er Skip og `text()`. Skip er gratis hos Paz og Díaz, mens `text()` formentlig ryger i kategorien indholdsattributter. Jeg har dog vurderet, at condition singularity er så lav for skip-skridt, og at change likelihood er så høj for `text()`-funktionen, at de bør være dyrere end som så.

Selve beregningen af energien er triviel, når først kategorierne og vægtninging af disse er på plads. Da er det blot at summere vægten skridt for skridt i stien.

Temperatur og P

Helt kort om disse to funktioner, skal det blot siges, at kilden ikke beskriver dem. I stedet har jeg fulgt beskrivelsen i Optimization using simulated annealing [3]. `Temperature`-funktionen transformerer blot $time/ANNEAL\_TIME$ til en værdi mellem 1 og 100. `P` returnerer 1, hvis $energy - newEnergy$ er større end 0, og ellers returneres $e^{(energy-newEnergy)/temperature}$.

Dette har den effekt, at en forbedring af udtrykket - hvor *newEnergy* er mindre end *energy* - altid vælges. Derudover har det den effekt, at jo lavere temperaturen er og jo mere *newEnergy* er større end *energy*, jo mindre bliver sandsynligheden for, at det nye udtryk vælges.

Terminering

Algoritmen terminerer, hvis energien af udtrykket bliver tilfredsstillende lav - en foruddefineret grænseværdi - i Algoritme 5 kaldet $FREEZE\_LEVELS$, stopper algoritmen. Hvis ikke dette sker, stopper den efter et bestemt antal iterationer - i algoritmen kaldet $ANNEAL\_TIME$.

I min implementation er $FREEZE\_LEVEL$ sat til 25 og $ANNEAL\_TIME$ til 1000. Paz og Díaz vurderer, at disse tal er mindre væsentlige for et godt resultat end energifunktionen, og de giver ikke selv nogle konkrete tal [11, s. 74]. Gennem afprøvning har der vist sig nævneværdig forskel efter et skift fra 100 til 1000 i annealing tid og en smule bedre resultater ved endnu højere tider.

Når algoritmen terminerer, returneres den aktuelle XPath.

Normalisering

Som sidste led i konstruktionen af lokationsbeksrivelsen, skal XPath-udtrykket oversættes til tekstuel form og "normaliseres". Normaliseringen er nødvendig, da de foregående processer kan have lavet udtryk, der enten bør reduceres eller slet ikke giver mening uden efterbehandling som for eksempel `///p`.

4.2.4 Relokalisering

Da Resilient XPaths udfører stort set alt arbejde under udarbejdningen af lokationsbeskrivelsen, er relokaliseringen meget enkel. DOM'ens `evaluate` metode benyttes til at finde det første element, som henholdsvis start- og slutbeskrivelsen udpeger. Disse bruges som start- og slutelement i det returnerede DOM-Range. Hvis et af disse elementer ikke kan findes, returneres et null-range.

Rangering er 100, hvis der findes et range, og 0 ellers.

4.2.5 Forskelle fra kilden

Paz og Díaz fremhæver, at et element-skridt - udover via elementets navn og position blandt sine søskende - også kan beskrives ved hjælp af andre egenskaber. De nævner antal børn, tekst-indhold og attributter - og skriver i en fodnote, at det også eksempelvis kunne være det at have en foregående søskende, der indeholder et billede med en højde på en pixel ([11, s. 70]).

De skriver dog ikke hvad, der er implementeret i deres løsning. Min implementation inkluderer brug af element-navn, position, `count(*)`, `count(<tag>)`, `text()` samt eksplicitte attributter. Disse er udvalgt, da de enten kan hentes direkte fra DOM-elementet eller bereregnes uden større omkostning. Mere eksotiske udtryk, som det Paz og Díaz angiver i deres fodnote, vil formentlig sjældent finde anvendelse. Man kunne dog forestille sig, at sådanne udtryk kunne eftersøges i situationer, hvor der ellers mangler deskriptorer til unikt at udtrykke lokationerne.

Sammensmeltningen af XPaths er ikke triviel, når stierne ikke er lige lange. I kilden beskrives håndteringen af dette tilfælde kun med et simpelt eksempel. Ideelt set skal sammensmeltningen ske, sådan at så store fælles stier som muligt bevares, men der er forskellige mulige fremgangsmåder. Jeg er i udførslen af testene ikke stødt på situationer, hvor min implementation af dette har ført til problemer - men den giver sandsynligvis ikke helt samme output som forfatternes oprindelige løsning.

Simulated annealing er på mange måder en enkel algoritme, så derfor formoder jeg, at min implementation i store træk minder om Paz og Díaz'.

Dog har de ikke angivet, hvorledes de reducerer temperaturen, eller hvordan de beregner sandsynligheden for at en forværrende transformation gennemføres. Jeg har fulgt opskriften fra Optimization Using Simulated Annealing af Brooks og Morgan [3, s. 243], men Paz og Díaz nævner i øvrigt, at det ikke er en central del af algoritmen.

Udførslen af transformationerne er også underspecificeret i kilden. Paz og Díaz skriver, de finder samtlige naboer, hvilket som nævnt virker som en lidt voldsom løsning. Min fremgangsmåde er at foretage tilfældige transformationer til en gyldig variation af XPath-udtrykket er fundet.

I forhold til energiberegningen, har jeg valgt at tildele en energi til "skipskridt" jævnfør afsnit 2.5.2. Jeg har også valgt at tildele en særlig høj energi for brug af `/text()` i udtrykket, da change likelihood for denne synes meget høj, selvom den også har en høj condition singularity.

Alt i alt er der en del forskelle fra kilden. Dog er flere af forskellene på punkter, hvor forfatterne enten opfordrer brugere af metoden til at prøve sig frem, eller forklarer, at det er mindre væsentligt for metodens funktion. De indledende tests, giver da også foranledning til at tro, at metoden i vid udstrækning fungerer korrekt.

4.3 TextFinder

TextFinder er mit eget bud på en LocSpecMethod. Den er opstået på baggrund af erfaringerne med implementationerne af og de tidlige testresultater fra de øvrige metoder. Men den primære inspiration kommer fra Robust Annotation Positioning in Digital Documents [4], hvor der i diskussionsafsnittet konkluderes at "*Surrounding Context is Less important*" - [4, s. 291], og hvor en deltager fra en undersøgelse udtaler "*The key words are there, it should have been able to somehow connect that sentence [in the modified version] with the original*".

Metoden husker hele anker-teksten, men ingen omkringliggende tekst. Relokalisering foregår ved at lede efter det stræk i dokumentet, der har den højeste koncentration af nøgleordene.

4.3.1 Lokationsspecificering

Der er ikke mange ord at sige om lokationsspecificeringen. Metoden gemmer blot det tekststykke, som anker-ranget spænder over. Det bliver gemt i både start- og slut-LocSpec, men det er egentlig ikke nødvendigt, da det tekststykkerne er identiske.

4.3.2 Relokalisering

Relokaliseringen starter på samme måde, som Phelps og Wilenskys kontekstmetode. Først opbygges nemlig et indeks-map som det, der er illustreret i Tabel 4.1, s. 54. Dette bruges til senere at finde det element, som et givent indeks peger på. Først skal dette indeks dog beregens.

Næste skridt i processen er at opbygge en frekvens-liste for ordene i lokationsbeskrivelsen. Listen indeholder både frekvensen af ordet i lokationsbeskrivelsen og i det dokument, lokationen skal findes i. Endvidere konstrueres en liste bestående af alle de indici i dokumentet, hvor ordet findes.

Baseret på denne frekvensanalyse opbygges tre andre ordlister: Én med ubetydelige ord, én med unikke ord og én med de resterende ord.

Ubetydelige defineres i metoden som ord, der forekommer mere end dobbelt så mange gange i dokumentet som median-frekvensen for nøgleordene. Unikke ord er ord, der kun optræden ét sted i hele dokumentet. De unikke ord udgør sammen med de resterende ord nøgleord for lokationsbeskrivelsen.

Den egentlige søgning i dokumentet kigger efter forekomsten af alle disse nøgleord i "vinduer" af en vis længde startende fra hver af nøgleordenes positioner. Søgningen ses i let forenklet version i Algoritme 6.

