Simulering af menneskelig opførsel i en grafbaseret repræsentation af et virkeligt scenarie

Mikkel Kaas Pedersen

Kongens Lyngby 2012
IMM-B.eng-2012-35

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.eng-2012-35

Contents

Chapter 1

Indledning

Det er interessant at se på adgangskontrol i miljøer som f.eks. kontorlandskaber. Hvem har adgang til hvilke dele af bygningen, hvilket data har de adgang til og så fremdeles. Når der sker sikkerhedsbrister i sådanne miljøer, er det tit insidere, der har adgang til for meget data og forkerte ressourcer, som bliver udnyttet. For at undgå, at data bliver misbrugt, er det interessant at se på hvordan man kan efterprøve adgangskontrolsystemer i et miljø.

Denne rapport omhandler opbygningen af et system bestående af en server og klient, der kan vise en graf, der er en repræsentation af et virkeligt miljø. I grafen kan der være knuder, der er adgangskontrollerede, så kun personer, der opfylder bestemte krav, har adgang dertil.

Det færdige system vil indeholde følgende komponenter:

Server: Serveren udvikles i C#. Denne indlæser grafer fra XML-filer og sender graferne til klienten. Serveren simulerer også handlinger i grafen. Disse sendes også til klienten.

Klient: Klienten udvikles i HTML5/Javascript. Klienten vises er grafen til brugeren, som kan aflæse information om denne. Klienten kan sende besked til serveren, om at serveren skal begynde at sende handlinger til klienten.

Server og klient kommunikerer via JSON-strenge via en websocket-forbindelse.

1.1 Rapporten

Først laves en analyse af systemet, der skal udvikles. Derefter beskrives de overordnede linjer i systemet. Efterfølgende gennemgås selve implementeringen. Ikke alt kode er beskrevet ligeligt. Trivielt kode er undladt og det interessante er fremhævet. Derefter følger en brugervejledning og i appendiks er kodestumper, der er for store til at have med i implementeringsafsnittet, samt XSD-filer.

Selve implementeringen er at finde som bilag.

Chapter 2

Baggrund

Dette projekt tager udgangspunkt i artiklerne "An extensible analysable system model" [1] og "Where Can an Insider Attack" [2].

Artiklerne tager udgangspunkt i problemstillingen, at en insider har bedre adgang til informationer om systemet og dettes ressourcer. Dette kan misbruges af insideren, der har mulighed for ubemærket at indsamle data, og dermed udgøre en sikkerhedsrisiko.
Disse artikler beskriver hvordan et virkeligt system, og de sikkerhedsaspekter der er forbundet med dette, kan modelleres som en graf.

I Figur 2.1 ses en skitse af et kontorlandskab. Dette kontorlandskab kan repræsenteres ved en mængde af data:

Infrastruktur: Rum, døre, computere, låse etc. kan moddeleres som **knuder** i systemet. Imellem disse knuder er der **forbindelser**. Forbindelser går kun én vej - altså en forbindelse fra punkt A til punkt B er ikke ensbetydende med en forbindelse fra punkt B til punkt A.

Aktører: Personer der udfører handlinger på systemet - går ind af døre, bruger en computer, printer etc.

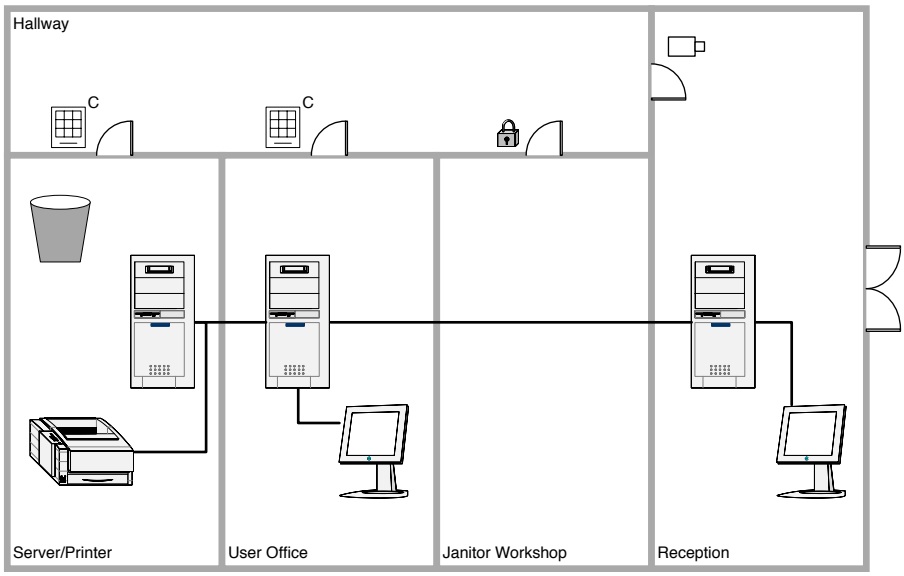


Figure 2.1: Skitse af et kontorlandskab.

Data: Data kan både være noget personer bærer rundt på og data på knuder. F.eks. kunne data på døren ind "Janitor workshop" være, at der skal bruges en bestemt nøgle til at åbne døren.

Handlinger: Handlinger kan være, at en person flytter sig fra en knude til en anden. Til dette er det nødvendigt at vide hvem der flytter sig hvorfra og hvortil. En handling kan også være at udskrive på printeren, samle data op og flere andre ting.

Adgangskontrol: Der kan være adgangskontrol tilknyttet de forskellige knuder i grafen. Dette kunne f.eks. være, at "janitor" har nøglen til "Janitor workshop" og receptionisten har adgang til at printe på printeren.

Logging: Grafen kan have logging, så handlinger der udføres på systemet, bliver husket. Steder, hvor handlinger bliver logget, er markeret med en overstreget notation i den visuelle repræsentation af grafen.

