Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Interaktiv visualisering av IP-nätverk

av

Steve Eriksson

LIU-IDA...

2010-xx-xx

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Interaktiv visualisering av IP-nätverk

av

Steve Eriksson

LIU-IDA/…

2010-xx-xx

Handledare: Dennis Sehalic
Examinator: Juha Takkinen

Sammanfattning

Telenors svenska nätverksövervakning har utvecklat ett system för att automatiskt generera nätverkskartor i formatet SVG. De har ställt frågan om det går att göra dessa interaktiva och koppla ihop dem med befintliga verktygsprogram.

Denna studie visar tekniker som kan användas för att utveckla ett system som gör SVG-baserade nätverkskartor interaktiva i en webbläsare.

Ett system baserat på öppen mjukvara och öppna standarder utvecklas för att visa hur teknikerna kan användas i praktiken. Systemets arkitektur beskrivs i tre systemvyer. Nätverkskartorna berikas med bindningar mellan händelser i webbläsaren och JavaScript-funktioner genom att transformera dem med XSLT. Användargränssnittet utgörs av SVG-objekt och JavaScript varifrån det går att asynkront anropa program på en webbserver.

Studien avslutas med att utvärdera systemet och ge förslag på hur det kan förbättras.

Innehåll

Figurer

Listningar

Kapitel 1

Inledning

1.1 Bakgrund

Telenor [1] i Sverige är en av landets största leverantörer av kommunikation via mobil- och fast telefoni och är med köpet av Bredbandsbolaget [2], Sveriges näst största bredbandsleverantör.

Övervakning och drift av Telenors IP- och mobiltelefoninät i Sverige görs av Telenor NOC[1] som är belägen i Karlskoga.

Övervakning av nätverken innebär att operatörerna på NOC tar emot larm från utrustning i dessa. Larmen kommer från ett flertal olika leverantörers övervakningssystem. Operatörerna tar även emot felanmälningar via telefon, e-post och ärendehanteringssystem.

Drift av nätverken innebär att operatörerna reagerar på ovan nämnda larm och felanmälningar och ansvarar för att defekter åtgärdas antingen av operatören själv eller av en lämplig tekniker.

De senaste åren har det på Telenor utvecklats flertal verktygsprogram för att underlätta operatörernas dagliga arbete. Exempel på dessa är program som rapporterar hur många kunder som är inkopplade mot en viss nätverksnod, hur många noder det finns i en viss stad och hur många av dem som är kontaktbara. De flesta av verktygsprogrammen är UNIX-baserade program skrivna i programmeringsspråken Python, Perl och Bash och anropas via ett kommandoskal i UNIX.

1.2 Existerande system

Visualisering av IP-nätverk

Telenor NOC har tillgång till nätverkskartor som visar viktiga delar av IP-nätverket. Denna dokumentation utgörs av dokument i olika filformat men gemensamt för dessa är att de ej är dynamiska eller interaktiva; det saknas koppling mellan element i dokumenten och motsvarande nod i IP-nätverket.

[1] Network Operations Center

Verktygsprogram

Nätverkstekniker på Telenor har utvecklat ett flertal verktygsprogram. Dessa program är skapta att köras i ett kommandoskal. Varje program har egna förutsättningar och begränsningar vilket betyder att användaren måste känna till vilka parametrar dessa använder och vilka krav som måste uppfyllas innan exekvering.

1.3 Problem

Telenors IP-nätverk (fortsättningsvis kallat nätverket) är omfattande och innehåller många förbindelser mellan noderna i det. Varje nod i nätverket innehåller åtminstone ett tiotal förbindelser, redundanta och icke redundanta, till andra noder. Det är därför svårt att bilda en mental modell över nätverket. När ett larm rörande en bruten förbindelse eller att kontakten förlorats med en nod i nätverket inkommer till Telenor NOC är det ofta svårt att förstå hur problemet påverkar nätverket. Ett larm är vanligtvis stämplat med en identifierare för den aktuella noden och om det rör en förbindelse även ett indexnummer för den port förbindelsen är kopplad till.

En vanlig uppgift på Telenor NOC är att uppskatta omfattningen av en driftstörning som uppstår då noder i nätverket går ner. Driftstörningar kan uppstå via oförutsedda händelser som till exempel strömavbrott eller planerade arbeten, exempelvis uppgradering av en nätverknods operativsystem. Det finns på Telenor NOC verktygsprogram som listar antalet anslutna kunder till en nod men den listan innehåller ej information om förbindelser till andra noder i nätverket. Konfigurationen för varje port i aktuell nod måste undersökas för att avgöra vilka andra noder den är ansluten till.

1.4 Motivering

En grafisk representation av nätverket ger operatörerna på Telenor NOC en övergripande bild över de komplexa kopplingar som existerar mellan noder i nätverket och de slipper skapa en mental modell över dessa.

Telenor NOC har börjat undersöka vilka möjligheter det finns att visualisera delar av nätverket i ett SVG[2]-dokument och göra det interaktivt. Man vill också göra det möjligt för en användare att anropa externa program från dokumentet.

Dokumentformatet SVG har valts av flera anledningar:

- SVG är ett XML[3]-baserat dokumentformat och en fil i detta format är tillgänglig i klartext. Dokumentet kan därför, till skillnad från binära format, enkelt analyseras av både människor och program. Data i dokumenten är tillgängliga för framtida användare och utvecklare.
- SVG är ett dokumentformat för webben. SVG-dokument är enkla att infoga i webbsidor och kan även användas självständigt i en webbläsare. Ett system utvecklat för att exekveras i en webbläsare kräver ingen installation eller administration på en användares dator vilket minskar arbetet med

[2]Scalable Vector Graphics
[3]Extensible Markup Language

att underhålla det. Det finns även möjligheter att använda tillgängliga webbtjänster som till exempel Google Maps[4] för att öka funktionaliteten i systemet.

- SVG-specifikationen innehåller bindningar till JavaScript och dokumenten kan därför göras interaktiva på samma sätt som webbsidor i HTML-format. Ett SVG-dokument kan användas både som en nätverkskarta och som användargränssnitt i en webbaserad applikation.

1.5 Syfte

Syftet med detta arbete är att utveckla ett prototypsystem som gör nätverkskartor i formatet SVG interaktiva. Det ska vara möjligt att via nätverkskartorna anropa befintliga verktygsprogram.

1.6 Avgränsningar

Beställaren har beslutat att använda visualiseringsmjukvaran GraphViz för att generera nätverkskartor i formatet SVG. Systemet behöver ej innehålla funktionalitet för att skapa konfigurationsfiler till GraphViz baserat på befintlig topologi av nätverket. Systemet behöver ej generera dessa nätverkskartor.

Den del av systemet som exekveras på klientsidan ska vara väl fungerande i webläsaren Firefox. Stöd för fler webläsare är ej nödvändigt.

Arbetets huvudsakliga syfte är att undersöka hur element i ett SVG-dokument kan kopplas till ett verktygsprogram. Det är därför ej nödvändigt att göra en komplett matchning mellan alla befintliga verktygsprogram och vilka typer av system i nätverket dessa har stöd för.

Frågor rörande säkerhet och tillgänglighet behöver ej besvaras.

1.7 Metod

För att strukturera arbetet har jag valt att använda George Pólyas problemlösningsmetod [3]. Pólyas metod består av fyra steg för att strukturera arbetet med att lösa problem. Under varje punkt följer en beskrivning på hur jag applicerar metoden i mitt arbete.

1.7.1 Förstå problemet

För att förstå problemet som beställaren vill lösa ska jag genomföra informella, ostrukturerade intervjuer med min handledare på Telenor. Efter varje intervju ska de problem som identifierats analyseras och frågor som dyker upp i denna ska antecknas och användas i en uppföljande intervju.

När inga frågor och oklarheter återstår ska jag i tätt samarbete med min handledare ta fram en kravspecifikation. Kravspecifikationen kommer sedan att användas för att avgränsa arbetet och göra de möjligt för beställaren och mig att avgöra om arbetets mål har uppfyllts.

[4] En webbaserad karttjänst.

Nyckelord i kravspecifikationen kommer att användas som stöd när jag tar fram en litteraturbas. Mitt mål med litteraturstudien är att skapa en förståelse för området och ge en uppfattning om vilka tekniker som kan användas för att lösa de problem som identifieras genom de ovan nämnda intervjuerna.

1.7.2 Skapa en plan

Baserat på den teori jag tillskansar mig genom litteraturstudien ska jag ta fram olika lösningsalternativ till de problem som jag identifierat. De lösningsalternativ jag anser uppfylla beställarens krav bäst och som är enklast att implementera kommer att väljas.

En lista kommer att upprättas av mig som innehåller de valda lösningarna. Listans ordning ska baseras på hur viktigt varje problem är att lösa. Varje punkt i listan ska sedan brytas ned till mindre delproblem.

1.7.3 Utföra planen

Jag ska bearbeta listan med lösningar sekventiellt och implementera dessa i ett prototypsystem.

Programutvecklingen kommer inte följa en formell metod utan vara av en explorativ art. Utvecklingen av systemet ska ske i små iterationer där fler och fler funktioner läggs till och testas kontinuerligt.

1.7.4 Utvärdera resultatet

När jag anser att systemet uppfyller beställarens krav ska det testas enligt kravspecifikationen.

1.8 Rapportens disposition

- Det andra kapitlet innehåller en kravspecifikation som tagits fram av författaren och uppdragsgivaren.
- I rapportens tredje kapitel ges en teoretiskt grund för de begrepp och tekniker som används i arbetet.
- Kapitel fyra redovisar de olika designval som jag identifierade baserat på kravspecifikationen och den teori som tillskansats under litteraturstudien.
- I det femte kapitlet beskrivs implementationen av ett prototypsystem. En systemöversikt ges först genom tre systemvyer följt av en djupare genomgång av systemets viktigaste komponenter.
- Arbetets resultat visas i kapitel sex. I kapitlet utvärderas systemet mot kravspecifikationen och det avslutas med en sammanfattande diskussion.
- I kapitel sju som avslutar denna rapport för jag en diskussion om arbetet och ger förslag på framtida förbättringar av systemet.

Kapitel 2

Krav

De krav som listas i detta kapitel framkom under arbetets tidiga fas. När arbetet var definierat av uppdragsgivaren fanns några grundläggande krav, nämligen att systemet skulle använda SVG-dokument för den grafiska representationen, att dessa skulle visas i webbläsaren Firefox och att dokumentet skulle göras interaktivt genom JavaScript. Efter att jag genomfört en preliminär analys diskuterade jag med uppdragsgivaren vilka tekniker som var godtagbara att använda i systemet. När de grundläggande kraven ovan var bestämda tog slutligen jag och uppdragsgivaren fram de funktionella krav som skulle ställas på systemet.

Jag valde att dela upp de olika kraven i två kategorier så att jag kunde fokusera på de viktigaste kraven först. Detta gjorde på grund av att implementationsfasen av systemet skedde under en väldigt begränsad tidsperiod och eftersom jag ej genomfört ett liknande arbete tidigare var det svårt att uppskatta den faktiska tidsåtgången för att lösa varje problem. De viktigaste kraven klassificerades som obligatoriska och skulle uppfyllas för att arbetet ska anses vara helt slutfört. De mindre kritiska kraven klassificerades som frivilliga och skulle uppfyllas i mån av tid.

Varje krav har ett ID-nummer och de är ej sorterade i prioritetsordning.

2.1 Obligatoriska krav

K1 Mjukvarupaketet GraphViz ska användas för att generera SVG-dokument.