Algoritme 6 TextFinder - ankersøgning

```
for startPos in Unique ∪ Other do
  badness ← 0
  for otherPos in Unique do
    if otherPos > startPos + anchor.length * 1.25 then
      badness ← badness + uCost
    end if
  end for
  for otherPos in Unique do
    if otherPos > startPos + anchor.length * 1.25 then
      badness ← badness + oCost
    end if
  end for
  posList.push(startPos, badness)
end for
```

Længden på vinduet, hvori der kigges efter nøgleordene, er defineret som 125 % af længden på lokationsbeskrivelsen. Denne værdi er fundet efter nogle få indledende tests. Den reflekterer en balancering mellem risikoen for at finde forkerte lokationer (ved for stor længde) og at udelade lokationer

med tilføjet indhold (ved for lav længde).

I algoritmen beregnes *uCost* og *oCost*, således at *uCost* er dobbelt så stor som *oCost*, og således, at hvis ingen af de senere positioner ligger inden for vinduet, vil *badness* blive 100. De unikke ord tillægges altså større betydning end de mere almindelige, mens ordene, der endte med deklarationen ubetydelig slet ingen indflydelse har.

Det skal bemærkes, at listen med de resterende ord, i algoritmen *Other*, indeholder flere positioner for hvert ord. Prisen *oCost* lægges kun til *badness* en gang for hvert ord, og kun hvis alle positionerne for det ord ligger for langt fra startpositionen.

Når søgearbejdet er udført konstrueres et range. Ranget strækker sig fra den bedste startposition til slutningen på den sidste position af et nøgleord i vinduet fra den pågældende startposition.

Dette range returneres fra metoden, sammen med en rangering på 100 minus startpositionens *badness*. Dog returneres et null-range og en rangering på 0, hvis *badness* for den bedste position er større end 75.

4.3.3 Styrker og svagheder

TextFinder er fri for alle afhængigheder af struktur og omkringliggende indhold. Dette er en styrke, da der således kan foretages alle de ændringer, det skal være i dette, uden det har nogen betydning for TextFinder. Det er også en svaghed, da det medfører total afhængighed af den tekst ankeret spænder over. TextFinder kan modstå en del omrokeringer, sletninger og tilføjelser af ankerteksten og må først give op, når 75 % af nøgleordene[1] forsvinder ud af det definerede vindue.

Et anker med mange almindelige ord fra en tekst med en meget "flad" sprog-brug, vil risikere at finde en forkert lokation, da nøgleordene måske vil kunne findes i nærheden af hinanden flere steder i teksten.

I situationer, hvor ankeret for eksempel er et billede, banner eller andet ikke-tekstuelt indhold, vil metoden fejle, da den er bundet til tekst-mediet. Hvis anker-teksten er meget kort, for eksempel et eller to ord, vil det også kunne medføre problemer, hvis ikke ordene er unikke i dokumentet.

4.4 Opsummering

I dette kapitel har jeg detaljeret gennemgået mine implementationer af henholdsvis Phelps og Wilenskys Robust Locations og Paz og Díaz' Resilient

[1] Antaget at der ikke findes nogle unikke nøgleord. Findes de, vil procent-satsen afhænge af hvilke typer ord, der forsvinder.

XPaths. Jeg har fremhævet de forskelle og usikkerheder, der er i forhold til kilderne, men argumenteret for, at metoderne trods disse forskelle kan antages at være repræsentative for metoderne.

Jeg har endvidere præsenteret min egen metode, TextFinder, der er baseret på erfaringer fra de øvrige metoder og inspiration fra Brush *et al.* [4]. Jeg har desuden givet en gennemgang af metodens implementation og påpeget de styrker og svagheder, jeg umiddelbart ser ved metoden.

I næste kapitel vil metodernes virke i de forskellige testcases blive beskrevet og sammenlignet.

5

Resultater

I dette kapitel gives en kort beskrivelse af de test, jeg har foretaget samt resultaterne fra disse. Testene er opdelt i typerne omformulering, anker-flytning, paragraf-flytning, sletning og wiki-redigeringer. Den sidste, wiki-redigeringer, dækker over typiske redigeringer af artikler fra wikipedia.org[1], og der vil her være tale om reelle ændringer på siderne foretaget af brugerne gennem en længere periode.

De fire første er ændringstyperne identificeret af Brush *et al.* [4] (se afsnit 2.1.3, s. 7). Her er test-casesne konstruerede for at sikre, at det er den rette type ændringer, der er foretaget fra version til version. Disse test-cases er dannet på baggrund af en kopi af Seth's blog[2], der dog er blevet rettet til, så den overholder XHTML-standarden[3].

Alle test er udført med Phelps og Wilenskys submetoder både enkeltvist under deres egne navne og samlet under navnet Robust Locations. Desuden er testene udført med Paz og Díaz' Resilient XPaths og min egen TextFinder.

5.1 Sådan læses resultaterne

Hver testkørsel giver to rækker til en tabel, der ser ud som i Tabel 5.1. Første række er absolutte tal for testen, mens anden række angiver procentdele for testen.

Test	Forsøg	Succes	Fejl	$\surd$ udeladt	$\surd$ lok.	$\sim$ lok. (kvl)	$\div$lok. (dst)	$\div$udeladt
Metode	60	40	20	2	30	8 (25 %)	14 (10 %)	6
	100 %	66.6 %	33.3 %	100 %	51.7 %	13.8 %	24.1 %	10.3 %

Tabel 5.1: Eksempel på testresultater

Første kolonne angiver navnet på metoden, der er brugt til testen. Anden kolonne angiver det samlede antal relokationer, der er forsøgt. Fem lokationer, der er forsøgt fundet i seks dokumenter giver i alt 30 forsøg. Tredje og fjerde kolonne (*Succes* og *Fejl*) angiver, hvor mange af disse forsøg, der er henholdsvis accepterede og ikke accepterede.

[1] `http://en.wikipedia.org/wiki/Main_Page`

[2] `http://sethgodin.typepad.com/seths_blog/`, "*his blog is perhaps the most popular in the world written by a single individual*" - `http://www.sethgodin.com/sg/bio.asp`. I hvert fald var det den første blog, der dukkede op ved en søgning på ordet "blog".

[3] `http://www.w3.org/TR/xhtml1/` - en W3C standard.

Femte kolonne, $\surd$ *udeladt*, angiver i hvor mange tilfælde metoden korrekt har undladt at angive en lokation. Altså hvor lokationen er slettet, og metoden ikke har fundet en ny, forkert, lokation.

I sjette, syvende og ottende kolonne kan aflæses, hvor mange af de lokationer, metoden har fundet frem til, der er henholdsvis korrekte, delvist korrekte og forkerte. Niende og sidste kolonne, $\div$*udeladt*, fortæller, hvor mange lokationer, metoden har måttet opgive at give et bud på, vel at mærke hvor en lokation burde findes.

De to tal i parenteser, *kvl* og *dst*, er gennemsnittene for de kvalitetsvurderinger, der blev foretaget af test-suiten for henholdsvist delvist korrekte og forkerte lokationer.

Baggrunden for tallene

Som nævnt i afsnit 3.4.4, s. 43, beregner test-suiten de rå resultater baseret på en liste af manuelt udpegede lokationer. Det giver tallene for korrekte, delvist korrekte og forkerte lokationer, samt korrekte og forkerte udeladelser.

Af disse tal udledes de øvrige: De accepterede løsninger, kolonnen *succes*, er summen af korrekte og delvist korrekte lokationer, samt korrekte udeladelser. De ikke accepterede løsninger i *Fejl*-kolonnen er summen af de forkerte lokationer og forkerte udeladelser.

Procent-satserne beregnes for hver kolonne i forhold til det maksimalt opnåelige. Det vil sige, at *Succes* og *Fejl* er vurderet i forhold til *Forsøg*. $\surd$ *udeladt* er beregnet i forhold til, hvor mange lokationer, der reelt er slettet - hvilket ikke er vist i tabellen. De resterende er bergnet i forhold til, hvor mange lokationer der *ikke* er slettet - altså antal forsøg minus antal slettede lokationer.

Kvl og *dst* er gennemsnitstal, og gennemsnittene er beregnet i forhold til henholdsvis delvist korrekte lokationer og forkerte lokationer - altså ikke i forhold til alle tests.

Vurdering af tallene

Det bedste resultat vil selvfølgelig være, at alle løsninger er enten korrekte lokationer eller korrekte udeladelser. Jeg har valgt at medtage delvist korrekte løsninger som accepterede også, da nogle metoder, for eksempel Resilient XPaths, opererer på element-niveau. Derfor vil de kun give præcise løsninger, når et anker nøjagtigt spænder over et enkelt element, hvilket sjældent er tilfældet. Desuden vil en delvist korrekt lokation formentlig ofte give mening for en eventuel slutbruger.