I Figur 2.2 ses hvordan kontorlandskabet fra Figur 2.1 kan modelleres som en graf vha. ovenstående definitioner.

I artiklerne er også beskrevet hvordan det er muligt at gennemgå hele grafen, for at se alle de handlinger, som alle aktører har adgang til at udføre.

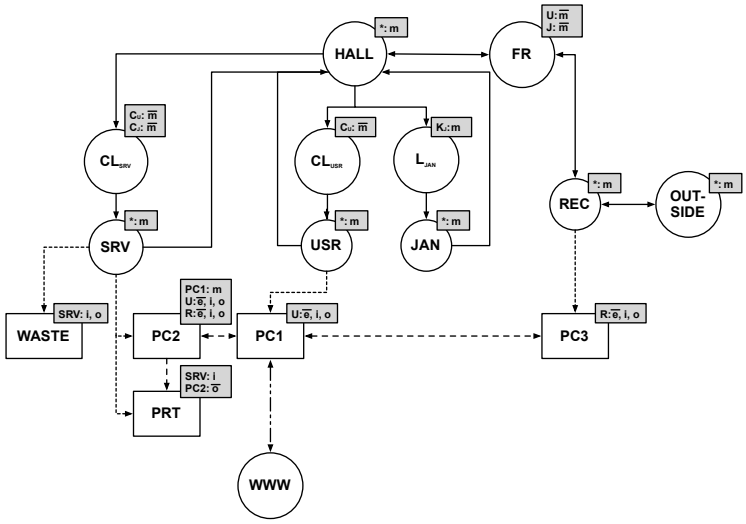


Figure 2.2: Graf over kontorlandskabet fra Figur 2.1.

Systemet, der udvikles til dette projekt, er opbygget ud fra disse artikler og tankerne med aktører, handlinger, datakontrol og logging fra disse er ført ind i systemet. Systemet er ikke opbygget 100% som beskrevet i artiklerne. Data bliver gemt i andre formater, logging er anderledes og meget andet.

2.1 Lyngby Svømmehal

Lyngby svømmehal er blevet brugt til at danne en eksempel-graf, der bliver brugt hele vejen igennem udviklingsprocessen, i brugervejledning og de fleste tests. Lyngby svømmehal indeholder flere forskellige ting, der gør denne velegnet som eksempel i dette system. Den indeholder dele, hvor kun nogle ansatte har adgang, den indeholder sluser, hvor man skal have billet for at komme ind og herre- og dameomklædningsrum. Det er muligt at lave cirkulære grafer og døde ender. En skitse af Lyngby Svømmehal er vist i Figur 2.3. Grafen, der afspejler dette er vist i Figur 2.4 på side 7. Rødt markerer område kun for ansatte, lyseblå herreomklædningen og lyserød dameomklædningen.

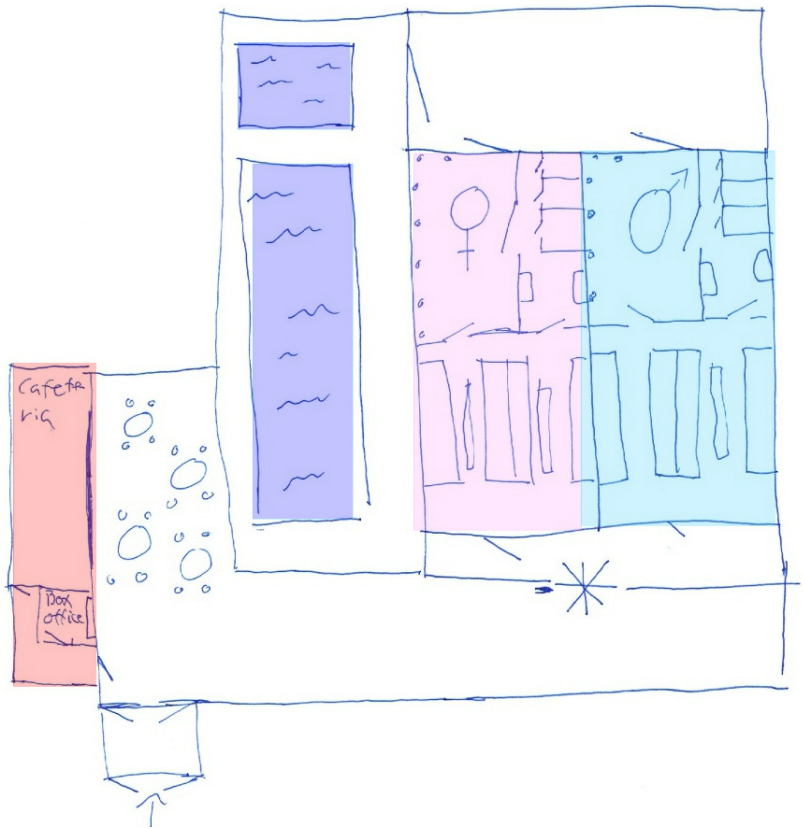


Figure 2.3: Skitse af Lyngby svømmehal.

2.2 C♯

C# er et programmeringssprog udviklet af Microsoft. Første version udkom i 2001. C#s udviklingshold er ledet af danskeren Anders Hejlsberg. C# er nemt at gå til - Det ligner C++ og Java meget i opbygning, så har man erfaring med disse, er det nemt at udvikle i C#. C# bliver primært brugt på windows-platformen, men det er også muligt at afvikle C#-kode på Linux vha. Mono-projektet, der er drevet af Novell[1]

Navnet C# er afledt af programmeringssproget "C". I musikkens verden bruges # ("sharp") til at angive at en tone er løftet en halv tone. Altså er C# C løftet en halv.