K2 Den grafiska representationen ska vara i formatet SVG.

K3 Applikationer på serversidan ska vara av typen CGI-skript skrivna i programmeringsspråket Perl.

K4 Applikationer på klientsidan ska vara skrivna i programmeringsspråket JavaScript.

K5 Webservern som används i systemet ska vara LightTPD.

K6 Systemet ska stödja webläsaren Firefox version 3.5 eller senare.

K7 Ett befintligt verktygsprogram ska kunna anropas via användarinteraktion med SVG-dokument i webläsaren.

K8 Anrop enligt krav K7 ska ske asynkront.

K9 Resultatet av körningen av verktygsprogrammet ska visas i den webläsare där anropet initierades.

K10 Interaktion med SVG-dokument på klientsidan ska ej påverka originaldokumentet på serversidan.

K11 Alla komponenter i systemet ska använda öppen mjukvara.

2.2 Frivilliga krav

F1 När en användare högklickar på ett element i SVG-dokumentet ska en lista med tillgängliga funktioner visas.

F2 Listan i krav F1 ska genereras baserat på elementets identifierare.

F3 Systemet ska innehålla funktionalitet för att automatiskt anropa ett verktygsprogram baserat på en timer.

2.3 Testning av krav

Uppfyllande av krav K1, K2, K3, K4, K5 och K11 testas genom att undersöka de ingående komponenterna.

K6 Utför de tillgängliga funktionerna i klientsidans gränssnitt i Firefox version 3.5 och version 3.6. Granska om SVG-dokumentet renderas korrekt i webläsaren.

K7 Utför funktion på klientsidan och notera resultatet. Utför verktygsprogrammet manuellt och jämför resultaten. Detta förutsätter att verktygsprogrammet är deterministiskt.

K8 Undersök programmets källkod på klientsidan.

K9 Utför funktion i webläsaren och notera om resultat visas efter körning.

K10 Utför de tillgängliga funktionerna på klientsidan. Kontrollera att originaldokumentet på servern är oförändrat.

F1 Högerklicka på element i dokumentet och notera vilka funktioner som listas. Jämför denna lista med de funktioner som gjorts tillgängliga för elementet i källkoden på klientsidan.

F2 Undersök källkod på klientsidan för att se om elementets identifierare används för att generera listan med tillgängliga funktioner.

F3 Starta timer och kontrollera att verktygsprogram anropas genom att skapa spårutskrifter på serversidan.

2.4 Sammanfattning

Framställandet av kravspecifikationen var en mycket viktig del av arbetet. Genom att ha regelbundna diskussioner med min handledare under arbetets tidiga faser kunde jag snabbt förkasta vissa idéer och fokusera mer på andra. Arbetet med kravspecifikationen skedde parallellt med problemanalysen och litteraturstudien och fungerade som ett hjälpmedel för att avgränsa och precisera vad som skulle genomföras under arbetet med systemet. Jag valde att dela upp kravspecifikationen i två delar där den första berör obligatoriska krav som ska uppfyllas och den andra berör frivilliga krav som ska uppfyllas i mån av tid.

I nästa kapitel ges en kort orientering av de olika tekniker jag valde att använda, baserat på kravspecifikationen och litteraturstudien.

Kapitel 3

Bakgrund

Detta kapitel ska ge läsaren en teoretisk grund för de viktigaste begreppen och teknikerna som används i rapporten.

Kapitlet inleds med att förklara vad som menas nätverksvisualisering följt av hur jag valt att tolka begreppet öppen mjukvara som används genomgående i arbetet. Därefter beskrivs de tekniker som används i arbetet för att representera ett nätverk grafiskt följt av en förklaring av AJAX som används för att göra denna representation interaktiv och dynamisk. Slutligen ges en överblick av de tekniker som används på serversidan för att leverera data till och mottaga anrop från klienten.

3.1 Nätverksvisualisering

Ett icke trivialt IP-nätverk består av många noder som är förbundna med varandra genom olika typer av transmissionsmedier. Man kan föreställa sig nätverket som ett moln av noder och förbindelser. Nätverksvisualisering handlar om att kika in i molnet och snabbt få en övergripande bild över de komplexa relationer som finns i det. Grafer är ett kraftfullt sätt att representera ett IP-nätverk. I grafen representeras nätverksutrustningen som noder och förbindelserna mellan dem som bågar.

3.2 Öppen mjukvara

Termen öppen mjukvara kan tolkas på olika sätt men har ofta innebörden att mjukvarans källkod ska finnas tillgänglig. När termen öppen mjukvara används i denna rapport avses den definition organisationen Open Source Initiative skapat [4]. Open Source Initiative menar att källkoden ska vara tillgänglig utan kostnad. Det ska också vara möjligt att göra förändringar i koden och distribuera dessa.

3.3 SVG

Scalable Vector Graphics [5] är ett språk som beskriver tvådimensionell vektorbaserad grafik i formatet XML. SVG-bilder kan göras interaktiva och dynamiska

genom att använda ECMAScript och en anpassad version av DOM level 2 kallad SVG-DOM.

3.4 GraphViz

Graph Visualization Software [6] är ett grafvisualiseringssystem utvecklat av AT&T Research. GraphViz innehåller verktyg och bibliotek för att generera grafer och presentera dem grafiskt i en mängd olika filformat. GraphViz använder språket DOT [7] för att beskriva en graf i en textbaserad konfigurationsfil.

3.5 AJAX

AJAX är en förkortning skapad av Jesse James Garret som står för "Asynchronous JavaScript And XML" och är en samling tekniker för att skapa snabba och dynamiska webbsidor. Enligt Garret [8] består teknikerna av:

- Presentation baserad på standarderna XHTML och CSS.
- Dynamisk visning och interaktion genom Document Object Model.
- Utbyte och manipulation av data genom XML och XSLT.
- Asynkron hämtning av data genom XMLHttpRequest.
- JavaScript för att binda ihop teknikerna.

3.5.1 XHTML

XHTML [9] är en familj av XML-baserade dokumenttyper som är en delmängd och utökning av HTML version 4. Detta medför att ett XHTML-dokument kan analyseras, läsas, valideras och editeras med vanliga XML-verktyg.

3.5.2 CSS

Cascading Style Sheets [10] är en teknik som används för att styra utseendet på ett dokument. Genom att utnyttja CSS kan ett dokuments utseende och innehåll separeras.

CSS används ofta tillsammans med HTML-dokument men kan användas i XML-tillämpningar som exempelvis SVG och XHTML.

3.5.3 DOM

Document Object Model [11] är ett API som möjliggör dynamisk interaktion med ett dokument. Dokumentet representeras som ett träd där varje nod i trädet är ett element i DOM.

DOM-gränssnittet tillåter att ett program dynamiskt ändrar innehåll, struktur och utseendet på ett dokument.

3.5.4 XML

Extensible Markup Language [12] är ett standardiserat textformat som beskriver innehållet i ett dataobjekt, företrädelsevis ett dokument.

XML är baserat på det äldre märkspråket SGML och påminner därför ytligt om HTML.

Ett XML-dokument som uppfyller kraven på att alla taggar ska vara avslutade och enbart innehålla tillåtna tecken kallas välformulerat. Anges en DTD[1] i dokumentets huvud kan dess innehåll valideras. En DTD innehåller definitioner för hur dokumentet ska struktureras med en lista av godkända element och attribut. Ett validerat XML-dokument måste också vara välformulerat.

3.5.5 XSLT

Extensible Stylesheet Language Transformations [13] är ett XML-baserat språk som används för att transformera XML-dokument. Det resulterande dokumentet kan skilja sig markant från det ursprungliga då element i det kan filtreras och ordnas om. Godtyckliga XML-strukturer kan också läggas till det.

3.5.6 XMLHttpRequest

XMLHttpRequest [14] är ett JavaScript-objekt ursprungligen skapat av Microsoft som tillåter hämtning av data givet en URL[2]. Trots namnet stödjer objektet [15] dataöverföringar i flera olika format, bland annat text. Objektet stödjer även överföringar via andra protokoll än http som till exempel ftp.

De populäraste webbläsarna har en implementation av detta objekt och W3C[3] har även standardiserat det.

Genom att utnyttja XMLHttpRequest-objektet kan asynkrona funktionsanrop göras till en server och resultatet visas i dokumentet i webbläsaren utan att hela dokumentet behöver laddas om.

3.5.7 JavaScript

JavaScript [16] är en dialekt av det standardiserade programmeringsspråket ECMAScript och utvecklades ursprungligen av företaget Netscape. Den vanligaste miljön för JavaScript är webbläsaren som exponerar ett dokuments DOM-gränssnitt för JavaScript-programmeraren.

JavaScript stödjer programmering i flera olika paradigmer så som objektorienterad, procedurell, och funktionell programmering [17].

3.6 LightTPD

LightTPD [18] är en webbserver som finns tillgänglig som öppen mjukvara. Den fokuserar på hög prestanda och låg minnesanvändning.

[1] Document Type Definition
[2] Unified Resource Locator
[3] World Wide Web Consortium

3.7 CGI

Common Gateway Interface [19] är en defacto-standard som specificerar hur en webbservers funktionalitet kan utökas genom att låta ett externt program generera det dokument som webbservern ska leverera till en klient.

Webbservern avgör vilket program som ska utföras baserat på en URL. och information rörande förfrågan överförs till det externa programmet via miljövariabler [20].

3.8 Sammanfattning

Detta kapitel har gett en orientering i vad nätverksvisualisering innebär och vilka olika tekniker som använts för att implementera systemet.

Med de ovan beskrivna teknikerna som utgångspunkt redovisar jag i nästa kapitel vilka designval som identifierades under arbetet.

Kapitel 4

Analys

Jag har valt att bryta ned problemet med att göra en interaktiv och grafisk representation av ett IP-nätverk i en webbläsare i fyra mindre delproblem. De två första delproblemen berör klientdelen i systemet och de två sista berör även serverdelen:

- Det första delproblemet är att göra ett SVG-dokument interaktivt. Det ska programmatiskt gå att förändra innehållet i nätverkskartan och visa information i denna baserat på en användares handlingar. För detta krävs bindningar mellan element i dokumentet och JavaScript-funktioner.
- Det andra delproblemet som måste lösas är hur användarinitierade händelser i dokumentet såsom musklick ska fångas upp och behandlas av klienten.
- Det tredje delproblemet berör kommunikation mellan klient och server. Klienten ska anropa servern när en användare valt att utföra en funktion på ett av elementen i nätverkskartan.
- Det fjärde och sista delproblemet är hur anropen från klienten till servern ska behandlas.

De fyra delproblemen och de alternativa lösningar jag funnit till dessa behandlas nedan.

4.1 Bindning av JavaScript-funktioner

För att göra nätverkskartan i ett SVG-dokument interaktiv, måste det skapas bindningar till funktioner som utförs baserat på vilka händelser användare utlöst i webbläsaren.

Elementen i ett SVG-dokument kan kopplas till händelser i en webbläsare genom att göra tillägg i deras attributlistor. Attributen för dessa händelser kan referera till JavaScript-funktioner. En händelse binds till en funktion via ett element [21].

Om JavaScript-funktionerna inte ska definieras i SVG-dokumentet måste det finnas referenser i form av script-element [22] till JavaScript-filer innehållande dessa definitioner.

Jag har identifierat fyra olika metoder för att skapa bindningar mellan händelser och JavaScript-funktioner.

4.1.1 Traversering av SVG-DOM i webbläsare

När ett SVG-dokument har analyserats av en webbläsare kan tillägg göras till dokumentets element genom att använda JavaScript och DOM-gränssnittet [23]. På detta sätt kan bindningar av händelser ske på klientsidan.