Når metoden fejler, sker det enten ved at finde en forkert lokation eller ved helt at opgive at finde en lokation. Jævnfør Brush *et al.*'s brugerundersøgelser (se afsnit 2.1.3, s. 7) vil en udeladelse fremfor en fejlagtig placering ofte være at foretrække. Bemærk, at jeg, som angivet i det refererede afsnit, ikke forholder mig til, om ændringens størrelse ændrer den semantiske betydning så meget, at selv en "korrekt" relokalisering bør udelades. Det ville i givet falde have krævet en viden om, hvilket formål ankeret skulle tjene samt en dybere vurdering af ændringens grad.

Kvaliteten af en delvist korrekt løsning skal selvfølgelig helst nærme sig 100 %, da det betyder et stort overlap. Hvor stor en procentdel, der skal til, før resultatet er hæderligt, er situationsbestemt: 25 % kan være fint for et kort anker, mens et langt anker måske skal overlappe 50 eller 75 % for at give mening. På samme måde gælder det for en forkert løsnings afstand til den korrekte lokation, at den helst skal være 0 %. I korte dokumenter kan større værdier måske nemmere accepteres end i lange dokumenter. Fælles for disse to procentsatser er, at de kan være nyttige til at sammeligne resultater inden for en test, men at de er svære at opstille nogle absolutte mål for.

Med disse anmærkninger in mente er tid at kigge på de egentlige resultater.

5.2 Type-test

I dette afsnit testes metoderne på små datasæt, hvor der i hvert datasæt kun er foretaget en enkelt type ændring. Formålet med disse test er at identificere generelle styrker og svagheder ved metoderne. Jeg har derfor ikke fokuseret på at gøre testene fuldstændigt repræsentative for ændringstyperne, og der vil findes ændringer, der ikke bliver modsvaret i disse test.

5.2.1 Omformulering

Omformulering dækker over ændringer i ankerets ordlyd - men ikke placering. Det kan være en rettelse af en stavefejl eller en komplet omskrivning af sætningen. Denne test vil give et indblik i metodernes afhængighed af det tekstuelle indhold i ankeret. Dårlige resultater vil være meget bundne af teksten, mens gode resultater vil opnås af strukturelt funderede metoder.

Testdata

Som nævnt i kapitlets indledning tager testen udgangspunkt i en enkelt blog[4]. Der er udvalgt en enkelt lokation i bloggen, der bliver gemt som anker. Efterfølgende er der blevet lavet følgende modifikationer i anker-teksten:

Oprindelig: Promise a group of six managers that one of them will get promoted in six months and watch the energy level rise

Variation 1: **Promise a group of six** manegers **that one of them will get promoted in** two **months and watch the energy level rise**

Variation 2: To make your **managers** work harder, it might pay off to tell **them** you **will** promote **the** best **of them in** half **a** year

Variation 3: Your **managers** work efforts on **a** short to medium term **will** to some extend depend on the prospect of a promotion

Variation 4: Deceive your **managers** into working harder by hinting a promotion coming up within **a** limited time frame

Variation 5: **Promise a group of** seven **managers** than **one of them** might **get promoted in** a half year **and** see **the energy level rise**

Variation 6: The **one** true motivational remedy you have as CEO is to give your employees **a** cause, **a** goal, something to achieve

I den første variation er der blot introduceret to små ændringer, mens den sidste version blot indeholder to ord fra den oprindelige tekst, nemlig *one* og *a*. De mellemliggende variationer er en mellemting mellem disse to yderpunkter - dog ikke ordnet efter ændringsgraden. Ordene med fed er ord, der optræder i den oprindelige tekst.

Som side-sæt for Resilient XPaths er der brugt to kopier af den originale side med ændringer i samme stil som de, der er nævnt her.

Resultater

I denne test, hvor ankeret ikke flytter sig i dokumentet, står Resilient XPaths rigtig stærkt. Den finder lokationen hver gang, da det benyttede XPath-udtryk er helt uafhængigt af det tekstuelle indhold og peger direkte på lokationen.

[4]Datasættet brugt til denne test findes i test-suiten i biblioteket tests/testA

Test	Forsøg	Succes	Fejl	√ udeladt	√ lok.	∼ lok. (kvl)	÷lok. (dst)	÷udeladt
Unikt id	6	0	6	0	0	0 (0 %)	0 (0 %)	6
	100 %	0 %	100 %	-	0 %	0 %	0 %	100 %
Tree-walk	6	2	4	0	2	0 (0 %)	0 (0 %)	4
	100 %	33.3 %	66.7 %	-	33.3 %	0 %	0 %	66.7 %
Kontekst	6	2	4	0	2	0 (0 %)	0 (0 %)	4
	100 %	33.3 %	66.7 %	-	33.3 %	0 %	0 %	66.7 %
Robust Locs	6	2	4	0	2	0 (0 %)	0 (0 %)	4
	100 %	33.3 %	66.7 %	-	33.3 %	0 %	0 %	66.7 %
Res. XPaths	6	6	0	0	0	6 (23.7 %)	0 (0 %)	0
	100 %	100 %	0 %	-	0 %	100 %	0 %	0 %
TextFinder	6	3	3	0	0	3 (83.7 %)	0 (0 %)	3
	100 %	50 %	50 %	-	0 %	50 %	0 %	50 %

Tabel 5.2: Resultater for test af omformuleringer

Treewalk-metoden, der også er struktur-baseret, burde således også stå stærkt i denne test. Den finder imidlertid kun to af de seks lokationer. Det skyldes, at den trods sin tilknytning til dokumentets struktur i sidste ende er bundet op på at finde præcis det ord, som den oprindelige lokation lå i - og det gælder tilmed for både start- og slutlokationen for ankeret.

Kontekst-metoden finder de samme lokationer som tree-walk-metoden i denne test. Det skyldes to ting. Først betyder pre-kontekst-strengen for start-lokationen og post-kontekst-strengen for slut-lokationen, at det anker metoden eventuelt finder, må strække over præcis det samme stykke, som det oprindelige anker gjorde. Dernæst betyder de to andre kontekst-strenge, der er i spil, at metoden kun finder de to variationer, der starter og slutter med samme ord, som den oprindelige tekst.

Unikt id-metoden finder ingenting, da ankeret ikke udgør et helt element. Havde ankeret udgjort et helt indlæg i bloggen, havde sagen set anderledes ud, da samtlige indlæg i bloggen har et unikt id, hvorfor metoden da ville have fundet lokationen hver gang.

Kombineret finder Robust Location metoderne to af seks lokationer, da tree-walk og kontekst fandt de samme, og unikt id ingen fandt.

TextFinder finder tre af de seks lokationer. Metoden er kun afhængig af tekstindholdet og finder derfor de tre løsninger, der har flest gengangere blandt ordene fra den oprindelige tekst.

For de fleste metoder vil billedet formentlig se anderledes ud, når ankeret flyttes i teksten.

5.2.2 Anker-flytning

Anker-flytning er en ændringstype, hvor anker-teksten forbliver uændret, men flyttes til en ny kontekst.

Umiddelbart må Resilient XPaths forventes at klare testen ringest, da de XPaths, metoden genererer, ikke har nogen håndtag ind i teksten, med mindre det element ankeret ligger i er ens i alle sider i side-sættet, hvorved elementet kan blive specificeret med text()-funktionen.

Phelps og Wilenskys kontekst-metode, må formentlig også give op, da den omkringliggende tekst efter flytningen vil forhindre den i at finde både start og slut på ankeret. Med tree-walk-metoden kan det gå begge veje - en flytning langt væk i dokumentet kan reducere rangeringen for meget, til at at metoden finder ankeret. Lokationer tæt på burde metoden dog finde.

TextFinder, der er helt uafhængig af konteksten, burde finde ankeret i alle tilfælde.

Testdata

Her er brugt samme blog som udgangspunkt for testen som ved Omformulerings-testen, men der er udvalgt et andet anker, og ændringerne er naturligvis nogle andre[5]. Ankerteksten er uændret, men dens position er ændret i dokumentet. Tabel 5.3 viser et XPath-udtryk frem til det element, der inderholder ankeret for henholdsvis de oprindelige og de ændrede sider. De skridt, der er forkortet til `//` i tabellen, er fælles for alle stier. Offset henviser til, hvor mange tegn inde i elementet ankeret begynder.

Original 2 og 3 er brugt sammen med original 1 som basis for Resilient XPaths.

Resultater

Resultaterne af testen svarer fint til forventningerne. Tree-walk, og dermed Robust Locations, samt TextFinder klarer sig godt, mens ingen af de øvrige metoder finder en eneste lokation.