[1] http://www.mono-project.com

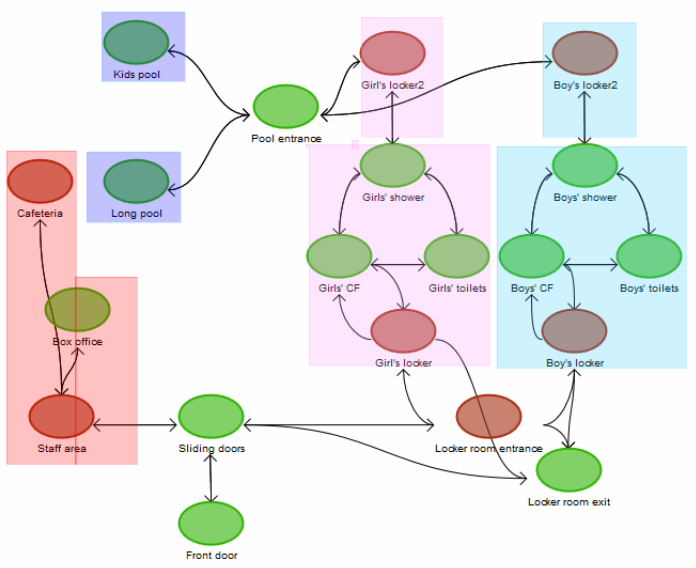


Figure 2.4: Lyngby svømmehal repræsenteret ved en graf

2.3 W3C

World Wide Web Consortium (W3C)[2] er en organisation, der holder styr på de mange forskellige standarder, der findes på internettet. De sørger for at drive udviklingen af nye standarder fremad. W3C består af lidt under 400 forskellige medlemmer/organisationer, der alle kan foreslå nye standarder. Det er en lang process der skal til, før at noget bliver godkendt som en standard af W3C:

Indsendelse af forslag: Et af medlemmerne indsender et forslag.

Arbejdsgruppe nedsættes: Ud fra forslaget går flere interesserede medlemmer sammen om at lave en arbejdsgruppe, der skal arbejde med forslaget.

Arbejdsudkast: Arbejdsgruppen udgiver et "working draft", som er det første oplæg til en standard. Denne skal dog ikke bruges som referencemateriale, da denne helt sikker bliver ændret.

Anbefalingsudkast: Her er standarden mere udviklet - ting er blevet afprøvet og har fået tid til at modne. Dette er dog stadig kun et udkast og skal ikke bruges i et produktionsmiljø.

[2] http://w3.org/

Foreslået udkast: Standarden er ved at være færdigudviklet. Standarden kan stadig ændre sig, men den er så tæt på at være færdig, at dette højest sandsynlig ikke sker. I dette begynder mange at implementere det i produktionsmiljøer, da det kun er småting der vil ændre sig inden sidste stadie.

W3C Anbefaling: Standarden er her færdiggjort og blevet officiel. Denne er nu at betegnes som et referencemateriale.

2.4 HTML

HyperText Markup Language (HTML) er grundlaget for websider på internettet. Disse bliver vist i en browser. HTML bliver brugt til at beskrive indholdet på en webside - dvs. tekst, links, billeder m.m.

Ud over tekst og billeder m.m., som bliver vist for brugeren, indeholder websider også data, der ikke bliver vist, men fortæller noget om siden. F.eks. hvem har lavet siden, hvilke ting er blevet brugt, hvordan skal tegn fortolkes og meget mere. Alt dette er kaldet metadata.

2.4.1 Historie

W3C frigav HTML4-specifikationerne i december 1999. Næsten samtidig blev XML 1.0-specifikationerne ud givet af W3C. Det var hurtigt tydeligt, at HTML4 ikke nemt kunne bliver omdannet til XML1.0 - HTML4-specificeringen var simpelthen for løs. Denne var meget overbærende over for f.eks. rækkefølgen af elementer - Man behøvede ikke lukke tags i den række følge de blev oprettet i osv.

Dette ledte til udviklingen af XHTML1-standarden (og senere hen 1.1), som gjorde op med dette. Nu var der en standard, der indeholdt mange ting, der kunne skabe en god weboplevelse, samtidig med at en streng XML-standard blev overholdt.

2.4.2 HTML5

I januar 2008 frigav W3C første arbejdsudkast til HTML5-standarden. Siden hen er der kommet flere anbefalingsudkast. Det seneste i december 2012[3]. HTML5 er altså endnu ikke en officiel stndard, men er på vej til at blive det. Standarden bliver allerede brugt mange steder på internettet, selvom denne endnu ikke er officiel og færdigudviklet.

HTML5 er forskellige fra HTML på mange punkter. Elementer har en striks stak-lignende struktur - elementer skal lukkes i den rækkefølge de er blevet åbnet i. Alle elementer og attributter skal skrives med småt. Enkeltstående elementer (f.eks. `<br />` for ny linje) skal alle afsluttes med en / og meget mere.
HTML5 indeholder også mange nye elementer, f.eks. `<article>`, `<header>`,`<audio>` og `<nav>`.

HTML5 står til at blive frigivet som endelig standard i 2014.

2.4.3 CSS(3)

HTML indeholder ikke synderligt meget information om hvordan ting skal præsenteres for brugeren. Til dette bruges CSS[4]. CSS bruges til at beskrive hvor tekst vises, hvilken baggrundsfarve en side har, at der skal en kant rundt om noget tekst, og denne skal have afrundede hjørner - altså alt der har med udseende at gøre.

På samme måde som der er flere forskellige HTML-standarder, findes der flere forskellige CSS-standarder. Den sidste officielle standard er CSS2.1 fra juni 2011[5]. Denne har dog kun få ændringer fra CSS2, frigivet maj 1998[6].

Den næste version af CSS, CSS3, bliver frigivet i "moduler" - der er over 50. Ind til videre er der fire af disse, der er officielle standarder: Media Queries, Namespaces, Selectors Level 3 og Color. Af nye ting i CSS3 kan nævnes nye skrifttyper, afrundede hjørner, gennemsigtighed i farver og meget mere.