SVG-dokumentet i fråga måste innehålla en referens till en fil som innehåller den JavaScript-funktion som utför tilläggen. En sådan referens skapas genom att lägga till ett script-element i dokumentet.

4.1.2 Traversering av SVG-dokument på servern

Ett SVG-dokument kan analyseras och tillägg göras i dess element på serversidan genom att använda ett bibliotek som implementerar DOM. CPAN[1] innehåller flera Perlbibliotek som tillhandahåller denna funktionalitet.

När bindningarna genomförts går det ej att ändra dem utan direkt manipulation av dokumentet.

4.1.3 Transformation av SVG-dokument genom XSLT

XML-dokument kan transformeras med XSLT. Genom att använda en XSL-transformerare kan ett SVG-dokument analyseras och ändras baserat på regler som definierats i en XSL-formatmall. Det är möjligt att lägga till, ta bort och ändra attribut i ett dokument. Förutom att berika dokumentet med bindningar till JavaScript-funktioner kan referenser till JavaScript-filer och CSS-filer läggas till.

XSLT stöds av de populäraste webbläsarna: Internet Explorer [24], Mozilla Firefox [25], Opera [26] och de som är baserade på WebKit [27]. Transformation kan således ske på klientsidan och kräver att en referens till XSL-formatmallen läggs till i dokumentet.

Fristående XSL-transformerare kan exekveras på serversidan. Källdokumentet och XSL-formatmallen kan anges som parametrar till XSL-transformeraren och en referens behöver inte läggas till i dokumentet.

Två populära XSL-transformerare som är tillgängliga som öppen mjukvara är Xalan och libxslt.

Apache Xalan

Apache projektets XSL-transformerare Xalan [28] finns tillgänglig som Javaprogram. Xalan kan anropas från ett Perlprogram.

XML::LibXSLT

LibXSLT [29] är ett Perlbibliotek som omsluter GNOME-projektets XSL-transformerare libxslt [30]. LibXSLT går att använda i ett Perlprogram genom att inkludera det i källkoden.

[1] Comprehensive Perl Archive Network. En stor samling mjukvara och dokumentation för Perl. Se http://www.cpan.org/

4.1.4 Skapa bindning till funktioner i GraphViz konfigurationsfil

GraphViz konfigurationsfil för en graf styr hur ett SVG-dokument ska utformas. Det går att ange att givna element ska innehålla länkattribut.

Bindningar kan skapas genom att låta länkattributen referera till JavaScript-funktioner. Detta innebär dock en begränsning då endast en händelse som avfyras när användaren väljer att exekvera en länk, kan bindas till JavaScript-funktioner.

4.2 Hantering av användarinitierade händelser

En användare interagerar med nätverkskartan i SVG-dokumentet genom att klicka på noder och bågar i det. Ett musklick på ett element i dokumentet är en händelse som ska anropa en JavaScript-funktion.

Standarden för SVG-DOM [31] definierar en mängd händelser som kan bindas till anrop av JavaScript-funktioner. Varje händelse som är definierad i elementens attributlistor kan bindas till en JavaScript-funktion.

För att en mängd olika funktioner ska kunna bindas till en given händelse i ett element kan en expedieringsfunktion behandla funktionsanropet och utföra ett godtyckligt antal funktioner i sekvens. Expedieringsfunktionerna bildar en abstraktion mellan händelser i SVG-dokumentet och de funktioner som ska utföras när dessa utlöses. Expedieringsfunktionerna kan senare utökas med fler funktioner utan att förändra programkoden i den.

Ett alternativ till att använda expedieringsfunktioner är att definiera en funktion för varje enskilt element och händelse i SVG-dokumentet. En nackdel med denna lösning är att när kopplingen mellan en given händelse och en funktion skapas så är det svårt att senare göra utökningar av funktionen. Utökningar av funktionen kräver att programkoden i den ändras.

4.3 Anrop från klient till server

När en användare klickat på en förbindelse eller nätverkselement i nätverkskartan ska en given åtgärd utföras. Detta kan till exempel vara att visa minnesanvändningen i en router. Begäran att visa denna information ska skickas till servern.

Enligt kravspecifikationen ska begäran att utföra ett program på servern ske asynkront så att klienten inte är låst under tiden som servern utför programmet. Asynkrona anrop kan göras genom att utnyttja JavaScript-objektet XMLHttpRequest i webbläsaren Firefox. W3C:s nuvarande specifikation [14] av XMLHttpRequest har en begränsning i att anrop enbart får ske till samma värdnamn som angetts vid begäran av det aktuella dokumentet. W3C har producerat ett arbetsutkast [32] där denna begränsning tagits bort. Firefox har sedan version 3.5 implementerat ett XMLHttpRequest-objekt som inte har denna begränsning [33].

Ett alternativ till XMLHttpRequest är att skapa ett nytt script-element i dokumentet där sökvägen till skriptet pekar på ett CGI-skript [34]. Detta CGI-skript returnerar ett JavaScript-program som kan innehålla funktioner och data som vid analys lägger till nya element och textnoder i dokumentet. Denna

lösning begränsas inte av att kommunikationen måste ske mot samma server som levererat det aktuella dokumentet.

4.4 Behandling av anrop från klient

När en begäran att utföra ett program eller visa information rörande en nätverksnod eller förbindelse mottagits av servern ska denna behandla anropet och leverera resultatet av begäran till klienten. Två generella lösningar för hur detta ska ske har identifierats.

4.4.1 Monolitiskt CGI-skript

Ett CGI-skript ansvarar för att behandla klientens anrop. Skriptet innehåller en expedieringsfunktion som baserat på parametrar i anropet utför önskat program.

Tillägg av nya program som systemet ska stödja kräver att ändringar sker i de JavaScript-program som utförs på klientsidan och i expedieringsfunktionen i CGI-skriptet.

4.4.2 CGI-skript som agerar skalprogram

För varje program i systemet som klienten kan anropa finns ett CGI-skript som agerar skalprogram. Det behandlar klientens anrop och utför önskat program.

Utökning av tillgängliga program i systemet kräver ändringar i det JavaScript-program som utförs på klientsidan och att ett nytt CGI-skript skapas för att hantera anropet från klienten. Befintliga CGI-skript på serversidan behöver ej ändras.

4.5 Sammanfattning

I detta kapitel har jag visat hur jag valt att dela upp problemet att skapa en interaktiv visualisering av ett IP-nätverk i fyra delproblem. De två första delproblemen berör, på klientdelen av systemet, kopplingar mellan användarinitierade händelser i ett SVG-dokument och JavaScript-funktioner och hur händelserna ska hanteras. De två övriga delproblemen berör även serverdelen av systemet, nämligen hur anrop från en klient till servern ska ske och hur anropen ska behandlas av servern.

Nästa kapitel visar implementationen av ett system för nätverksvisualisering baserat på analysen jag genomförde i detta kapitel.

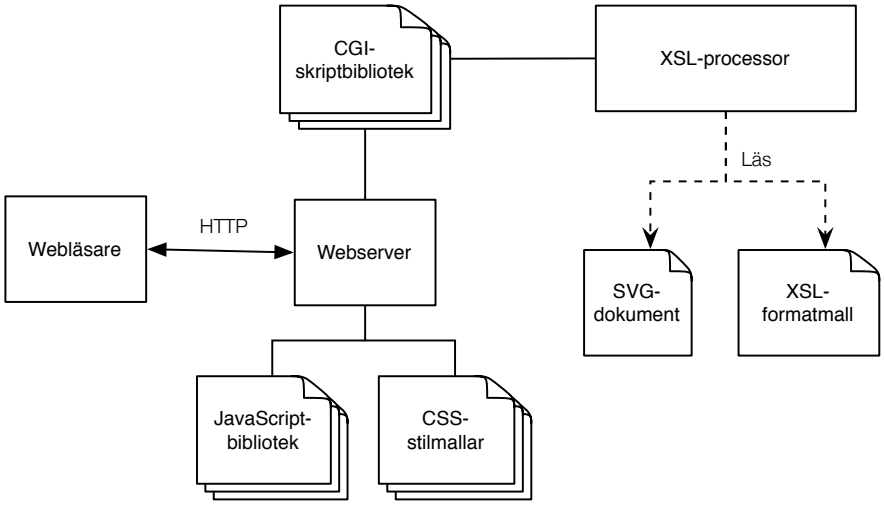
Kapitel 5

Implementation

I detta kapitel visas den systemarkitektur jag skapat baserat på kravspecifikationen och de designval som jag identifierade i min analys i föregående kapitel. Arkitekturen beskrivs med hjälp av tre systemvyer: exekveringsvy, implementationsvy och driftsättningsvy. De viktigaste komponenterna i systemet beskrivs sedan djupare.

5.1 Exekveringsvy

Exekveringsvyn i figur 5.1 visar systemets övergripande arkitektur och relationen mellan dess komponenter under exekvering. Komponenterna kommer att beskrivas i detalj nedan.



Figur 5.1: Exekveringsvy

Webbläsare

Systemet stödjer endast webbläsaren Firefox version 3.5 eller senare. Inga insticksprogram eller andra modifikationer krävs för att systemet ska fungera som avsett.

Webbserver

Systemet använder en standardinstallation av webbservern LightTPD.

JavaScript-bibliotek

Ett bibliotek med JavaScript-filer innehåller all funktionalitet som systemet kräver på klientsidan. Det SVG-dokument som genererats av webservern innehåller referenser till nämnda JavaScript-filer som hämtas automatiskt av webläsaren när dokumentet analyseras.

CSS-stilmallar

Det genererade SVG-dokumentets utseende styrs till viss grad av stilmallar.

CGI-skriptbibliotek

Generering av berikade SVG-dokument och kopplingar till externa program sker via anrop till CGI-skript programmerade i Perl.

Varje externt program hanteras av ett specifikt CGI-skript.

XSL-processor

Ett CGI-skript berikar ett SVG-dokument vid förfrågan. Detta innebär att det ursprungliga dokumentet lämnas oberört och kan användas i andra tillämpningar.

Systemet använder XSL-processorn libxslt genom biblioteket XML::LibXSLT.