Resilient XPaths kunne formentlig have fået et bedre resultat, hvis ankerteksten udspændte et eller flere elementer helt. I test-casen er ankerteksten en sætning, men der er en sætning både foran og bagved i det `p`-element, teksten ligger i.

TextFinder finder alle lokationer i testen; det eneste, der kunne have forhindret det i en test, hvor ankerteksten ikke ændrer sig, ville have været forekomsten af et meget lignende tekst-stykke tidligere i dokumentet.

[5]Datasættet brugt til denne test findes i test-suiten i biblioteket tests/testB

Side	**XPath**	**Offset**
Original 1	`//div[1]/div[21]/div[1]/div[1]/p[3]`	98
Original 2	`//div[1]/div[19]/div[1]/div[1]/p[2]`	299
Original 3	`//div[1]/div[21]/div[1]/div[1]/p[8]`	393
Ændret 1	`//div[1]/div[21]/div[1]/div[1]/p[11]`	229
Ændret 2	`//div[1]/div[2]/div[1]/div[1]/p[4]`	67
Ændret 3	`//div[1]/div[23]/div[1]/div[1]/p[1]`	1
Ændret 4	`//div[1]/div[1]/div[1]/div[1]/p[4]`	62
Ændret 5	`//div[1]/div[17]/div[1]/div[1]`	405
Ændret 6	`//div[1]/div[2]/div[1]/div[1]/p[1]`	70
Ændret 7	`//div[1]/div[22]/div[1]/div[1]/p[5]`	210
Ændret 8	`//div[1]/div[21]/div[1]/div[1]/p[3]`	64

Tabel 5.3: Ankerpositioner i ankerflytningstesten

Test	Forsøg	Succes	Fejl	$\surd$ udeladt	$\surd$ lok.	$\sim$ lok. (kvl)	$\div$lok. (dst)	$\div$udeladt
Unikt id	8	0	8	0	0	0 (0 %)	0 (0 %)	8
	100 %	0 %	100 %	-	0 %	0 %	0 %	100 %
Tree-walk	8	7	1	0	4	3 (40.7 %)	0 (0 %)	1
	100 %	87.5 %	12.5 %	-	50 %	37.5 %	0 %	12.5 %
Kontekst	8	0	8	0	0	0 (0 %)	0 (0 %)	8
	100 %	0 %	100 %	-	0 %	0 %	0 %	100 %
Robust Locs	8	7	1	0	4	3 (40.7 %)	0 (0 %)	1
	100 %	87.5 %	12.5 %	-	50 %	37.5 %	0 %	12.5 %
Res. XPaths	8	0	8	0	0	0 (0 %)	8 (44.6 %)	0
	100 %	0 %	100 %	-	0 %	0 %	100 %	0 %
TextFinder	8	8	0	0	0	8 (98 %)	0 (0 %)	0
	100 %	100 %	0 %	-	0 %	100 %	0 %	0 %

Tabel 5.4: Resultater for test af ankerflytninger

Tree-walk-metoden klarer sig godt og misser kun en enkelt lokation - den fra siden "Ændret 5". Kombinationen af et fjernet skridt i stien og flytningen i dokumentet samt den store forskel i offsettet ind i elementet medfører, at rangeringen bliver meget lav. I en gentagelse af testen uden rangering fandt metoden korrekt startlokationen, men den slutlokation, den fandt, lå tidligere i dokumentet, hvorfor resultatet stadig blev "ikke fundet".

De delvist korrekte lokationer skyldes, at metoden finder en rigtig startlokation men en forkert slutlokation. "Slutnøgleordet" for metoden er "in.". Da "in." også forekommer et andet sted relativt tæt på den oprindelige lokation, bliver den forkerte lokation i flere tilfælde fundet, når den rigtige er flyttet for langt væk. "Startnøgleordet" er mere unikt, hvorfor samme problem ikke

er opstået for det i denne test.

Kontekst-metoden klarede sig lige så dårligt, som det var forventet. Næste udfordring, paragraf-flytning, ligger til gengæld bedre til denne metode.

5.2.3 Paragraf-flytning

Når et anker bliver udsat for paragraf-flytning, betyder det, at både ankerteksten og den omkringliggende tekst bliver flyttet. Dermed er der gode vilkår for både indholds-baserede og kontekst-baserede metoder. Afhængigt af hvor meget af den omkringliggende struktur, der bliver flyttet med, kan også de strukturelt baserede metoder have en chance.

Test-data

Seth's blog bliver udsat for endnu en række ændringer[6]. Denne gang består ankeret af teksten: "oversold the miles and undelivered on the free flights,". Teksten er i den oprindelige version af siden en del af et `p`-element, som ligger i et `div`-element, der udgør brødteksten af en af indføringerne i bloggen.

For alle flytninger gælder, at teksten i `p`-elementet er urørt. Dette er kriteriet for en paragraf-flytning. Flytningerne består både af flytning af `p`-elementet alene indenfor sin egen blog-indføring og til andre blog-indføringer tidligere og senere i teksten. Der er også to flytninger, hvor hele blog-indføringen er flyttet på siden. Denne sidste type svarer nogenlunde til den bevægelse, der sker på siden, når en ny indføring offentliggøres.

Resultater

Som det ses af Tabel 5.5, klarede alle metoder denne ændring fornemt. Kontekst og TextFinder scorede 100 %, som det kunne forventes uden nogle ændringer i tekst eller kontekst. Mere overraskende er det måske, at tree-walk-metoden også scorer 100 %, men et godt håndtag i ordene "oversold" og "flights,", og lidt af den omkringliggende struktur bevaret, viste sig tilstrækkeligt. Kontekst- og tree-walk-metodernes succes betyder selvfølgelig også at Robust Locations som helhed klarer sig perfekt.

Det kan også undre, at Resilient XPaths ligger så højt, som den gør, når den ikke klarede anker-flytningen bedre. Det skyldes, at teksten i ankerets indeholdende element ved anker-flytningen ændrede sig i det sidesæt, som de modstandsdygtige XPaths blev opbygget ud fra. Ved paragraf-flytningen er teksten i paragraffen uændret, og derfor kan metoden drage fordel at XPath-funktionen `text()`.

[6]Datasættet brugt til denne test findes i test-suiten i biblioteket tests/testC

Test	Forsøg	Succes	Fejl	$\surd$ udeladt	$\surd$ lok.	$\sim$ lok. (kvl)	$\div$lok. (dst)	$\div$udeladt
Unikt id	5	0	5	0	0	0 (0 %)	0 (0 %)	5
	100 %	0 %	100 %	-	0 %	0 %	0 %	100 %
Tree-walk	5	5	0	0	5	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	-	100 %	0 %	0 %	0 %
Kontekst	5	5	0	0	5	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	-	100 %	0 %	0 %	0 %
Robust Locs	5	5	0	0	5	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	-	100 %	0 %	0 %	0 %
Res. XPaths	5	4	1	0	0	4 (9 %)	0 (0 %)	1
	100 %	80 %	20 %	-	0 %	80 %	0 %	20 %
TextFinder	5	5	0	0	5	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	-	100 %	0 %	0 %	0 %

Tabel 5.5: Resultater for test af paragrafflytninger

I et enkelt tilfælde har metoden ikke fundet lokationen, men under en gentagelse af testen, hvor annealing-tiden var øget med en faktor 5, fandt metoden faktisk alle lokationer. Længere annealing tid kan give metoden bedre mulighed for finde en tilstand med meget lav energi.

Metoden baseret på unikt id ramte igen ingen resultater, da anker-teksten ikke tilfulde udgjorde et element med et id. I næste test vil en af lokationerne være konstrueret så den har et id - et `div`-element med en id-attribut, hvorfor metoden bør have mere held med sig der.

5.2.4 Sletning

At afgøre, at noget er væk, er at fejle i at finde det. Metoderne er konstruerede til at finde et anker selv efter talrige ændringer. Derfor kan udfordringen, der består i at afgøre, om ankeret er slettet, være svær. Der er en udpræget risiko for, at metoderne finder lignende passager i stedet.

Især en metode som Resilient XPaths kan komme ud på dybt vand, da den typisk ikke har nogen beskrivelse af indholdet i ankeret. Hvis ikke metoden har hold på indholdet via `text()`-funktionen, vil et efterfølgende element af samme type som det slettede, ofte blive udpeget af metoden.

For tree-walk-metoden består risikoen i at udpege et stykke tekst med samme start- og slutord, der ligger et "lignende" sted i DOM-træet. Kontekst-metoden, der har fat i større bidder af indholdet og den omkringliggende tekst, står med bedre kort. Unikt id-metoden bør have 100 % korrekt udeladte ankre, såfremt den altså har kunnet identificere ankeret inden sletningen.