[3] http://www.w3.org/TR/2012/CR-html5-20121217/
[4] Cascading Style Sheets
[5] http://www.w3.org/TR/2011/REC-CSS2-20110607/
[6] http://www.w3.org/TR/2008/REC-CSS2-20080411/

2.4.4 Javascript

Javascript er et programmeringssprog, der (for det meste) kører i en brugers webbrowser.

HTML-sider er statiske. Ved at bruge Javascript er det muligt at skabe en dynamisk hjemmeside - en hjemmeside der virker "levende" for brugeren ved at f.eks. validere indtastninger, vise popup-bokse, animere ting m.m.

Javascript startede sit liv i starten af 90érne som "Livescript" i Netscape Navigator, der på dette tidspunkt var den klart mest brugte webbrowser. På dette tidspunkt vandt programmeringssproget "Java" frem i verden og Netscape ville gerne bruge lidt af denne succes, så derfor blev Livescript hurtigt omdøbt til Javascript - dog uden at Java og Javascript har noget med hinanden at gøre.
For ikke at falde for langt bagud, fik Internet Explorer snart understøttelse af to forskellige script-sprog - VBScript og JScript. JScript lignede til forveksling Javascript.

Da Netscape Navigator havde en klart større brugerskare end Internet Explorer, bevægede Internet Explorers JScript sig stille og roligt i retningen af Javascript. Som tiden gik, blev Internet Explorers markedsandele større og større og JScript havde bevæget sig så meget imod Javascript, at disse to i praksis var ens.
I 1996 fik organisationen ECMA[7] overdraget ansvaret for Javascript og dets efterfølgende udvikling. Som resultat heraf skiftede Javascript officielt navn til ECMAscript. Dog har dette navn aldrig slået igennem og Javascript er stadig det generiske term for sproget.

2.4.5 WebSockets

Når klienter/websider skal kommunikere asynkront med serveren, foregår dette tit via AJAX. Dette fungerer ved at klienten sender en forespørgsel til serveren, der sender et svar tilbage. Såkaldt "pulling". For at serveren kan sende data til websiden, kræver det altså at websiden har forespurgt det. Det er altså ikke muligt for serveren at sende noget til klienten, uden at klienten har bedt om det. En måde at opnå "push"-funktionalitet på, hvor serveren selv kan sende noget til klienten, når der er noget at sende, er at benytte "long polling". Her sender klienten en forespørgsel til serveren. Hvis ikke serveren har noget data at sende, så venter serveren med at sende et svar, til der rent faktisk er noget at sende. Når dette er modtaget af klienten, så forespørger klienten igen serveren efter data,

[7] http://www.ecma-international.org/

som så igen leverer data, når dette er klart. Dette ligner "push"-forespørgsler, men er det ikke.

Heldigvis er der med HTML5 kommet "websockets". Når en klient opretter en websocket-forbindelse til serveren, så er der etableret en full-duplex-kommunikationsvej via en TCP-socket. Dermed kan serveren pushe data til klienten asynkront og ægte push-funktionalitet er opnået. Websockets bliver eksponeret til HTML via et Javascript-interface.

Chapter 3

Kravanalyse

Der ønskes udviklet et system, der kan bruges til at vise en graf, der repræsenterer et fysisk miljø, f.eks. en kontorbygning.

I systemet kan døre, printere, computere etc. vises som knuder. Disse knuder kan enten være åbne for alle, eller være adgangskontrollerede. F.eks. skal alle have adgang til en printer, og en kok have adgang til køkkenet, men ikke til direktørens kontor.

Systemet skal være opdelt i to enheder - en server og en klient.

Klienten står for den grafiske repræsentation til brugeren. Klienten skal ikke have kendskab til rettigheder og handlinger, der bliver udført. Den skal blot vise det, som serveren sender, og antage at alle handlinger, som serveren sender, er lovlige.

Serveren skal levere grafer til klienten. Serveren skal holde styr på hvilke personer, der er i grafen, hvilke knuder de har adgang til, samt hvor personerne bevæger sig hen i grafen.

Serveren og klienten kommunikerer med hinanden via et netværk.

3.1 Graf

Grafen er omdrejningspunktet i systemet. Det er denne, der indeholder alt information, der er relevant for at kunne repræsentere et fysisk miljø

- Navn
- Hvem har senest opdateret filen
- Hvornår er den genereret
- En liste af et vilkårligt antal knuder
- En liste af et vilkårligt antal kanter

3.1.1 Knude

En knude skal have information om

- Id
- Navn
- Beskrivelse
- Position
- Er knuden adgangskontrolleret? - Hvis ja
  - Liste over nøgler, der giver adgang
  - Liste over personer, der har adgang

3.1.2 Kant

I et rigtigt miljø er det ikke altid muligt at gå fra B til A, selvom det er muligt at gå fra A til B. Tænk f.eks. på svingdørene i et supermarked eller en udgangsmølle.
En kant skal derfor være retningsbestemt – dvs. at den går fra A til B, men ikke den anden vej. Ønsker man muligheden for at gå fra B til A, skal der oprettes en kant der går fra B til A.

En kant skal have information om

- Start-knude
- Slut-knude

3.2 Personer

Systemet skal kunne indlæse en liste af personer. Til hver person er tilknyttet

- Id
- Liste af nøgler

3.3 Handlinger

Der skal kunne udføres to typer handlinger på grafen. Begge handlinger tager udgangspunkt i start-knuden.

En **dybde-først-søgning**, hvor hele grafen løbes igennem for hver person, i de knuder personen har adgang til.

En **tilfældig søgning**, hvor der vælges en tilfældig knude. Personen går tilfældigt rundt i mellem de knuder, som denne har adgang til, ind til den ønskede knude er fundet, eller der er gået for lang tid.