XSL-formatmall

XSL-processorn använder sig av denna formatmall för att transformera det ursprungliga SVG-dokumentet och skapa bindningar mellan händelser och JavaScript-funktioner

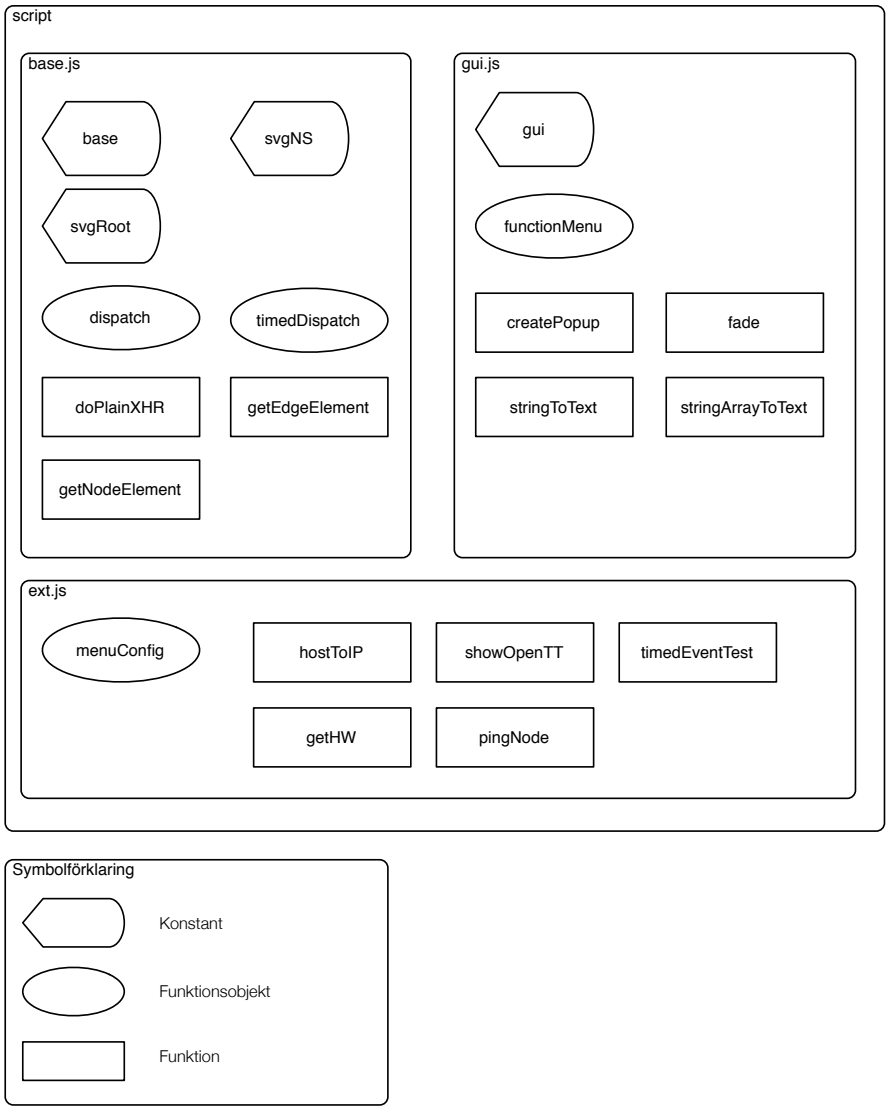
Formatmallen innehåller även regler för att skapa referenser till JavaScript-filer och CSS-stilmallar.

SVG-dokument

Nätverkskartor i SVG-format skapas av ett externt program med hjälp av GraphViz.

5.2 Implementationsvy över JavaScript-bibliotek

JavaScript-biblioteket består av filer som hanterar systemets logik på klientsidan. Filerna visas i figur 5.2 på nästa sida och kommer att beskrivas i detalj nedan.



Figur 5.2: Implementationsvy över JavaScript-bibliotek

5.2.1 Base.js

Filen *base.js* innehåller de funktioner som är bundna till händelser i det berikade SVG-dokumentet och hjälpfunktioner som abstraherar delar av systemets underliggande implementation. Både *gui.js* och *ext.js* kräver att denna fil är exekverad innan de kan analyseras av webläsaren.

- *Base* är ett namnrymdsobjekt som används för att skydda konstanter och funktioner från namnkrockar då variabler är globala i JavaScript.
- Konstanten *svgRoot* håller en referens till SVG-dokumentets rot i DOM-trädet.
- *SvgNS* innehåller XML-namnrymden för SVG och underlättar vid namnrymdsspecifika funktionsanrop i DOM.
- Funktionsobjektet *dispatch* används för att hantera de funktionsanrop som bundits till händelser i SVG-dokumentet och tillåter att flera funktioner utförs givet en specifik händelse.
- Funktionsobjektet *timedDispatch* innehåller metoder för att lägga till och ta bort funktioner som är schemalagda att utföras i ett givet intervall.
- Funktionen *doPlainXHR* underlättar skapandet av asynkrona funktionsanrop till webservern.
- *GetEdgeElement* är en av flera funktioner som abstraherar den underliggande representationen i systemet. Givet ett händelseobjekt kan denna funktion ta fram det grafiska element som representerar en båge i grafen.
- *GetNodeElement* har samma funktionalitet som *getEdgeElement* förutom att det element som representerar en nod i grafen returneras.

5.2.2 Gui.js

GUI står för graphic user interface vilket betyder grafiskt användargränssnitt. Denna JavaScript-fil innehåller funktioner kopplade till det grafiska gränssnittet.

- *Gui* är ett namnrymdsobjekt med samma funktion som *base* i *base.js*.
- Funktionsobjektet *functionMenu* innehåller metoder för att lägga till verktygsprogram i en kontrollmeny som visas i SVG-dokumentet.
- Funktionen *createPopup* skapar en dialogruta i SVG-dokumentet.
- *Fade* tonar ut eller in ett element i SVG-dokumentet och används av *createPopup* och *functionMenu*.
- Funktionen *stringToText* skapar en grupp innehållande ett textelement och kan användas för att lägga till text i en dialogruta i SVG-dokumentet.
- *StringArrayToText* har samma funktionalitet som *stringToText* förutom att den kan skapa flera textelement givet ett objekt med strängar.

5.2.3 Ext.js

Denna fil innehåller exempel på hur systemet kan programmeras med hjälp av de ovan nämnda filerna. Utökande av ny funktionalitet till systemet bör göras i denna fil.

- *MenuConfig* är ett objekt innehållande en konfiguration som används vid skapandet av ett funktionsobjekt av typen *functionMenu*.
- Funktionen *hostToIP* anropar ett CGI-skript på webservern som returnerar ip-adressen som är bunden till givet ett värdnamn.
- Funktionen *showOpenTT* demonstrerar hur ett verktygsprogram kan anropas utan hjälp av *doPlainXHR* och visa resultatet som ett HTML-dokument i ett nytt webläsarfönster.
- *TimedEventTest* är ett exempel på hur *timedDispatch* kan utnyttjas för att en eller flera funktioner ska utföras regelbundet och automatiskt.
- Funktionen *getHW* anropar ett CGI-skript som returnerar information rörande ett nätverkselements hårdvarubestyckning.
- *PingNode* sänder en förfrågan till ett CGI-skript att kontrollera om ett nätverkselement är åtkomlig via nätverket webservern är kopplad till.

5.3 Implementationsvy över CGI-skriptbibliotek

De komponenter som ingår i CGI-skriptbiblioteket och dess beroenden visas i figur 5.3 på följande sida och beskrivs nedan. Samtliga CGI-skript är utvecklade i programmeringsspråket Perl.

5.3.1 GetHWInfo.pl

CGI-skriptet *getHWInfo.pl* hämtar information rörande ett nätverkselements hårdvarubestyckning givet ett värdnamn.

5.3.2 ShowOpenTT.pl

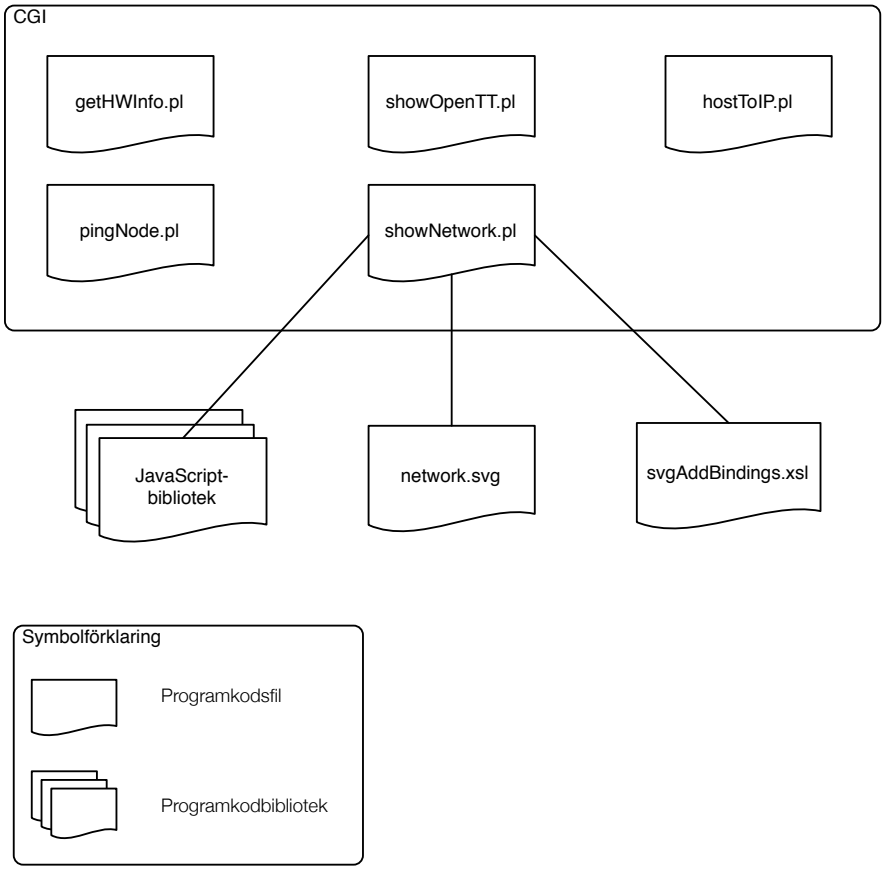
Programmet *showOpenTT.pl* skapar ett HTML-dokument och returnerar det till klienten.

5.3.3 HostToIP.pl

Programmet *hostToIP.pl* returnerar en IP-adress tillhörande ett givet värdnamn.

5.3.4 PingNode.pl

CGI-skriptet *pingNode.pl* kontrollerar om ett nätverkselement kan nås via det nätverk webservern är ansluten till.



Figur 5.3: Implementationsvy över CGI-skriptbibliotek

5.3.5 ShowNetwork.pl

CGI-skriptet *showNetwork.pl* utgör den viktigaste delen av systemet då det berikar ett SVG-dokument med bindningar och referenser.

Ett nytt dokument skapas baserat på ett SVG-dokument som transformeras med hjälp av XSLT och returneras till webservern som därefter skickar det till klienten.

XSL-transformationen kräver att ett externt Perlbibliotek används. Biblioteket heter XML::LibXSLT och kan installeras genom CPAN.

5.3.6 Network.svg

Dokumentet *network.svg* är ett exempel på en nätverkskarta i SVG-format. Det är skapat av ett externt program med hjälp av GraphViz.

5.3.7 SvgAddBindings.xsl

XSL-formatmallen innehåller regler för hur exempelfilen *network.svg* ska transformeras så att det berikas med bindningar och referenser.

5.4 Driftsättningsvy

Driftsättningsvyn i figur 5.4 på nästa sida visar de hårdvarukomponenter som ingår i systemet och var mjukvarukomponenterna är utplacerade.

5.4.1 Klient

Klienten består av en dator med ett operativsystem som är kompatibelt med webbläsaren Firefox 3.5 eller senare.