TextFinder har risiko for at finde et stykke tekst med mange af de samme ord. Sådan et stykke tekst vil typisk have lavere rangering end den oprindelige ankertekst ville give, men når ankeret er slettet, kan mindre

være nok til at blive udpeget. Især for korte ankre i en ensartet tekst vil dette være en risiko.

Test-data

Seth's blog[7] er ingenlunde ensartet i sit sprog, selvom det meste af teksten er inden for emnet marketing. Jeg har valgt tre lokationer i den originale tekst som ankre. De tre lokationer er henholdsvis et brudstykke af en sætning på syv ord, indholdet af et helt `p`-element og indholdet af et `div`-element, der har et unikt id, og der spænder over to `p`-elementer.

I hver af de senere tekster er en, to eller alle tre ankre blevet slettet. Der er syv forskellige kombinationer af sletninger, foruden muligheden helt uden sletninger, der ikke er medtaget. Den vigtigste grund til at have test-cases, hvor ikke alle ankrene er slettet, er at dobbelttjekke, at der ikke er tale om ankre, der aldrig kan findes.

Hver lokation vil være slettet i fire af de senere filer og være uændret i tre, så metoderne skal i alt foretage 21 relokaliseringer i denne test, hvoraf tolv gerne skulle ende med udeladelse.

En sidste overvejelse om test-dataene til denne test var, hvorledes de ekstra sider i side-sættet til Resilient XPaths skulle se ud. Jeg besluttede, at bruge en identisk kopi af den oprindelige side, da den typiske ændring i denne test er sletning, hvilket ikke meningfuldt kan udpeges.

Resultater

Som det ses i Tabel 5.6 klarer alle metoderne sig fint. Alle har 100 % korrekt udeladt pånær Resilient XPaths, der dog har to tredjedele korrekt udeladt.

For at teste unikt id-metoden, var det nødvendigt med en lokation, metoden kunne finde - og det ses også af tabellen, at metoden korrekt fandt lokationen tre gange. Da metoden ikke fandt de to andre lokationer, skal de korrekte udeladelser nærmere læses som fire ud af fire end som tolv ud af tolv, fordi de sidste otte dermed skyldes, at metoden slet ikke har en lokations-beskrivelse for lokationen.

TextFinder kunne være kommet i problemer, men de alternativer, der dukkede op, når den rigtige tekst manglede, faldt for rangeringsgrænsen. I en test med en reduceret grænseværdi (ti i stedet for 25) havde metoden kun to korrekte udeladelser. Balanceringen af denne værdi er altså væsentlig. I en mere ensartet tekst havde det måske været nødvendigt med en endnu højere grænse for at undgå forkert fundne postioner.

[7] Datasættet brugt til denne test findes i test-suiten i biblioteket tests/testD

Test	Forsøg	Succes	Fejl	√ udeladt	√ lok.	∼ lok. (kvl)	÷lok. (dst)	÷udeladt
Unikt id	21	15	6	12	3	0 (0 %)	0 (0 %)	6
	100 %	71.4 %	28.6 %	100 %	33.3 %	0 %	0 %	66.7 %
Tree-walk	21	19	2	12	7	0 (0 %)	2 (0 %)	0
	100 %	90.5 %	9.5 %	100 %	77.8 %	0 %	22.2 %	0 %
Kontekst	21	21	0	12	9	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	100 %	100 %	0 %	0 %	0 %
Robust Locs	21	21	0	12	9	0 (0 %)	0 (0 %)	0
	100 %	100 %	0 %	100 %	100 %	0 %	0 %	0 %
Res. XPaths	21	15	6	8	1	6 (48.5 %)	4 (100 %)	2
	100 %	71.4 %	28.6 %	66.7 %	11.1 %	66.7 %	44.4 %	22.2 %
TextFinder	21	21	0	12	3	6 (91.2 %)	0 (0 %)	0
	100 %	100 %	0 %	100 %	33.3 %	66.7 %	0 %	0 %

Tabel 5.6: Resultater for test af sletninger

Kontekst-metodens præstation var forventet, mens tree-walk godt kunne have risikeret at klare sig værre, men ankrenes start- og slutpositioner var i denne test tilstrækkeligt unikke til, at den korrekt udelod dem alle.

5.2.5 Konklusion

I det ovenstående er metoderne blevet afprøvet på de rene ændringstyper. For de fleste metoder var omformuleringstypen, den største udfordring. Kun Resilient XPaths distancerede sig fra de andre og genfandt alle ankre, mens de øvrige ikke kom over 50 % genfundne ankre. De fleste klarede de mindre ændringer godt. Robust Locations fik problemer, når start- og slutordet blev berørt, og TextFinder fik problemer, når størstedelen af ordene var udskiftede.

Rollerne var byttet om, da det gjaldt ankerflytninger. Her scorede Robust Locations takket været tree-walk-metoden 87,5 %, og TextFinder opnåede 100 % genfundne lokationer. Resilient XPaths fandt ingen af de otte nye lokationer. De lokationsbeskrivelser, metoden havde gemt, var møntet på det element, der oprindeligt indeholdt anker-teksten. Når elementet bliver liggende, men teksten flytter sig, er Resilient XPaths ude af stand til at relokalisere ankeret.

Ved både sletninger og paragrafflytninger klarede alle metoder sig fornemt. Paragrafflytningerne tillod Resilient XPaths at tage fat i anker-teksten, hvilket gav forbedringen i forhold til ankerflytning. Robust Locations og TextFinder nød også godt af den uændrede tekst mellem originalen og de senere sider, og fandt lokationen i alle siderne.

Sletningstesten viste, at alle metoderne er i stand til at "ikke-finde" en lokation korrekt. Men den viste også, at især Resilient XPaths har en risiko

for falske positiver.

Den samlede konklusion er ikke overraskende: Den udpræget strukturelle metode, Resilient XPaths, står stærkt overfor omformuleringer - indholds-ændringer - og mindre stærkt overfor flytninger - strukturelle ændringer. De indholdsbaserede metoder, kontekst og TextFinder, har omvendt deres styrke netop ved flytninger og er noget sårbare overfor omformuleringer. Tree-walk-metoden har karakter af at være både strukturelt og indholds-baseret. Den er mere alsidig, hvilket testresultaterne også antyder. Robust Locations har som samlet system haft det sværest med ændringer af omformuleringstypen, der voldte problemer for både kontekst- og tree-walk-metoderne.

I virkeligheden bliver tekst sjældent udsat for kun en enkelt type ændring. Alle typer kan forekomme på siden på en gang, og et anker kan risikere at blive både omformuleret og flyttet i samme redigering. Der forekommer heller ikke nødvendigvis lige mange af hver type ændring. Så hvor testene i dette afsnit påpeger metodernes styrker og svagheder, siger de ikke meget om metodernes generelle anvendelighed.

5.3 Wiki-redigeringer

Wiki-redigeringer[8] dækker over de ændringer, der naturligt er forekommet over en længere periode i en lang række wiki-sider. Ændringerne i disse sider er en kombination af alle de ovenstående ændringer i et virkelighedsnært indbyrdes forhold.

Det kan diskuteres, i hvor høj grad indholdet, strukturen og ændringerne i wiki-sider ligner andre websider som nyhedssider, blogs og personlige hjemmesider. På nyhedssider og blogs vil der ofte være flere sletninger - eller ret beset inter-dokument-flytninger, når et indlæg eller en nyhed ikke længere er aktuel og flyttes fra forsiden til et arkiv. Wikipedias XHTML-kode har også en bedre separering af struktur og layout end de fleste andre sider på nettet. Det betyder, at der vil være færre håndtag for Resilient XPaths at gribe fat i på stien fra rod til blad. Der vil formentlig også være flere rent tekstuelle ændringer i wiki-artikler end andre websider, da grundprincippet for wikisider er, at alle til enhver tid kan opdatere og tilrette dem. På øvrige sider med en eller få forfattere vil en publiceret tekst muligvis blive flyttet men sjældnere omskrevet.

Resultaterne er således ikke fuldstændigt repræsentative for internettets alsidighed, men de vil alligevel give et bedre indblik i metodernes virkelige, relative styrker, end en konstrueret test kunne. Adgangen til revisionshistorien betyder desuden, at der er langt større mængder test-data til rådighed,

[8]Wiki henviser her til den wiki, der findes på `http://www.wikipedia.org`

end det havde været muligt at indsamle på anden vis i løbet af den til rådighed værende tid.