3.4 Klient

For ikke at afgrænse systemets brug, skal der udvikles en klient, der kan køre på de fleste tilgængelige platforme. Dette gøres ved at implementere klienten som en webside.

Klienten skal kunne køre på alle nyere browsere, der understøtter HTML5 Websockets. Dette er alle nyere versioner af

- Firefox
- Google Chrome
- Safari

Internet Explorer er ikke understøttet.
Klienten vil være stærkt afhængig af Javascript, hvorfor dette også er et krav til browseren.

3.5 Server

Serveren skal kunne køres på en windows-computer. Serveren skal være "autonom" - altså virke uden nogen indgriben fra en bruger.

Serveren skal kunne håndtere flere samtidige forbindelser fra klienter.

3.6 Performance

Det er vigtigt, at systemet ikke har lange svartider. Der kan være flere årsager til lange svartider. Nogle af disse kan der optimeres på, andre kan der ikke.

- **Graf-søgning** Når der skal søges igennem grafen, er der flere forskellige algoritmer at bruge. Hvis implementeringen er dårlig, kan en grafsøgning have lang køretid
- **Forbindelsen** Hvis netværksforbindelsen imellem klienten og serveren er langsom, vil dette have indflydelse på svartiderne i programmet.

Chapter 4

Overordnet design

Systemet er opbygget i to uafhængige komponenter - Serveren og klienten. Disse to komponenter kan køre på samme computer, eller på to forskellige computere med et netværk imellem.

En sketch af den overordnede opbygning ses i figur 4.1.

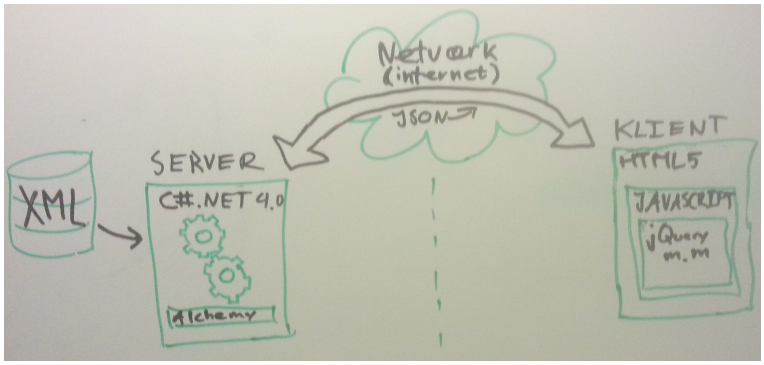


Figure 4.1: Tegning af overordnet opbygning.

4.1 Klient

Klienten udvikles som en webapplikation. Klienten er baseret på HTML5, CSS, Javascript og SVG-grafik. Der er flere grunde til at det er valgt at udvikle klienten på denne måde

- Ingen installation hos brugeren
- Systemuafhængighed - Klienten virker både på Linux, Mac OS X, Windows, iOS m.m.
- Hurtig og nem udrulning af opdateringer. Da brugerne navigerer til en webside, skal der ikke udrulles opdateringer til brugernes lokale maskine – siden skal kun opdateres ét sted.
- Gode Javascript-biblioteker letter udviklingsarbejdet.

4.1.1 Javascript-biblioteker

Javascript-biblioteker er blevet meget populære at bruge. Der er flere gode grunde til at gøre brug af Javascript-biblioteker.
De sikrer ensformighed på tværs af browsere og hjælper med at udvide Javascripts indbyggede funktioner. Dette gør det meget nemmere at håndtere events, kommunikation, animationer og DOM[1] -behandlinger. F.eks.

```
1 \\"Ren javascript":
2 document.getElementbyId('testDiv').style.background = "blue";
3
4 \\jQuery:
5 $("#testDiv").css("background", "blue");
```

Med Javascript-biblioteker sparer man tid, opnår mere funktionalitet med mindre kode og genopfinder ikke den dybe tallerken hver gang.

I dette projekt gøres brug af

- **jQuery 1.8.2**[2] Bruges primært til nem adgang til DOM-elementerne.

[1] Document Object Model
[2] http://jquery.com/

- **jQuery UI 1.9.2**[3] Udvidelse til jQuery. Bruges til "folde ud-boksene" i højre side.

- **Raphaël**[4] Vektorgrafik-bibliotek, der bruges til at tegne grafen.

4.1.2 Validering

Klienten udvikles i HTML5/CSS3 og overholder disse standarder[5]. Det sikrer platformuafhængighed og understøttelse i alle browsere, der understøtter HTML5, hvilke er alle de mest udbredte browsere, bortset fra Internet Explorer.

4.2 Server

Serveren skrives i C#.NET 4.0.
Det eksterne bibliotek Alchemy[6] bliver brugt til at lave en websocket-server.

4.2.1 Datalag

Der er store krav til graf-dataen - hvilke elementer har vi brug for at vide noget om, hvordan er id, navn, nøgler m.m. på knuderne defineret, hvilke kanter er der og går de til nogle knuder, der eksisterer etc. På grund af dette er XML valgt til datalag i systemet. Andre datakilder kunne sagtens være valgt, men XML er simpelt og nemt at skrive "i hånden". Da der er skrappe krav til grafen, er der brug for en god XSD[7]-fil, der definerer indholdet af XML-filen. XSD-filer indeholder information om hvilke elementer der er og i hvilken rækkefølge de kommer, regulære udtryk der definerer værdier, sørger for at definere referencer (her at der ikke kan være en kant til en knude der ikke er defineret).
XML-filen med grafen, systemet skal behandle, kan tjekkes op mod XSD-filen, og hvis denne består valideringen, er det en gyldig graf, der kan vises i systemet.