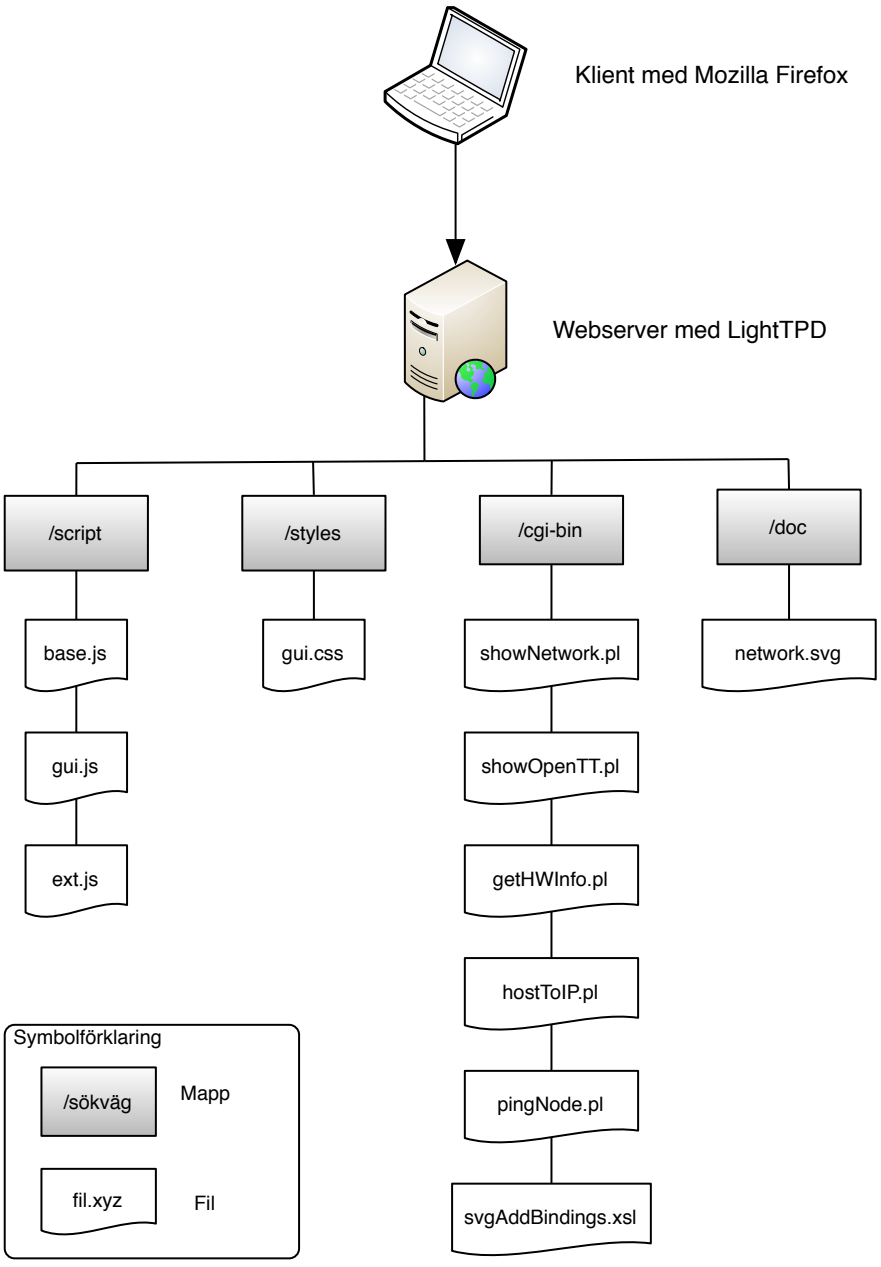
5.4.2 Server

Servern består av en dator med ett UNIX-baserat operativsystem med webservern LightTDP installerad. I webserverns rotkatalog för webdokument finns det fyra mappar.

- */script* innehåller systemets JavaScript-filer.
- */styles* innehåller CSS-stilmallar som styr dokumentens utseende.
- */cgi-bin* innehåller alla CGI-skript som klienten kan anropa.
- */doc* innehåller tillgängliga nätverkskartor i form av SVG-dokument.

5.5 Generering av SVG-dokument

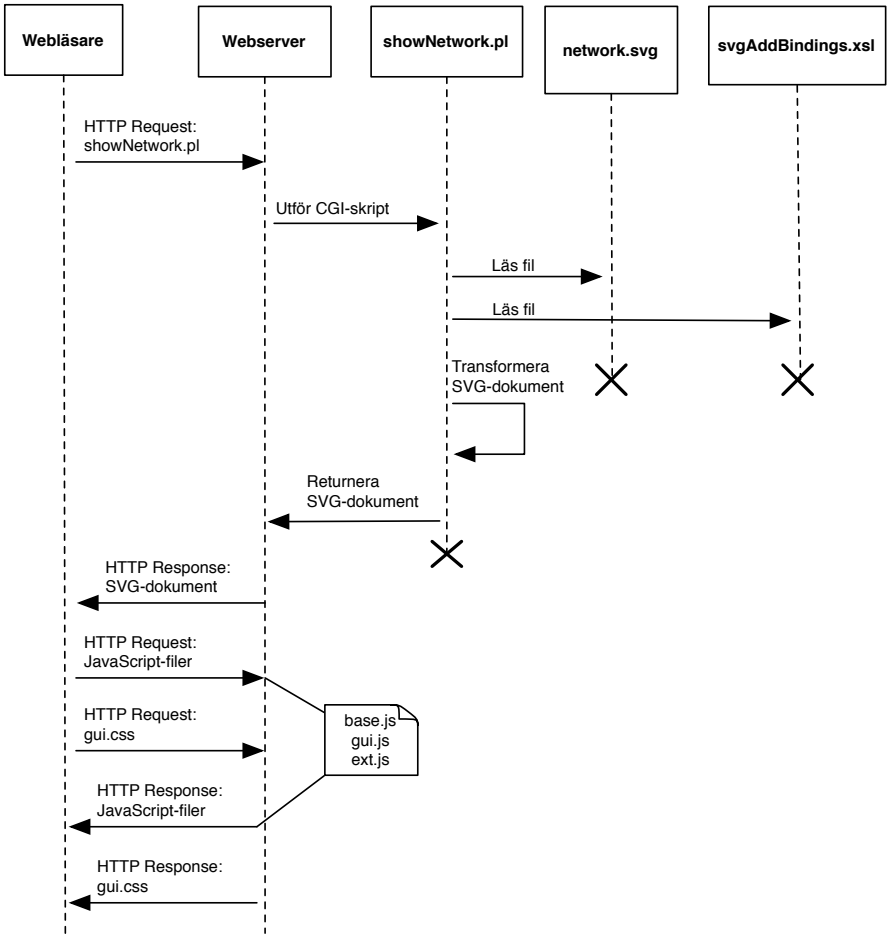
Klienten kan göra en förfrågan till servern att leverera ett interaktivt SVG-dokument. Förfrågan görs genom att ange en URL på formen `http://värdnamn.domän/cgi-bin/showNetwork.pl?network=namn.svg`. Sekvensdiagrammet i figur 5.5 på sidan 24 visar hur i detalj hur genereringen går till.



Figur 5.4: Driftsättningsvy

5.5.1 Förfrågan från klient

Generering av SVG-dokument sker genom att anropa ett CGI-skript på servern. Dokumentet returneras till klienten som begär att få de filer som är refererade i det.



Figur 5.5: Sekvensdiagram över begäran av SVG-dokument

5.5.2 Behandlande av förfrågan

CGI-skriptet *showNetwork.pl* tar emot klientens förfrågan om att visa en nätverkskarta och genererar ett nytt, berikat SVG-dokument. Den del i CGI-skriptet som utför transformationen och returnerar det resulterande dokumentet visas i listning 5.1 på nästa sida.

Listning 5.1: XSL-transformation

```
my $parser = XML::LibXML->new();
my $xslt   = XML::LibXSLT->new();

my $source     = $parser->parse_file( $svg_file );
my $style_doc = $parser->parse_file( $xsl_file );

my $stylesheet = $xslt->parse_stylesheet( $style_doc );

my $result = $stylesheet->transform( $source );

print $cgi->header( "image/svg+xml" );
print $stylesheet->output_string( $result );
```

5.6 XSL transformation

CGI-skriptet showNetwork.pl tar emot en förfrågan om att berika ett SVG-dokument.

Formatmallsfilen *svgAddBindings.xsl* innehåller regler för hur dokumentet ska transformeras. De viktigaste transformationerna är kopiering av element, referenser till JavaScript-filer och bindningar mellan händelser och funktioner.

5.6.1 Identitetstransformation

För att det nya berikade dokumentet ska innehålla all information från ursprungsdokumentet måste alla element och dess attribut kopieras. Detta görs genom att applicera en så kallad identitetstransformation som visas i listning 5.2.

Listning 5.2: Identitetstransformation

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

5.6.2 Tillägg av referenser till JavaScript-filer

Roten i dokumentet är ett SVG-element. En bindning till en *onload*-händelse görs i elementets attributlista. Detta möjliggör anrop till en JavaScript-funktion när dokumentet laddats färdigt. Befintliga attribut kopieras. Referenser till JavaScript-filer görs genom att skapa script-element innehållande sökvägar till dessa. Notera att olika namnrymder måste användas då olika typer av XML-element används i formatmallen.

Listning 5.3 på nästa sida visar hur referenser läggs till.

Listning 5.3: Skapa referenser genom XSLT

```
<xsl:template match="/svg:svg">
  <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:attribute name="onload">
        svg_onload(evt)
    </xsl:attribute>
    <svg:script type="text/ecmascript" xlink:href="../
      script/base.js" />
    <svg:script type="text/ecmascript" xlink:href="../
      script/gui.js" />
    <svg:script type="text/ecmascript" xlink:href="../
      script/ext.js" />
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>
```

5.6.3 Bindning av JavaScript-funktioner

Det ursprungliga SVG-dokumentet innehåller två klasser av element som är intressanta för användaren att interagera med. Dessa är klassen *nod* som representerar en nod i nätverket som till exempel en IP-router och klassen *edge* som representerar en förbindelse mellan två noder i nätverket. Bindningar ska skapas för dessa två klasser. De händelser som jag valt att skapa bindningar till är *onclick*, *onmouseover* och *onmouseout* vilka är en delmängd av de befintliga mus-händelserna i DOM level 2. Listning 5.4 på följande sida visar hur bindningen genomförs.

5.7 Asynkrona anrop via XMLHttpRequest

Klientens anrop till servern att utföra program ska enligt kravspecifikationen göras asynkront. Detta innebär att klientens gränssnitt som utgörs av SVG-dokumentet ej låses medan den väntar på svar från servern.

För att underlätta asynkrona anrop har en hjälpfunktion skapats kallad *doPlainXHR*. Funktionen använder parametrar som anger på vilket sätt anropet ska ske (Get eller Post), vilken URL den ska utföra anropet mot och en funktion som ska utföras när servern skickar ett svar. *doPlainXHR* returnerar serverns svar som ren text men objektet XMLHttpRequest kan även returnera svaret som XML.

Ett XMLHttpRequest-objekt kan endast hantera ett anrop i taget. Ett svar från ett anrop måste ges innan objektet kan utföra nästa anrop. Eftersom *doPlainXHR* i listning 5.5 på sidan 28 skapar ett nytt XMLHttpRequest-objekt vid varje anrop döljs denna begränsning för användaren.