Testdata

Jeg har udvalgt fem sider fra den engelske wiki. Siderne er alle udvalgt fra listen over "Featured content[9]". Udover at være på den liste lever de alle op til to særlige udvælgelseskriterier: De skal have eksisteret i mindst to år, og de skal have mindst 20 redigeringer per halvår. Disse kriterier skal sikre, at der kan findes flere versioner af siderne, og at der en vis forskel fra version til version.

Ankrene er valgt med en vis tilfældighed. Jeg har tilstræbt at vælge lokationer i alle typer indhold - menuer, overskrifter, citater, lister, tekst og figurer - med en overvægt at tekst-ankre. Tilfældigheden består i, hvilke bidder af hver type, der er udvalgt.

Lokationerne er udpeget i en tidlig version af siden og forsøges relokaliseret i et antal af de senere sider. Typisk er der valgt en version er per halve år fra den oprindelige side frem til den seneste indhentede redigering. Sidesættet, der bruges som udgangspunkt for Resilient XPaths, består af den oprindelige side samt to eller tre sider fra de efterfølgende måneder.

De forskellige versioner af siderne er hentet fra revisionshistorien for følgende artikler:

1. `http://en.wikipedia.org/wiki/Castle`
2. `http://en.wikipedia.org/wiki/Emu`
3. `http://en.wikipedia.org/wiki/Toraja`
4. `http://en.wikipedia.org/wiki/Unification_of_Germany`
5. `http://en.wikipedia.org/wiki/Law`

De datasæt, der er brugt til wiki-testene kan findes i test-suiten i bibliotekerne test/test9, -10, -11, -12 og -13. Her er det muligt at se, præcist hvilke versioner og ankre, der er brugt til testene.

Der er i hver testsæt hentet mellem fem og syv ændrede versioner af den originale side. Det er dog vigtigt at holde sig for øje, at der i de fleste tilfælde er cirka et halvt år mellem de brugte revisioner, hvilket vil sige mindst 20 redigeringer af artiklen. Disse kan være af større og mindre grad, og det er ikke alle ændringer, der påvirker ankrenes tekst eller position direkte.

[9] "Featured content" eller "fremhævet indhold" i den danske wiki er artikler, der udmærker sig ved deres kvalitet. Typisk vil de være relativt lange og have hyppige redigeringer.

Det kunne være interessant at tælle og klassificere de ændringer, ankrene har været udsat for, men denne opgave har været for omfattende at foretage på denne mængde data.

Resultater

Fem sider, tre til fem lokationer per side og mellem fem og syv ændrede versioner giver i alt 139 relokaliseringer. Resultaterne af disse ses i Tabel 5.7

Test	Forsøg	Succes	Fejl	√ udeladt	√ lok.	∼ lok. (kvl)	÷lok. (dst)	÷udeladt
Unikt id	139	11	128	0	9	2 (98 %)	4 (75.3 %)	124
	100 %	7.9 %	92.1 %	0 %	6.6 %	1.5 %	2.9 %	91.2 %
Tree-walk	139	67	72	3	49	15 (3.2 %)	0 (0 %)	72
	100 %	48.2 %	51.8 %	100 %	36 %	11 %	0 %	52.9 %
Kontekst	139	29	110	3	26	0 (0 %)	0 (0 %)	110
	100 %	20.9 %	79.1 %	100 %	19.1 %	0 %	0 %	80.9 %
Robust Locs	139	73	66	0	59	14 (2.5 %)	3 (100 %)	63
	100 %	52.5 %	47.5 %	0 %	43.4 %	10.3 %	2.2 %	46.3 %
Res. XPaths	139	37	102	3	1	33 (16.1 %)	89 (11.3 %)	13
	100 %	26.6 %	73.4 %	100 %	0.7 %	24.3 %	65.4 %	9.6 %
TextFinder	139	107	32	3	29	75 (85.2 %)	15 (85.9 %)	17
	100 %	77 %	23 %	100 %	21.3 %	55.1 %	11 %	12.5 %
RL + TF	139	116	23	0	63	53 (77.1 %)	8 (94.8 %)	15
	100 %	83.5 %	16.5 %	0 %	46.3 %	39 %	5.9 %	11 %

Tabel 5.7: Resultater for test af Wiki-redigeringer

Robust Locations klarer sig samlet hæderligt. Metoden har succes i lige godt 50 % af relokaliseringerne, og der er tilmed en stor del af disse, der er korrekte lokationer - 59 korrekte mod 14 delvist korrekte. Tilgengæld er kvaliteten af de delvist korrekte lokationer ikke ret god.

Den primære faktor til metodens succes er tree-walk-metoden, der alene kun klarer sig en smule dårligere end Robust Locations samlet. Unikt id-metodens primære bidrag er uheldigvis, at den forkert finder de tre lokationer, der burde være udeladt. De tre er egentlig en enkelt lokation, en fodnote, der er slettet i de tre nyeste versioner af den pågældende side. Fodnoterne i wiki-artiklerne får et id baseret på deres nummer i teksten, så når en fodnote slettes overtager den efterfølgende note id'et. Kontekst-metoden har succes i godt 20 % af tilfældene, men må have en del overlap med tree-walk-metoden, da det samlede resultat ellers ville have været højere.

Resilient XPaths klarer sig ikke ret godt i denne test. Den finder 25 % af lokationerne og får på grund af tre korrekt udeladte positioner sin samlede succesrate op på 26,6 %. Typetestene viste, at Resilient XPaths kan have problemer med anker-flytninger, hvilket kan forklare en del af de manglende

fund. En anden forklaring kan være, at sidesættet, som XPath-udtrykket er baseret på, ikke har været tilstrækkeligt varieret til at udtrykke de senere ændringer. En sidste forklaring kan være, at wikisiderne ikke tilbyder nok håndtag i elementerne i form af attributter. Mange steder vil kun elementets position og `count()`-funktionen kunne bruges til at sige noget udover typen på et element.

Testens "vinder" er TextFinder, der har en succesrate på hele 77 %. Hovedparten er kun delvist korrekte lokationer, men kvaliteten af disse er relativt høj. At det ikke bliver til helt korrekte lokationer, skyldes i mange tilfælde udeladelsen af "ubetydelige" ord - hvis disse ligger i starten eller slutningen af ankerteksten, vil de ikke være inkluderet i den fundne løsning. Når TextFinder har succes, betyder det, at ved hovedparten af ændringerne er en stor del af ankerteksten bevaret. Enten fordi ankeret blot er flyttet, eller fordi omformuleringerne af teksten har været små.

Ser man på metodernes forkert fundne lokationer holdt op imod de forkert udeladte lokationer, har Robust Locations det pæneste forhold, såfremt man accepterer, at en udeladelse er bedre end en forkert lokalisering. Resilient XPaths har et meget dårligt forhold, men til gengæld ligger de forkerte lokationer relativt tæt på den korrekte lokation, hvilket bestemt er en formildende omstændighed. TextFinder har cirka lige mange af hver type fejl, men skyder til gengæld langt ved siden af, når der findes en forkert lokation.

Under kodenavnet RL+TF gemmer der sig resultaterne fra en testkørsel, hvor TextFinder var medtaget som submetode i Robust Locations. Resultatet forbedres noget, men der forekommer også en del overlap mellem metodernes succeser. Resilient XPaths rangeringsmekanisme er for simpel til, at den meningsfuldt kan medtage som submetode, da den giver alle ikke-null lokationer en rangering på 100.

5.4 Analyse

Testene har vist at metodernes styrker afhænger af, hvad deres lokationsbeskrivelse består af. Strukturelt baserede metoder står stærkt overfor rent tesktuelle ændringer, mens de indholdsbaserede metoder står stærkt overfor strukturelle ændringer.

Blandt de indholdsbaserede løsninger kan der også spores en forskel. TextFinders lokationsbeskrivelse er baseret på ankerteksten, og Robust Locations' kontekst-metode baserer sin lokationsbeskrivelse på både ankertekst og kontekst. Sidstnævntes resultater er alt eller intet: Enten findes en lokation korrekt eller også udelades den. Dette står i stor kontrast til førstnævnte,

der i næsten alle test har flere delvist korrekte lokationer end helt korrekte lokationer. De lidt løsere kriterier anvendt af TextFinder betyder dog også, at den finder mange flere lokationer i alt end kontekst-metoden.

Testene har også vist, at i levende dokumenter med hyppige redigeringer af vilkårlig type[10], er Resilient XPaths en mindre god løsning end Robust Locations og TextFinder.