[3] http://jqueryui.com/
[4] http://raphaeljs.com/
[5] Valideret af W3C, http://validator.w3.org/
[6] http://alchemywebsockets.net
[7] **X**ML **S**chema **D**efinition

4.3 Kommunikation imellem server og klient

Når der skal kigges på kommunikation imellem server og klient, er der to oplagte dataformater - XML og JSON. De kan begge fint bruges til dette system.

XML-filer og -elementer har navne, attributter, åbne- og lukke-tags, XSD'er m.m. Alt dette har JSON ikke, og er derfor meget simplere, til denne simple dataoverførsel, hvor klienten ikke står for noget logik, da den blot ukritisk skal vise det den får sendt fra serveren.
JSON er lettere at læse, hurtigere at parse og fylder mindre. Det har ikke den store indflydelse på dette system, men ved større projekter kunne det have noget at sige.

JSON er rigtig godt egnet til webapplikationer - specielt med Javascript. JSON kan mappes direkte til Javascript-objekter. Derfor er JSON valgt som kommunikationsform imellem serveren og klienten.

For at kommunikere imellem server og klient er WebSockets valgt. Dette sikrer, at serveren kan sende data til klienten asynkront - data kan pushes. Klienten behøver altså ikke have spurgt efter data fra serveren først. På denne måde kan serveren sende handlinger, der skal vises på grafen, til klienten, når disse handlinger sker. Alternativt skulle klienten hele tiden have forespurgt serveren efter nye handlinger, selvom der ikke var nogen. Nu sker kommunikation kun, når der er noget at sende. Internet Explorer understøtter ikke Websockets, hvorfor dette system ikke virker i Internet Explorer.

4.3.1 Logging

Logging bliver udført ved at udskrive handlinger, der udføres på systemet til konsollen.

Chapter 5

Detaildesign og implementering

I det følgende gennemgås implementering og programdesign af systemet. I dette afsnit gennemgås de enkelte klasser, samt nogle udvalgte metoder.

Systemet udvikles med et Model-view-controller-designmønster for øje. Dette hjælper til at skabe svag kobling[1] og stærk sammenhæng[2]. Hvis der er udviklet efter MVC-modellen, er systemet delt op i data (model) og brugergrænseflade (view), således at ændringer i brugergrænsefladen ikke vil have nogen indvirkning på hvordan dataene håndteres og omvendt. I mellem view og model ligger controlleren, som sørger for al logikken.

5.1 Udviklingsmiljø

Under udviklingen af dette system, er der brugt flere forskellige programmer.

[1] Coupling: stærkt koblede klasser medfører kode, der er svær at holde ved lige, da en ændring i en klasse ofte vil medføre fejl i en anden klasse, hvis klasserne er stærkt koblede

[2] Cohesion: opgaver, der hører til i en bestemt klasse, bør udføres i metoder i den pågældende klasse. Dette er også med til at sikre en svagere kobling

Til at udvikle serveren er primært brugt Windows 7 og Visual Studio 2010 Professional.
Til at arbejde med XML er Sublime Text[3] brugt.

Til at udvikle klienten er er både Mac OS X og Windows 7 brugt. Der er primært udviklet i Firefox og gjort brug af udvidelsen Firebug, der indeholder værktøjer der bla. kan debugge Javascript, en Javascript-konsol, mulighed for at analysere nettrafik og meget mere.
På på både Mac OS X og Windows er Sublime Text brugt som udviklingsmiljø.

Til at holde styr på kildekode og dele imellem de to udviklingsmiljøer (Windows og Mac) er SVN brugt i starten af projektet. Mod slutningen er Dropbox blevet brugt. Dropbox indeholder mulighed for revision control, rollback og hurtig deling.

5.2 Kommunikation imellem server og klient

Kommunikationen er baseret på JSON. JSON-objektet inderholder overordnet to ting - en type og en datapakke.
Typen er en integer, der kan antage en af disse værdier:

0: Ny forbindelse (ikke i brug)
1: Luk forbindelse (ikke i brug)
2: Hent graf
3: Hent DFS-handlinger
4: Hent tilfældige handlinger
5: Skriv til "konsollen"
255: Fejl

Datapakken indeholder forskelligt data, alt efter hvilken type data der bliver sendt. Alt data i pakken er JSON-formateret. Følgende kodestump viser tre eksempeler på kommunikation imellem server og klient:

```
1 //Eksempel 1:
2 {"Type":5,"Data":"p_1 is looking for node with name Girls' CF"}
3 //Eksempel 2:
4 {"Type":4,"Data":{"person_id":"p_9","from":"n_20","to":"n_2"}}
5 //Eksempel 3:
6 {
7   "Type":2,
8   "Data":{
9     "name":"Test graph",
```

[3] http://www.sublimetext.com/

```
    "last_updated_at":"2013-01-08T18:35:27.0640998+01:00",
    "last_updated_by":"Mikkel",
    "server":"PC41837",
    "nodes":[
      {
        "atNode":[],
        "description":"Node1",
        "name":"Node1",
        "x_pos":10,
        "y_pos":10,
        "access_controlled":false,
        "keys":[],
        "persons":[],
        "node_id":"n_1"
      },{
        "atNode":[],
        "description":"Node2",
        "name":"n2",
        "x_pos":20,
        "y_pos":20,
        "access_controlled":false,
        "keys":[],
        "persons":[],
        "node_id":"n_2"
      }
    ],
    "edge":[
      {
        "source":"n_1",
        "sink":"n_2"
      }
    ]
  }
}
```

Listing 5.1: Eksempler på datakommunikation

I eksempel 1 beder serveren klienten skrive "p_1 is looking for node with name Girls' CF" i konsollen.
I eksempel 2 beder serveren klienten om at flytte person "p_9" fra knuden "n_20" til knuden "n_2". I eksempel 3 ses en meget simpel graf med to knuder og en kant.