Listning 5.4: Bindning av JavaScript-funktioner

```
<xsl:template match="svg:g">
  <xsl:copy>
    <xsl:if test="@class='node'">
      <xsl:attribute name="onclick">
          node_onclick(evt)
      </xsl:attribute>
      <xsl:attribute name="onmouseover">
          node_onmouseover(evt)
      </xsl:attribute>
      <xsl:attribute name="onmouseout">
          node_onmouseout(evt)
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@class='edge'">
      <xsl:attribute name="onclick">
          edge_onclick(evt)
      </xsl:attribute>
      <xsl:attribute name="onmouseover">
          edge_onmouseover(evt)
      </xsl:attribute>
      <xsl:attribute name="onmouseout">
          edge_onmouseout(evt)
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>
```

Listning 5.5: XMLHttpRequest

```
base.doPlainXHR = function( method, url, fun ) {
    var xhr = new XMLHttpRequest();
    xhr.open( method, url );
    xhr.onreadystatechange = function() {

        // Request is done
        if (xhr.readyState == 4) {

            // Check status code and handle errors
            if ( xhr.status == 200 ) {    // it went well
                fun( xhr.responseText );
            } else if ( xhr.status == 404 ) {
                base.showNotFoundError( url );
            } else if ( xhr.status == 500 ) {
                base.showInternalError( url );
            } else {
                base.showUnknownError( xhr.responseText );
            }

        }
    };
    xhr.send( null );
}
```

5.8 Expedieringsobjekt

Systemet har stöd för att utföra flera funktioner givet en händelse i SVG-dokumentet. Detta är implementerat genom expedieringsobjekt som innehåller en datastruktur med nyckel/värde-par för ett godtyckligt antal funktioner. När JavaScript-filen *base.js* laddas av webläsaren skapas ett expedieringsobjekt för varje definierad händelse i SVG-dokumentet.

5.8.1 Funktion för att skapa expedieringsobjekt

Funktionen *makeDispatch* i listning 5.6 på följande sida skapar ett nytt expedieringsobjekt och returnerar ett gränssnitt till dess metoder. Metoderna gör det möjligt att lägga till, ta bort och lista funktioner i objektet. Metoden *handleEvent* tar ett händelseobjekt som parameter och utför alla funktioner som för tillfället finns i expedieringsobjektets datastruktur.

5.8.2 Tillägg av funktioner till expedieringsobjekten

Kodexemplet i listning 5.7 på nästa sida visar hur en funktion kan läggas till ett expedieringsobjekt för händelsen *onclick*. När händelsen avfyras kommer ett asynkront anrop att genomföras till ett CGI-skript med namnet *ajaxTest.pl*. Resultatet av körningen av skriptet kommer att visas i en dialogruta i webläsaren.

Listning 5.6: Skapa expedieringsobjekt

```
base.makeDispatch = function() {

    var funMap = {}; // functions

    // Return the interface to the dispatch object
   return {
       addFunction: function( name, fun ) {
           funMap[name] = fun;
       },

       removeFunction: function( name ) {
           delete funMap[name];
       },

       listFunctions: function() {
           var names = [];
           for (name in funMap) {
               // hasOwnProperty is used so that things
                 from the prototype chain
               // isn't dragged up here
               if (funMap.hasOwnProperty( name )) {
                   names.push( name );
               }
           }
           return names;
       },

        handleEvent: function( evt ) {
            for (name in funMap) {
                if (funMap.hasOwnProperty( name )) {
                    funMap[name]( evt );
                }
            }
        }

   }; // end return
}
```

Listning 5.7: Tillägg av funktioner i expedieringsobjekt

```
base.edgeClickDispatch.addFunction( 'ajax', function( evt )
{
    base.makePlainXMLRequest( 'GET', '/cgi-bin/ajaxTest.pl?
      key=edge_ajax',
    function( response )
    {
      alert( response )
    } );
} );
```

5.9 Funktionsmeny

När en användare klickar på en nod i SVG-dokumentet visas en meny som listar de funktioner som finns tillgängliga. När dessa funktioner utförs kan de använda den aktuella noden som parameter.

Funktionsmenyn är kopplad till ett funktionsobjekt som skapas när dokumentet laddats klart av webläsaren. Listning 5.8 visar hur objektet kan skapas.

Listning 5.8: Skapa funktionsmeny

```
base.svgLoadDispatch.addFunction( 'create_menu',
                                  function() {
                                      gui.functionMenu.
                                        create( menuConfig
                                        ) } );
```

5.9.1 Konfiguration av funktionsmeny

När funktionsmenyn skapas kan ett konfigurationsobjekt anges som parameter. Detta objekt innehåller information om vilka dimensioner menyn ska ha, hur den ska se ut och vilka funktioner den ska innehålla. Funktionerna anges som ett nyckel/värde-par där värdet är en referens till en namngiven eller anonym funktion. Om värden saknas i konfigurationsobjektet eller om objektet saknas helt så skapas funktionsmenyn med förvalda värden för dessa. Listning 5.9 visar ett exempel på ett konfigurationsobjekt.

Listning 5.9: Konfigurationsobjekt för meny

```
var menuConfig = {
  x: base.viewBoxCenter - 400, y: 0,
  width: 800, height: 200,
  rx: 5, ry: 5,
  functions: { 'Get IP adress': hostToIP,
    'Show open trouble tickets': showOpenTT,
    'Toggle timed events': timedEventTest,
    'Ping node': pingNode,
    'Show hardware info': getHW
  }
};
```

5.9.2 Funktionmenysobjektets gränssnitt

När funktionmenyn skapas och initieras med en konfiguration returneras funktionsmenyobjektets gränssnitt. Gränssnittet innehåller metoder för att visa och gömma menyn, lägga till och ta bort funktioner ur menyn. Det går även att erhålla en referens till det element användaren senast klickade på och var orsaken till att menyn visades. Listning 5.10 på följande sida visar gränssnittet som returneras när funktionsmenyobjektet skapas.

Listning 5.10: Funktionsmenyns gränssnitt

```
    return {
            show: function( evt ) {
                    menu.setAttribute( 'display', 'block' );
                    gui.fadeIn( menu );
                    if (evt) currentElement = evt.target;
            },

            hide: function( evt ) {
                gui.fadeOut( menu, function() {
                    menu.setAttribute( 'display', 'none'); } );
                if (evt) currentElement = evt.target;
            },

             addFunctions: function( functions ) {
                 this.removeFunctions();
                 this.createFunctionGroup( functions, menuGroup
                     );
            },

             removeFunctions: function() {
                 menuGroup.removeChild( 'functionGroup' );
             },

            currentElement: function() { return currentElement;
               },
    }
```

5.10 Sammanfattning

I detta kapitel har jag redovisat implementationen av ett system för interaktiv visualisering av IP-nätverk. Implementationen bygger på kravspecifikationen i kapitel 2 på sidan 5 och resultatet av min problemanalys i kapitel 4 på sidan 12. Kapitlet började med att ge en översikt över systemet för att sedan i djupare detalj visa varje komponent. Av praktiska och utrymmesskäl har jag valt att enbart visa programkod för de viktigaste delarna av systemet.

I nästa kapitel utvärderar jag resultatet av implementationen och visar testningen av kraven i kravspecifikationen.

Kapitel 6

Testning och utvärdering

För att kunna avgöra om ett utvecklingsprojekt nått en punkt där det kan anses vara färdigt måste tester genomföras för att verifiera kravspecifikationen. Jag använde tester för att avgöra när varje del av systemet var färdigställt och därför var jag tvungen att i samband med framställandet av kravspecifikationen också definiera hur kraven i denna skulle testas. En lista med testerna redovisades i kapitel 2.3 på sidan 6. För att testerna ska kunna genomföras måste en representativ testmiljö sättas upp. En detaljerad beskrivning av testmiljön ges i avsnittet som behandlar testningen av kravspecifikationen.

Kapitlet inleds med att visa hur syftet med arbetet uppfyllts följt av en utvärdering av de designval jag gjort i implementationen av systemet. Avslutningsvis redovisar min testning av kravspecifikationen och sammanfattar hur väl systemet uppfyller kraven.

6.1 Uppfyllande av syfte

Syftet med detta arbete som jag skrev i inledningen var att utveckla ett prototypsystem som gör nätverkskartor interaktiva. Det skulle vara möjligt att anropa befintliga verktygsprogram via dessa kartor. I kapitel 5 på sidan 16 beskrev jag ett system som uppfyller följande:

- Genererar interaktiva nätverkskartor baserade på SVG-dokument och JavaScript.
- Tillåter anrop av CGI-skript på en webbserver genom att en användare interagerar med nätverkskartan.
- Visar resultat från körning av CGI-skript på webbservern i nätverkskartan.

De tre punkterna ovan anser jag tillsammans uppfyller syftet med detta arbete.

6.2 Utvärdering av designval

I kapitel 4 på sidan 12 beskrev jag de designval jag funnit för att lösa var och ett av de fyra delproblem jag brutit ned arbetet till. I detta avsnitt, som har

samma uppdelning som kapitel 4 på sidan 12, utvärderar jag de alternativ jag valde i implementationen av systemet.

6.2.1 Bindning av JavaScript-funktioner

I kapitel 5.6 på sidan 25 framgår det att jag valt alternativet att skapa ett temporärt SVG-dokument berikat med bindningar genom att transformera originaldokumentet genom XSLT på serversidan.

Alternativet att använda XSLT valdes för att originaldokumentet ska lämnas oberört. Detta innebär att originaldokumentet kan användas i andra tillämpningar som inte behöver vara beroende av hur detta system använder det. Ändringar i systemet kan därför göras utan att andra tillämpningar berörs.

En nackdel med den valda lösningen är att ett nytt SVG-dokument måste genereras varje gång en klient begär att få dokumentet för en specifik del av nätverket. Systemet har ej testats under last med många klienter som begär nätverkskartor. Under arbetet uppfattade jag genereringen av dokumenten som mycket snabb.

6.2.2 Hantering av användarinitierade händelser

När ett SVG-dokument transformeras med XSLT skapas nya attribut i utvalda elements attributlistor. Dessa attribut binder en händelse till en JavaScript-funktion. Systemet använder sig av flera expedieringsobjekt, ett per händelse och element. I kapitel 5.8 på sidan 28 visade jag implementationen av de expedieringsobjekt som används i systemet.

Dessa expedieringsobjekt kan hålla ett godtyckligt antal (inklusive noll) funktioner som ska utföras när en händelse avfyras. Detta innebär att systemet enkelt kan byggas ut genom att lägga till, ta bort och förändra de funktioner som expedieringsobjekten håller. Alternativet till att använda expedieringsobjekt för att ta hand om utlösta händelser är som jag skrev i kapitel 4.2 på sidan 14 att programmera en specialiserad funktion för varje element och händelse. Jag anser att det senare alternativet gör systemet svårare att utöka med ny funktionalitet. Utökningar kräver även ändringar i funktionernas programkod vilket kan leda till att nya defekter introduceras i funktionernas programkod.

De flesta händelser i systemet kräver dock enbart att en funktion utförs och expedieringsobjekten medför därför onödiga beräkningar i dessa fall. Expedieringsobjekten är enkla att utöka med nya funktioner under exekvering och jag anser att det är en stor fördel att hantera alla typer av händelser på samma sätt.

6.2.3 Anrop från klient till server

I kapitel 4.3 på sidan 14 visade jag två alternativ för hur asynkrona anrop från klient till server kan ske. Jag valde att använda det första alternativet med XMLHttpRequest-objektet .

XMLHttpRequest fungerar som vilket JavaScript-objekt som helst. Jag anser att XMLHttpRequest-objektet som visades i listning 5.5 på sidan 28 är enkelt att använda. Alternativet att använda ett script-element för asynkron kommunikation är inte aktuellt då det inte finns något behov att anropa serv-

rar med olika värdnamn. Begränsningen i vilka värdnamn som får användas vid serveranrop finns inte längre kvar i Firefox 3.5 och senare.

Ett problem med att använda asynkron kommunikation enligt den valda lösningen är att användaren ej kan se om klienten anropar servern via webbläsarens gränssnitt.

6.2.4 Behandling av anrop från klient

I kapitel 5.3 på sidan 20 visade jag att systemet har ett specifikt CGI-skript för varje funktion som användaren kan anropa från funktionsmenyn i användargränssnittet. CGI-skripten ansvarar för att ta emot och analysera anropet, utföra funktionen och returnera resultatet till klienten.