Det betyder ikke, at Resilient XPaths er ubrugelig. Det betyder, at i sin nuværende form er den ikke særlig anvendelig til annotering af tekst. Den har sin styrke i at udpege regioner af websider, for eksempel en menu eller et banner, som i vid udstrækning ikke flytter rundt, eller som i hvert fald bibeholder en mængde karakteristika, der kan identificere elementet unikt. Metoden har ingen mulighed for at udpege tekstbidder på mindre end element-niveau, og når en test i et vist omfang beder om netop det, kommer metoden naturligvis til kort.

Man kan dog sagtens forestille sig, at Resilient XPaths i en udvidet udgave med mulighed for at pege ind i medie-elementer ville kunne klare sig langt bedre. Den lave afstand, de forkerte lokationer har til de rigtige, tyder på, at problemet primært ligger i de sidste par skridt mod lokationen. Med en reparations-funktion a la tree-walk-metoden kunne resultatet let have set bedre ud. En oplagt mulighed ville være XPointer-frameworket således, at metoden stadig er baseret på W3Cs standarder.

Robust Locations' succesrate på godt 50 % er også i underkanten af, hvad et annotationssystem bør levere for at tilfredsstille sine brugere. Metoderne i Robust Locations er dog også kun beskrevet som eksempler, der kan indgå i en samlet relokaliseringsstrategi (se afsnit 2.5.1). Det er oplagt at udvide Robust Locations med flere LocSpecMethods, for eksempel TextFinder. Inkluderes denne med de tre øvrige i Robust Locations fås da også betydeligt bedre resultater.

Robust Locations med TextFinder står stærkere end Resilient XPaths, når det gælder lokalisering af og i tekst, mens Resilient XPaths styrke ligger i lokalisering af hele regioner eller segmenter af websider. Hvilken metode, der skal benyttes, afhænger altså af anvendelsen. For begge metoder gælder det dog, at der stadig er plads til forbedringer og udvidelser.

5.5 Opsummering

I dette kapitel har jeg beskrevet de test og test-data, jeg har benyttet i evalueringen af LocSpec-metoderne. Jeg har testet metoderne på fire relativt

[10]Dog fortsat med anmærkningerne om wiki-sidernes generalitet fra indledningen af afsnittet i tankerne.

små testsæt, der hver indeholdt sin type af de ændringer Brush *et al.* har identificeret (se afsnit 2.1.3). Jeg har også testet metoderne på et større testsæt bestående af revisioner af fem forskellige wikisider.

Jeg har videre fremstillet resultaterne og givet en umiddelbar vurdering af dem. Slutteligt har jeg på baggrund af de samlede resultater analyseret metodernes præstation og sammenlignet dem.

Testene har vist, at metoderne har forskellige kvaliteter. Resilient XPaths er bedst egnet til lokalisering af elementer, der kun i begrænset omfang flytter sig på siden, eller som bibeholder en mængde karakteristika, der kan identificere elementet unikt. Robust Locations er, især med tilføjelsen af TextFinder som submetode, velegnet til relokalisering af tekststykker.

6

Konklusion

Formålet med dette speciale har været at efterprøve eksisterende metoders kvaliteter inden for relokalisering af dynamisk indhold i websider samt at bidrage til fremstillingen af en forbedret metode.

De eksisterende metoder, jeg har undersøgt, er Robust Locations fremstillet af Phelps og Wilensky i 2000 og Resilient XPaths fremstillet af Paz og Díaz i 2010. Den teoretiske afdækning og arbejdet med implementeringen af metoderne afslørede mulige problemer for begge. Resilient XPaths så ud til at være meget afhængig af et repræsentativt sidesæt. Robust Locations så qua sine submetoders indbyrdes komplementering ud til at stå bedre overfor alsidige ændringer. En svaghed ved systemet var, at de to bærende submetoder begge var afhængige af det første og sidste ord i ankerteksten.

Metodernes reelle kvaliteter er blevet afprøvet i test. De praktiske test underbyggede de iagttagelser, jeg gjorde i det teoretiske arbejde. De illustrerede Resilient XPaths svaghed ved ankre positioneret inde i tekstelementer. Analysen af resultaterne indikerer, at Resilient XPaths formentlig ville klare sig bedre i scenarier, hvor lokaliseringen gjaldt større segmenter af siderne.

Robust Locations håndterede dynamikken i testene markant bedre end Resilient XPaths, men den led stadig under de svagheder, det teoretiske arbejde havde afsløret. Resultaterne tydede ikke på, at metoden ville virke tilfredsstillende i et brugsscenarie.

På baggrund af de svagheder, som teori, implementation og test afslørede og med inspiration fra Brush *et al.*'s iagttagelser i "Robust annotation positioning in digital documents" har jeg selv udviklet en metode: TextFinder. Den baserer lokalisereringen på nøgleord i selve ankerteksten uden at være afgørende afhængig af et enkelt ord. Metoden kan både benyttes alene og som et element i Phelps og Wilenskys overordnede strategi. TextFinder klarede wiki-testen bedre end de øvrige metoder, men det absolut bedste resultat fremkom, da den blev brugt som en integreret del af Robust Locations.

Frameworket, jeg har udviklet til testene, letter udviklingen af nye metoder. Et grundprincip i frameworket er muligheden for at bruge flere metoder samtidig til relokalisering. Dette princip stammer fra Phelps og Wilenskys arbejde, og det synes centralt i indsatsen for at opnå de bedst mulige resultater.

I arbejdet med dette speciale har jeg identificeret en række punkter, hvor der med fordel kan foretages ændringer. Frameworkets behandling af metodernes resultater kan forbedres, og flere af de individuelle metoder rummer fortsat potentiale for udvidelse og udvikling. Disse potentialer vil jeg sammenfatte i næste kapitel.

7

Fremtidigt arbejde

Gennem arbejdet med specialet er det blevet klart, at der er mange muligheder for forbedringer på området relokalisering i dynamisk indhold. Mulighederne går i forskellige retninger: Forbedringer af de enkelte metoder, samspillet mellem metoderne og understøttelsen af forskellige medietyper. Jeg har også i nogen grad savnet kilder, der analyserede behovet for annotationer og hvilke typer dynamik, der er hyppigst på World Wide Web.

I dette kapitel vil jeg kort afdække de væsentligste tanker, jeg har gjort mig inden for disse retninger.

Metoderne

Det blev åbenlyst i testene, at hvis Resilient XPath skal kunne bruges til annotationer af tekst, har den brug for et redskab til at lokalisere længere ned end på elementniveau. Et oplagt valg for tekstmediet ville være xpointer()-scheme til XPointer frameworket, der er en standard baseret på XPaths. Det kunne være interessant og gavnligt at undersøge, hvordan induktion og simuleret annealing kunne håndtere xpointere.

Tree-walk-metoden har som nævnt i afsnit 4.1.2 den risiko, at den finder det resultat, der i DOM-træet ligger tættest på det oprindelige ankers position, også selvom dette ikke giver den bedste rangering. En forbedring kunne være at finde flere løsninger og returnere den bedste. Det ville selvfølgelig skulle opvejes mod en øget kørselstid.

Tree-walk- og kontekst-metoderne led begge under at være meget afhængige af ordet eller ordene lige omkring ankerets begyndelse og slutning. Det sikrer mere præcise match, når metoderne har succes - men det betyder også, at selv små ændringer i disse kritiske regioner kan føre til en forkert udeladelse. Begge metoder kunne derfor med fordel adoptere TextFinder-metodens brug af et sæt af nøgleord i uspecificeret rækkefølge.

TextFinder havde mange delvist korrekte lokationer i testene, hvilket i høj grad skyldes udeladelsen af ubetydelige ord under konkstruktionen af den fundne lokation. Denne opførsel kan forbedres, hvis metoden efter at have afgjort hvilket "vindue", der er bedst, også leder efter og inkluderer de ubetydelige ord, der findes i nærheden af vinduet.

Approksimeret tekst-søgning dækker over forskellige, lignende teknikker, hvor der benyttes en serie mutationer af søgestrengen (her ankerteksten eller nøgleord) til at lede i dokumentet. Man kan både forestille sig denne teknik anvendt som en selvstændig metode og som en forbedring af de beskrevne

metoder - for eksempel tree-walk, kontekst og TextFinder. Idéen til brug af approksimeret tekst-søgning stammer fra Phelps og Wilensky [12, s. 111].

Frameworket

For at frameworket i dets nuværende konstruktion kan fungere optimalt, forudsættes en grundig balancering af metodernes interne rangeringer af deres resultater. Jeg har forsøgt at opnå en sådan balancering, men den kunne formentlig fortsat forbedres.