5.3 Klient

Klienten er bygget op omkring `index.html`.
Denne fil inkluderer de forskellige javascript-biblioteker og CSS-filer, der er nødvendige. `index.html` indeholder ikke rigtig noget data. Det er Javascriptsne, der bliver inkluderet i `index.html`, der skaber indholdet og indsætter det de rigtige steder i filen.

5.3.1 Tredjeparts Javascript-biblioteker

De tre tredjeparts Javascript-biblioteker, der inkluderes, inkluderes øverst i `index.html`

jQuery[4] kaldes med vha. selectoren `$`
jQuery UI[5] er en jQuery-udvidelse og kalders derfor også vha. selectoren `$`
Raphaël[6] kaldes med vha. metoden `Raphael()`

5.3.2 Graphdrawer.js

`graphdrawer.js` står for at lave `Graph`-klassen, der håndterer tegning af grafen og data om denne.

To af hovedelementerne i Graph er listerne med alle knuder og alle kanter. Disse bliver gemt i et `nodes`-objekt og i et `edges`-array vha. metoderne `addNode` og `addEdge`:

```
9  (...)
10 function Graph() {
11   this.nodes = {}; // an empty object
12   // {} is a shortcut for "new Object()"
13   this.edges = []; //Array
14   this.addNode = function(content) { // If node is already there, just ↩
         return it.
15     if(this.nodes[content.id] == undefined) {
16       this.nodes[content.id] = content
17     }
18     return this.nodes[content.id];
19   };
20   this.addEdge = function(source, target) {
21     //create an edge using source and target
22     var edge = build(this.nodes[source], this.nodes[target];);
23     //add the edge
24     this.edges.push(edge);
25   }
26 };
```

Listing 5.2: "graphdrawer.js

`addNode()` tager data om en knude som parameter. Der tjekkes om der allerede findes en knude med det givne id. Hvis der ikke gør dette, så indsættes den nye knude i listen af knuder. Knuden på id'ets plads returneres

[4] http://jquery.com
[5] http://jqueryui.com/
[6] http://raphaeljs.com/

`addEdge()` tager to knuder som parameter; en source- og en target-knude. Der bliver oprettet en `edge`-variabel, som tilføjes til listen af edges.

Det næste interessante at kigge på er `draw()`, `drawNode()` og `drawEdge()`

```
31 (...)
32 Graph.Raphael.prototype = {
33   draw: function() {
34     //Draw all nodes
35     for(i in this.graph.nodes) {
36       this.drawNode(this.graph.nodes[i]);
37     }
38     //Draw all edges
39     for(var i = 0; i < this.graph.edges.length; i++) {
40       this.drawEdge(this.graph.edges[i]);
41     }
42   },
43   drawNode: function(node) {
44     //add method to object
45     node.render = function(r, n) {
46       //Create raphael eclipse that is 30 px wide and 20 px high
47       var ellipse = r.ellipse(0, 0, 30, 20).attr({
48         //set the color based on if it's access_controlled
49         fill: node.access_controlled ? color_closed : color_open,
50         stroke: node.access_controlled ? color_closed : color_open,
51         //and set a stroke and fill
52         "stroke-width": 2, "fill-opacity": 0.6
53       });
54
55       //work on the click event, to update infobox
56       ellipse.click(function() {
57         infobox(n)
58       });
59       //put it all in a set and return it
60       return r.set().push(ellipse).push(r.text(0, 30, n.label || n.id));
61     };
62     //get the node with shapes using the method we just added
63     var shape = node.render(this.r, node).hide();
64     //Get bounding box
65     var box = shape.getBBox();
66     //Move it into place
67     shape.translate(Math.round(node.x_pos - (box.x + box.width / 2)), Math.↩
           round(node.y_pos - (box.y + box.height / 2)))
68     //show it
69     shape.show();
70     //and reference it, so we can work on it later (with the edges)
71     node.shape = shape;
72   },
73   drawEdge: function(edge) {
74     edge.connection = this.r.connection(edge.source.shape, edge.target.↩
           shape);
75   }
76 };
```

Listing 5.3: "graphdrawer.js

`draw()` løber alle knuder og kanter igennem og kører `drawNode(node)` og `drawEdge(edge)` på dem.

`drawNode(node)`: Først oprettes metoden `render(r,n)`, der tager et Raphaël-objekt og en knude som parameter. Denne metode opretter tegningen af knuden vha. Raphaël. Der oprettes en ellipse med centerkoordinaterne $(0, 0)$ og en horisontal radius på 30px og en vertikal radius på 20px. Til ellipsen tilknyttes der en fyldfarve("fill") og en kantfarve ("stroke"). Disse bestemmet ud fra shorthand-udstykket
`node.access_controlled ? color_closed : color_open`. Dette er kort måde at skrive

```
1 if(node.access_controlled == true) {
2   return color_closed;
3 } else {
4   return color_open;
5 }
```

Derefter tilføjes et klik-event til ellipsen, der kalder metoden `infobox(n)`, som tager en knude som parameter.
Til sidst bliver ellipsen og en tekst samlet i et Raphaël-sæt og gemt i værdien `shape`, som returneres.

Efter `render`-metoden er defineret, bruges denne til at generere tegningen af knuden. Derefter hentes dennes afgrænsning ud vha Raphaëls `getBBox()`.
Denne bruges til at flytte tegningen på lærredet, så den er på den rigtige position vha. Raphaél-metoden `translate()`. Derefter vises den og gemmes i knuden, så vi kan tilgå den senere.

`drawEdge(edge)` tager en edge som parameter og danner tegningen af denne via `connection()`-metoden.

`connection(obj1,obj2)`(se A.1). Denne tager en tegning af en start-knude og en tegning af en slut-knude som parametre. Denne metode står for at tegne kurven og pilene imellem disse knuder. Først gennmløbes knuderne for at finde mulige start- og slutpunkter for kurven. Derefter gennemløbes alle start- og slutpunkterne, for at finde dem med kortest afstand imellem. Derefter bygges der en bézier-kurve[7] vha. en SVG-path[8]. Derefter udregnes pilene og disse tilføjes til SVG-path'en.
Hele SVG-path'en tilføjes til edgen, sammen med farve og fylde-attributter. Til sidst tegnes den nu udregnede kurve.