När systemet utökas med nya funktioner behöver inga tillägg eller ändringar göras i de befintliga programmen på servern. Det minskar risken att införa defekter i den befintliga programkoden.

En nackdel är att duplicering av programkod kan ske då flera CGI-skript hanterar klientens anrop på samma eller liknande sätt. Det kan vara svårt att överblicka systemet om det utökas med många CGI-skript. Det finns inga krav på hur GCI-skriptens gränssnitt ska se ut vilket kan medföra en risk att deras utformning skiljer sig helt mellan skripten. Detta i sin tur kan göra det svårare att underhålla systemet.

6.3 Uppfyllande av kravspecifikation

I detta avsnitt redovisar jag hur jag testat systemet mot kravspecifikationen och huruvida kraven blivit uppfyllda. Jag har på klientsidan använt följande testmiljö:

- Webbläsaren Firefox version 3.5 och 3.6.
- Microsoft Windows XP med Service Pack 3.
- Apple MacOSX version 10.6.3.

På serversidan har jag använt följande testmiljö:

- Debian 5.0.4 för 64-bitars arkitektur.
- Webbservern LightTPD 1.4.26.

6.3.1 Obligatoriska krav

K1 – Mjukvarupaketet GraphViz ska användas för att generera SVG-dokument

De SVG-dokument systemet hanterar är skapade av ett externt program som använder GraphViz.

K2 – Den grafiska representationen ska vara i formatet SVG

Användargränssnittet på klienten utgörs av ett SVG-dokument.

K3 – Applikationer på serversidan ska vara av typen CGI-skript skrivna i programmeringsspråket Perl

Alla applikationer på serversidan är skrivna i Perl och använder CGI.

K4 – Applikationer på klientsidan ska vara skrivna i programmeringsspråket JavaScript

All programkod i systemet som utförs på klienten är skriven i JavaScript.

K5 – Webbservern som används i systemet ska vara LightTPD

Systemet är utvecklat för och testat på webbservern LightTPD.

K6 – Systemet ska stödja webbläsaren Firefox version 3.5 eller senare

Systemet är utvecklat för Firefox version 3.6. Jag utförde de tillgängliga funktionerna i klientsidans gränssnitt i Firefox version 3.5 och version 3.6. Systemets funktionalitet skiljde sig inte mellan de två webbläsarversionerna och SVG-dokumentet renderades korrekt i bägge.

K7 – Ett befintligt verktygsprogram ska kunna anropas via användarinteraktion med SVG-dokument i webbläsaren

Detta krav är ej uppfyllt på grund av omständigheter som gjorde det omöjligt att testa systemet i uppdragsgivarens nätverk inom arbetets tidsram. Det går dock att exekvera ett godtyckligt program på serversidan genom att använda CGI. Programmet kan vara ett verktygsprogram. Uppdragsgivaren har godtagit detta.

K8 – Anrop enligt krav K7 ska ske asynkront

Alla anrop till servern sker genom att använda funktionen *doPlainXHR*. Funktionen använder XMLHttpRequest-objektets metod *open* med det förvalda värdet att operationen ska utföras asynkront. För att testa detta krav försäkrade jag mig först om att all programkod som genomför anrop till servern använder *doPlainXHR*. Efter det programmerade jag testprogram på serversidan som tog emot anropet, genomförde en paus under tre sekunder och returnerade en textsträng till klienten. Under tiden som testprogrammet exekverades på servern verifierade jag att det gick att interagera med nätverkskartan och utföra funktionerna i funktionsmenyn.

K9 – Resultatet av körningen av verktygsprogrammet ska visas i den webbläsare där anropet initierades

CGI-skripten på serversidan returnerar resultaten av körningen av verktygsprogram till den anropande klienten. Enligt RFC[1] 3875 [20] som beskriver CGI, ska en webbserver omvandla ett svar från ett CGI-skript till ett svar till den anropande klienten. Jag genomförde testade detta genom att utföra funktionerna i funktionsmenyn och notera om ett svar returnerades till webbläsaren och

[1] Request for comments. Ett dokument som beskriver ett förslag till standard.

visades i SVG-dokumentet. Precis som för krav K7 har detta krav ej testats med befintliga verktygsprogram men principen är densamma oavsett vilket program som utförs på serversidan.

K10 – Interaktion med SVG-dokument på klientsidan ska ej påverka originaldokumentet på serversidan

När en klient begär att få ett berikat SVG-dokument från webbservern genereras en temporär kopia av originaldokumentet. Interaktioner med dokumentet på klientsidan kan således ej påverka originaldokumentet. Jag testade detta krav genom att utföra de tillgängliga funktionerna på klientsidan och kontrollerade att originaldokumentet på servern var oförändrat.

K11 – Alla komponenter i systemet ska använda öppen mjukvara

De komponenter som ingår i systemet är alla av typen öppen mjukvara. Komponenterna är:

- Webbläsaren Firefox.
- Webbservern LightTPD med den inkluderade implementationen av CGI.
- Biblioteket XML::LibXSLT som inkluderar biblioteket libxslt.
- Mjukvarupaketet GraphViz.

6.3.2 Frivilliga krav

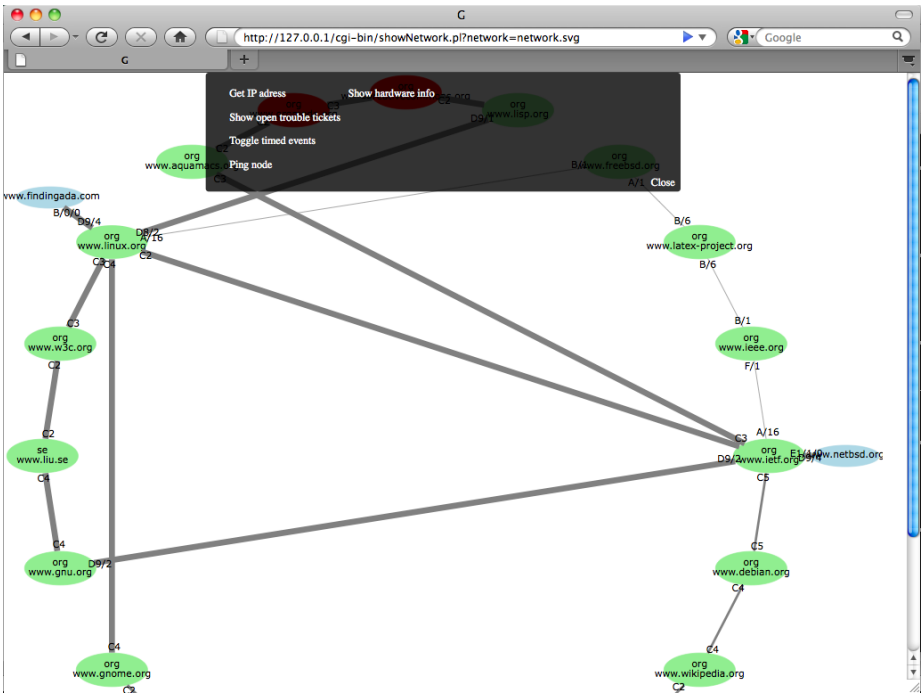
F1 – När en användare högerklickar på ett element i SVG-dokumentet ska en lista med tillgängliga funktioner visas

Kravet är ej uppfyllt men en liknande funktion implementerades. När en användare klickar på en nod i grafen visas en meny med funktioner (funktionsmenyn) i mitten av dokumentets övre del. Figur 6.1 på nästa sida visar funktionsmenyn i användargränssnittet.

Att visa en meny på den position i dokumentet där användaren högerklickar är problematiskt.

Problemet grundas i hur x- och y-koordinater används i webbläsaren och dokumentet. När ett nytt element ska läggas till anges dess x- och y-koordinater relativt dokumentets övre vänstra hörn. När en händelse avfyras i webbläsaren innehåller händelseobjektet x- och y-koordinater som är relativa med webbläsarfönstrets övre vänstra hörn. Om dokumentet förstoras så att det inte ryms i webbläsarfönstret kan det flyttas med hjälp av rullningslister i webbläsaren. Sker detta går det inte att använda de koordinater som händelseobjektet innehåller för att skapa ett nytt element på platsen användaren klickade. Händelseobjektets koordinater och dokumentets koordinater tillhör olika koordinatsystem. Koordinaternas värden anger pixlar på skärmen.

Firefox tillhandahåller värden för hur många pixlar ett dokument flyttats med rullningslisterna. Jag antog att det med dessa värden gick att översätta de koordinater som händelseobjektet innehöll till samma koordinatsystem som dokumentet använde.



Figur 6.1: Skärmavbild som visar funktionsmenyn

Jag tog fram denna formel där d står för dokument, h för händelse och f för förflyttning:

$$(x, y)_d = (x_h + x_f, y_h + y_f)$$

Dessvärre är ekvationen ej var sann för varje värde f. Ju mer dokumentet flyttas i webbläsaren desto större blir felet. Detta innebär att menyn visas längre från muspekaren ju mer dokumentet flyttas i webbläsarfönstret.

F2 – Listan i krav F1 ska genereras baserat på elementets identifierare

Detta krav har ej uppfyllts då jag anser att det tar för lång tid att utveckla ett system för att knyta en viss typ av hårdvara till en given mängd funktioner.

F3 – Systemet ska innehålla funktionalitet för att automatiskt anropa ett verktygsprogram baserat på en timer

Detta krav har ej uppfyllts då det inte har getts möjlighet att anropa verktygsprogram från webbservern. Jag har utvecklat en timer som kan anropa ett godtyckligt antal JavaScript-funktioner på klientsidan. En anropad funktion kan i sin tur anropa ett program på serversidan.

6.4 Sammanfattning

Systemet som beskrevs i kapitel 5 på sidan 16 uppfyllde de obligatoriska kraven K1 till och med K12 med undantag för krav K7. Detta krav är ej uppfyllt på grund av omständigheter som gjorde det omöjligt att testa systemet i uppdragsgivarens nätverk inom arbetets tidsram. CGI-skript på webbservern kan dock exekvera ett godtyckligt program på serversidan vilket innebär att det inte finns några tekniska begränsningar i att koppla ihop verktygsprogrammen med systemet.

Inget av de frivilliga kraven uppfylldes enligt kravspecifikationen. Istället för att visa en meny med funktioner för användaren när denne högerklickar på ett element i nätverkskartan enligt krav F1, visas menyn högst upp i mitten av nätverkskartan. Gällande krav F3 så implementerade jag en timerfunktion enligt kravspecifikationen. Jag implementerade en timerfunktion enligt krav F3 men kunde tyvärr inte uppfylla kravet helt då jag ej kopplade ihop timern med befintliga verktygsprogram.

Kapitel 7

Diskussion och slutsatser

I detta kapitel sammanfattar jag det arbete som beskrivits rapporten. Jag inleder kapitlet med en diskussion rörande systemet. Därefter kommenterar jag valet av metod för arbetet följt av en diskussion om hur systemet skulle kunna vidareutvecklas. Avslutningsvis nämner jag några av de erfarenheter jag fått under arbetet.

7.1 Diskussion

Mitt mål med detta arbete var att ett system för att möjliggöra interaktiv visualisering av IP-nätverk och koppla ihop noder i nätverket med befintliga verktygsprogram. Arbetet resulterade i:

- Ett XSL-program som berikar ett befintligt SVG-dokument skapat genom GraphViz med bindningar till JavaScript-funktioner.
- Tre JavaScript-bibliotek med funktioner som gör nätverkskartan interaktiv och kopplar ihop denna med CGI-skript på webbservern.
- CGI-skript för att hantera anrop från webbläsaren och utföra önskade funktioner.
- En CSS-stilmall som styr utseendet av nätverkskartan.