En anden og mere holdbar tilgang til problemet kunne være at se bort fra eller reducere betydningen af rangeringerne, da en perfekt balancering for alle tilfælde nok ikke kan opnås. Når der er tre eller flere returnerede lokationer at vælge i mellem, får AnchorManageren mulighed for at sammenligne på andre parametre end metodernes egne rangeringer. Er der overlap? Ligger to lokationer tæt på hinanden og et tredje langt væk? Jeg forestiller mig en udvidelse af frameworket med et analysemodul, der har til opgave at konstruere en endelig løsning baseret på de individuelle metoders svar.

Frameworket er som beskrevet i kapitel 3 forberedt på håndtering af CompLocSpecs - altså lokationsspecifikationer for den webside, der indeholder lokationen. En implementering af denne mulighed ville åbne muligheden for at opnå robusthed mod inter-dokument-ændringer. Det vil sige at genfinde et anker selv efter en flytning til en anden adresse.

Medier

I specialet har jeg alene fokuseret på ankre i tekstindhold, bortset fra en enkelt Wiki-test, der indeholdt et anker bestående af et billede. Det kunne imidlertid være nærliggende videre at kigge på metoder til at specificere ankre placeret i video, billeder eller lyd, selvom annotering af disse medier ikke på samme måde er udbredt som annotering af tekst. Reel specificering og brug af sådanne ankre forudsætter dog i vid ustrækning adgang til medieafspillerne, som der er begrænset adgang til direkte fra webbrowseren. Derfor er udfordringen for andre medietyper noget større end for tekst.

Analyser

Unsworth beskrev annotering som en af de videnskabelig primitiver [13]. Marshall fremhævede betydningen af annotationer for hypermediesystemer, og hun identificerede nogle af de akser, annotationer bevæger sig langs - for eksempel offentlig-privat og formel-uformel [9]. Men præcis hvor ligger behovet for annotationer i dag, på World Wide Web: Er det som videnskabeligt

værktøj eller som et socialt kommunikationsmiddel? Er der behov for at annotere forskningsartikler? Fakta-sider som wikipedia? Eller tilmed meget dynamiske sider som Facebook? En analyse af disse behov kan hjælpe til med at identificere, hvor og hvordan lokationsbeskrivelserne skal forbedres fremover.

Brush opdelte ændringer i indholdet på websider i kategorierne omformuleringer, ankerflytninger, paragrafflytninger og sletninger [4]. Jeg brugte disse som udgangspunkt for testene og benyttede wikipedia-redigeringer som repræsentant for ændringer i websider, men jeg manglede en undersøgelse af frekvensen af sådanne ændringer på tværs af forskellige, virkelige websider. En sådan undersøgelse ville kunne hjælpe prioriteringen mellem de forskellige metoder. Flest omformuleringer kunne pege på en opprioritering af de strukturelle metoder, mens en overvægt af flytninger ville indikere en højere vægtning af de tekstuelle metoder.

Opsummering

I dette kapitel har jeg sammenfattet de mest interessante idéer og tanker jeg har fået til videre arbejde indenfor området. Idéerne omfatter både konkrete forbedringer af de eksisterende metoder, en forbedret udnyttelse af flere metoder og en udvidelse af frameworket til også at håndtere inter-dokument-ændringer. Jeg har også efterspurgt analyser af behovet for annotationer samt undersøgelser over frekvensen af de forskellige typer webdynamik.

Der er således fortsat en masse spændende udfordringer af både praktisk og akademisk art på området for automatisk lokalisering af dynamisk indhold på World Wide Web.

A

Locations.xml

Dette er et eksempel på en locations.xml-fil, som de bruges i datasættene i test-suiten.

I eksemplet herunder er to lokationer specificeret for hver af to inkarnationer af URL'en "http://da.wikipedia.org/wiki/DAIMI". Attributten "file" på inkarnationerne angiver, hvilken fil websiden er gemt i, mens "xptr" som nævnt giver den præcise lokation i filen. "Date" angiver tidspunktet for offentliggørelsen af den pågældende inkarnation, men denne værdi bliver dog ikke anvendt i testene.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <locations>
3    <location id="1" url="http://da.wikipedia.org/wiki/DAIMI">
4      <incarnation date="2009-06-20" file="2009-06-20.html" xptr=
           'xpointer(string-range(/html[1]/body[1]/div[3]/div[4]/p
           [2], "", 17, 31))'>
5      </incarnation>
6      <incarnation date="2009-06-21" file="2009-06-21.html" xptr=
           'xpointer(string-range(/html[1]/body[1]/div[3]/div[4]/p
           [2], "", 17, 31))'>
7      </incarnation>
8    </location>
9    <location id="2" url="">
10     <incarnation date="2009-06-20" file="2009-06-20.html" xptr=
           'xpointer(string-range(/html[1]/body[1]/div[3]/div[4]/p
           [4], "", 1, 33))'>
11     </incarnation>
12     <incarnation date="2009-06-21" file="2009-06-21.html" xptr=
           'xpointer(string-range(/html[1]/body[1]/div[3]/div[4]/p
           [4], "", 1, 33))'>
13     </incarnation>
14   </location>
15 </locations>
```

Oversigt A.1: locations.xml

Konstruktionen er identisk for "originale" og "senere" lokationer.

B

Test-suite

Udover vejledningen her, findes der yderligere instruktioner i README-filen i testsuiten.

XULRunner og XPCShell

For at afvikle test-scriptet, start_tests.js, forudsættes en installation af XULRunner med XPCShell. XULRunner kan downloades fra:

```
http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases/
```

Under antagelse af, at XULRunner er installeret i `/usr/lib/xul/`, kan scriptet startes via XPCShell således:

```
/usr/lib/xul/bin/run-mozilla.sh /usr/lib/xul/bin/xpcshell -w
start_tests.js
```

For at benytte DynamicAnchor-frameworket skal XPT-filerne først placeres i XULRunner-installationen:

```
./xpt/*
```
kopieres til:
```
/usr/lib/xul/bin/components/
```

Testene er afviklet med xulrunner-devel-1.9.2.13 på en Ubuntu distribution af Linux.

Afvikling af en test

For at afvikle en test, skal scriptet startes med et argument, der angiver test-konfigurationen. Der findes test-konfigurationer for testene i specialet i bibliotekerne i testuisten der hedder "<type>tests"

En test kan afvikles således:

```
/usr/lib/xul/bin/run-mozilla.sh /usr/lib/xul/bin/xpcshell -w
start_tests.js t=wikitests/wiki-pwj
```

Litteratur

[1] E. Adar, J. Teevan, S.T. Dumais, and J.L. Elsas. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 282–291. ACM, 2009. 6

[2] N.O. Bouvin. Augmenting the Web through Open Hypermedia. 2000. 3

[3] SP Brooks and BJT Morgan. Optimization using simulated annealing. *The Statistician*, 44(2):241–257, 1995. 25, 29, 61, 63

[4] AJ Brush, D. Bargeron, A. Gupta, and J.J. Cadiz. Robust annotation positioning in digital documents. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 292. ACM, 2001. 7, 31, 63, 66, 67, 87

[5] K. Grønbæk and R.H. Trigg. Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects. In *Proceedings of the the seventh ACM conference on Hypertext*, pages 149–160. ACM, 1996. 10, 32

[6] F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994. 9, 10, 32

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *The Statistician*, 44(2):241–257, 1995. 25, 29

[8] W. Koehler. Web page change and persistence—A four-year longitudinal study. *Journal of the American Society for Information Science and Technology*, 53(2):162–171, 2002. 6

[9] C.C. Marshall. Toward an ecology of hypertext annotation. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems: links, objects, time and space—structure in hypermedia systems*, pages 40–49. ACM, 1998. 5, 31, 86

[10] N. McFarlane. *Rapid application development with mozilla*. Prentice Hall PTR, 2004. 34

[11] I. Paz and O. Díaz. Providing resilient XPaths for external adaptation engines. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, pages 67–76. ACM, 2010. 23, 26, 27, 28, 32, 56, 60, 61, 62

[12] T.A. Phelps and R. Wilensky. Robust intra-document locations. *Computer Networks*, 33(1-6):105–118, 2000. 17, 18, 22, 32, 51, 86

[13] J. Unsworth. Scholarly Primitives: what methods do humanities researchers have in common, and how might our tools reflect this. In *Humanities Computing: formal methods, experimental practice symposium, King's College, London*, pages 5–00, 2000. Tilgængelig 20/09/2010 på `http://www3.isrl.illinois.edu/~unsworth/Kings.5-00/primitives.html`. 5, 31, 86