Som jag nämnde i kapitel 6 på sidan 33, uppfyllde systemet alla obligatoriska krav i kravspecifikationen förutom kravet att befintliga verktygsprogram ska kunna anropas. Det finns inga tekniska begränsningar i systemet som omöjliggör anrop av verktygsprogrammen. Från ett CGI-skript på webbservern kan godtyckliga program exekveras.

Systemet är helt uppbyggt på öppen mjukvara och öppna standarder. Det har flera fördelar:

- Komponenterna i systemet är gratis att använda.
- All källkod i systemet är tillgänglig och kan undersökas.
- Öppen mjukvara är ofta väldigt robust.

- Genom att använda öppna standarder är det om inte enkelt åtminstone möjligt att byta ut komponenter i det.
- Öppna standarder minskar risken att låsa sig till en viss leverantör.

Som kunde läsas i kapitel 4 på sidan 12, behandlas enbart två olika XSL-transformerare i rapporten. En mer uttömmande undersökning i området skulle eventuellt kunna tillföra fler alternativ till dessa. Dock anser jag det tveksamt om något annat alternativ skulle ge bättre funktionalitet och vara enklare att använda.

Användbarheten i att presentera resultatet av en programkörning på servern i en SVG-baserad dialogruta kan ifrågasättas. Eftersom texten som presenteras i denna är vektorgrafik går det ej att markera och kopiera den som det gör i HTML-baserade applikationer. Eftersom text i SVG version 1.0 [35] hanteras som vilket grafiskt element som helst så måste den positioneras med x- och y-koordinater. Det finns alltså ingen automatisk funktion som inför radbrytningar när en textrad överstiger en given längd vilket gör det besvärligt att anpassa dynamiskt genererad data till dialogrutans storlek.

Arbetet har förenklats avsevärt tack vare att systemet enbart behöver stödja en webbläsare. En av de stora svårigheterna med att utveckla webbapplikationer är att göra dem kompatibla med de populäraste webbläsarna. Detta avspeglas på de talrika JavaScript-bibliotek som skapats för att underlätta detta arbete för utvecklaren. Detta belyser vikten av att utvecklare av webbläsare och webbapplikationer följer öppna standarder i så stor utsträckning som möjligt.

Programkod som ej redovisats i rapporten kan sändas vid önskemål genom att kontakta författaren.

7.2 Metodfrågor

Som jag beskrev i det inledande kapitlet valde jag att använda George Pólyas problemlösningsmetod för att strukturera arbetet. Metoden fungerade mycket bra för att skapa en övergripande struktur. Under utvecklingsfasen saknade jag en formell metod för utveckling. Det var Ibland svårt att överblicka hur långt jag kommit i implementationen av systemet och lade tidvis ned för mycket tid på att putsa på vissa mindre viktiga delar av systemet.

Min plan att använda en prioriterad lista med lösningar på problem som skulle lösas i arbetet fungerade inte väl i praktiken. Lösningarna i planen var dåligt definierade och kunde inte användas som måttstock för att avgöra om ett specifikt problem var löst. Jag använde istället kravspecifikationen för att avgöra detta.

7.3 Framtida arbete

Systemet som implementerades i detta arbete kan användas som en grund för att vidareutveckla ett mer komplett system för interaktiv visualisering av IP-nätverk. Systemets källkod innehåller många exempel på hur det kan byggas ut med nya funktioner. Arbetet som var av explorativ art kan också ses som en förstudie för ett större projekt där detta system fungerar som en del i ett större system.

Nedan beskriver jag de områden som arbetet ej berörde och begränsningar som behöver tas i akt vid eventuell vidareutveckling av systemet.

7.3.1 Kompabilitet med fler webbläsare

Systemet är idag utvecklat för Firefox version 3.5 eller senare. Systemet fungerar även bra i Opera version 10 men inte alls i Internet Explorer. Om användaren inte ska tvingas att använda vissa utvalda webbläsare måste systemet göras mer flexibelt så att det fungerar lika bra eller åtminstone acceptabelt i alla de stora webbläsarna. Framförallt måste hanteringen av asynkrona anrop göras om då implementationen av objekt med samma funktionalitet som XMLHttpRequest kan skilja sig åt mellan webbläsare.

7.3.2 Utveckling av funktionsmenyn

Funktionsmenyn är något begränsad vad gäller flexibilitet. Det är inte tillräckligt enkelt att låta menyn fyllas med ett godtyckligt antal funktioner då den saknar funktionalitet för att dynamiskt positionera text.

Menyn visas alltid högst upp i dokumentet. Om en användare förstorar och flyttar dokumentet i webbläsarfönstret är det möjligt att den inte syns. Det vore önskvärt att funktionsmenyn automatiskt skulle flyttas till webbläsarfönstrets övre kant när en användare flyttar dokumentet i det. Automatisk flyttning av funktionsmenyn skulle kunna lösas genom att i JavaScript definiera funktioner som lyssnar efter händelserna musklick och förflyttning av SVG-dokumentet i webbläsaren. När dokumentet flyttas i webbläsarfönstret ska funktionerna uppdatera funktionsfönstrets y-koordinat så att den har samma värde som y-koordinaten för den synliga delen av dokumentet.

7.3.3 Säkerhet

Anrop från klienten filtreras ej av de CGI-skript som tar emot dem på webbservern. Detta är en potentiell säkerhetslucka då en noggrant skriven URL eventuellt skulle kunna orsaka exekvering av godtyckliga program. Om känslig data transporteras mellan klient och server måste förbindelsen göras säker med tekniker som till exempel TLS[1].

7.3.4 Sammankoppling med befintliga verktygsprogram

Systemet innehåller ingen koppling till befintliga verktygsprogram. Om systemet installeras i uppdragsgivarens interna nätverk är det möjligt att från CGI-skripten anropa verktygsprogrammen.

7.4 Egna erfarenheter

Innan detta arbete påbörjades hade jag väldigt liten erfarenhet av JavaScript, CSS, XML och XSLT. Under förstudien upptäckte jag att teknikerna var mycket väldokumenterade i olika böcker och framförallt på Internet. Det gick därför snabbt att skapa förståelse för området.

[1] Transport Layer Security

I början av arbetet skapade jag en projektplan som innehöll en tidsplan med milstolpar. Projektplanen gjorde det möjligt att strukturera arbetet på ett bra sätt. En bra projektplan anser jag vara en förutsättning för ett lyckat projekt. Tidsplanen var ett mycket bra stöd under implementationen av systemet då jag enkelt kunde se hur jag låg till tidsmässigt.

Under arbetets gång har jag fört journal. I journalen har jag antecknat referenser till bra källmaterial och de problem som uppstått under arbetets gång och olika alternativ för att lösa dessa. Journalen har fungerat bra som ett minnesstöd under skrivandet av denna rapport.

Arbetet med implementeringen av systemet och framställandet av denna rapport har varit väldigt givande och gett mig många erfarenheter som jag för med mig i framtida uppdrag och studier.

Litteraturförteckning

[1] Telenor, "Om Telenor." http://www.telenor.se/privat/om-telenor/index.html, july 2010.

[2] Bredbandsbolaget, "Om Bredbandsbolaget." http://www.bredbandsbolaget.se/omoss/index.html, july 2010.

[3] G. Pólya, *How to solve it.* Princeton University Press, februari 1973.

[4] Open Source Initiative, "The Open Source Definition." http://opensource.org/docs/osd, 2010.

[5] W3C, "Scalable Vector Graphics (SVG) 1.0 specification." http://www.w3.org/TR/SVG10/index.html, september 2001.

[6] AT&T, "GraphViz." http://www2.research.att.com/areas/visualization/projects_software/graphviz.php, april 2010.

[7] S. N. Emden R. Gasner, Eleftherios Koutsofios, *Drawing graphs with* dot, december 2009.

[8] J. J. Garrett, "Ajax: A new approach to web applications." http://www.adaptivepath.com/ideas/essays/archives/000385.php, februari 2005.

[9] W3C, "XHTML^{TM} 1.0 The Extensible HyperText Markup Language (Second Edition)." http://www.w3.org/TR/xhtml1/, augusti 2002.

[10] W3C, "Cascading Style Sheets." http://www.w3.org/Style/CSS/, mars 2010.

[11] W3C, "Document Object Model (DOM) level 2 core specification." http://www.w3.org/TR/DOM-Level-2-Core, november 2000.

[12] W3C, "Extensible Markup Language." http://www.w3.org/XML/, mars 2010.

[13] W3C, "XSL transformations (XSLT)." http://www.w3.org/TR/xslt, november 1999.

[14] W3C, "XMLHttpRequest." http://www.w3.org/TR/XMLHttpRequest/, november 2009.

[15] Mozilla Developer Center, "XMLHttpRequest." https://developer.mozilla.org/en/XMLHttpRequest, mars 2010.

[16] M. D. Center, "About javascript." https://developer.mozilla.org/en/About_JavaScript, april 2010.

[17] D. Crockford, *JavaScript: The Good Parts*. O'Reilly, maj 2008.

[18] Okänd, "LIGHTTPD." http://www.lighttpd.net/, februari 2010.

[19] D. Connolly, "CGI: Common Gateway Interface." http://www.w3.org/CGI/, maj 2009.

[20] K. C. D. Robinson, "The common gateway interface (CGI) version 1.1." http://www.ietf.org/rfc/rfc3875.txt, oktober 2004.

[21] J. D. Eisenberg, *SVG Essentials*. O'Reilly, februari 2002.

[22] W3C, "Scalable Vector Graphics (SVG) 1.0 specification." http://www.w3.org/TR/SVG10/script.html, september 2001.

[23] W3C, "ECMAScript Language Binding." http://www.w3.org/TR/DOM-Level-2-Core/ecma-script-binding.html, november 2000.

[24] Microsoft, "Deploying XSLT in Internet Explorer." http://msdn.microsoft.com/en-us/library/ms760279(VS.85).aspx, 2010.

[25] Mozilla Developer Center, "Gecko FAQ." https://developer.mozilla.org/en/Gecko_FAQ, februari 2010.

[26] Opera Software, "Web specifications supported in Opera Presto 2.5." http://www.opera.com/docs/specs/presto25/#xml, april 2010.

[27] The Webkit Open Source Project, "XSLT." http://webkit.org/projects/xslt/index.html, 2010.

[28] Apache Software Foundation, "The Apache Xalan Project." http://xalan.apache.org/, 2005.

[29] P. Pajas, "XML::LibXSLT." http://search.cpan.org/dist/XML-LibXSLT/LibXSLT.pm, april 2010.

[30] D. Veillard, "The XSLT C library for GNOME." http://xmlsoft.org/XSLT/, april 2010.

[31] W3C, "SVG Document Object Model (DOM)." http://www.w3.org/TR/SVG/svgdom.html, januari 2003.

[32] W3C, "Cross-Origin Resource Sharing." http://www.w3.org/TR/cors/, mars 2009.

[33] Mozilla Developer Center, "HTTP access control." https://developer.mozilla.org/En/HTTP_Access_Control, januari 2010.

[34] J. Levitt, "Fixing AJAX: XMLHttpRequest considered harmful." http://www.xml.com/pub/a/2005/11/09/fixing-ajax-xmlhttprequest-considered-harmful.html, november 2005.

[35] J. D. Eisenberg, *SVG Essentials*. O'Reilly, februari 2002.