[7] http://en.wikipedia.org/wiki/Bézier_curve
[8] http://www.w3.org/TR/SVG/paths.html

5.3.3 realgraph.js

`realgraph.js` står for brugerinteraktionen. Det er denne, der henter grafen og viser relevant data for brugeren.

```
4  (...)
5  //Set up the accordion
6  $(function() {
7    $("#accordion").accordion({
8      collapsible: true,
9      active: false,
10     autoHeight: true,
11     clearStyle: true
12   });
13 });
14 //add a new remove method to Arrays, so we can easily remove things
15 Array.prototype.remove = function(value) {
16   //To avoid mess up if the item we're trying to remove doesn't exist.
17   if (this.indexOf(value)!==-1) {
18     this.splice(this.indexOf(value), 1);
19     return true;
20   } else {
21    return false;
22   };
23 }
```

Listing 5.4: "realgraph.js

Først sættes `<div>`-tagget med id'et `accordion` til at skulle kunne klappes sammen. Dette gøres vha. jQuery UIs `accordion()`-funktion[9].
Javascript har ikke en funktionalitet til at fjerne pladser i et array ud fra værdier. Denne funktionalitet tilføjes alle array-objekter vha prototyping. Dette gør, at alle array-objekter nu har en remove-funktion, også selvom objektet blev oprettet før prototype-deklareringen.

```
42 function connect(address) {
43   (...)
44   //If WebSocket isn't supported, MozWebSocket is probably
45   wsImpl = window.WebSocket || window.MozWebSocket;
46   connectionString = "ws://" + address + ":8181/";
47   //Connect!
48   window.ws = new wsImpl(connectionString);
49   siteLog("connecting to server ws://" + val + ":8181/..<br/>");
50   ws.onmessage = function(evt) {
51     //When we recieve something, we parese it and move on
52     var response_raw = JSON.parse(evt.data);
53     parse_response(response_raw);
54   };
55   //We open for business
56   ws.onopen = function() {
```

[9] http://jqueryui.com/accordion/

```
57      siteLog(".. connection open<br/>");
58    };
59    //Connection is closed. Clean up!
60    ws.onclose = function() {
61      siteLog(".. connection closed<br/>");
62      cleanup();
63    };
64    connected = true;
65    $("#serverbutton").prop("value", "Disconnect");
66  }
```

`connect()`-metoden er ansvarlig for at sætte kommunikationen mellem serveren og klienten op. Da WebSockets ikke er en færdig standard, har Mozilla ikke implemteret dette endnu under navnet "WebSockets", men kalder det istedet "MozWebSocket". På linje 45 gemmes `window.WebSocket`, hvis denne er tilgængelig, i variablen `wsImpl`. Er den ikke dette gemmes `window.MozWebSocket` istedet.
Derefter tilknyttes funktioner til de forskellige events, som javascripts WebSocket indeholder.
Funktionen `ws.onmessage` bliver brugt klart mest. Det er denne der bliver kørt hver gang klienten modtager data fra serveren. Her tages dataet fra serveren og sendes videre til `parse_response()`.

`draw_graph_json(graphjson)` (se A.2) bruges til at indlæse grafen og tegne denne. Som parameter tager den en JSON-kodet graf. Et eksempel på en graf ses i listing 5.1.
Til at starte med bliver der oprettet et nyt `graph`-objekt (fra `graphdrawer.js`) og grafens titel m.m. vist for brugeren. Derefter gennemløbes alle knuderne i et `for`-loop. Til hver knude bestemmes om der er adgangskontrol på knuden. Hvis der er dette, så gennemløbes rettighederne for at gemme alle personer og nøgler, der har adgang til knuden.
Knuden tilføjes til grafen vha. `addNode(content)`, der tager information om knuden som parameter. Til sidst tilføjes alle kanter til grafen vha. `addEdge(source, target)` og grafen tegnes.

```
183 //Show who's at the node
184 function updateNode(node_id) {
185   //create a string of persons at the node
186   var whoishere = "";
187   for(var i = g.nodes[node_id].atnode.length - 1; i >= 0; i--) {
188     whoishere += g.nodes[node_id].atnode[i];
189     if(i > 0) {
190       whoishere += ", ";
191     }
192   };
193   //remove the text from the node, if it's already there
194   for(var i = renderer.graph.nodes[node_id].shape.length - 1; i > 1; i--) {
195     renderer.graph.nodes[node_id].shape[i].remove();
196   }
```

```
197   //add the new text
198   renderer.graph.nodes[node_id].shape.push(renderer.r.text(renderer.graph.↩
          nodes[node_id].shape["0"].attr("cx"), renderer.graph.nodes[node_id].↩
          shape["0"].attr("cy"), whoishere));
199 }
```

Listing 5.5: updateNode(node_id)

`enterNode(node_id, person_id)` og `leaveNode(node_id, person_id)` hhv. tilføjer og fjerner personer fra `atnode`-arrayet i et knude-objekt. Begge disse kører derefter `updateNode(node_id)`.

`updateNode(node_id)` tager et node_id som parameter. Først laves en tom streng. Derefter gennemløbes alle personerne i knudens `atnode`-array. Disse tilføjes til strengen.
Det er ikke muligt at tilføje ekstra tekst til noget der allerede er tegnet på en knude. For at tilføje til teksten ved knuden, må denne derfor fjernes helt. Dette sker i linje 194-196 ved at gennemløbe alle de ting der udgør tegningen af knuden og fjerne dem alle - undtagen den første, som er selve knuden. Til sidste tilføjes strengen med personer til knuden. Knuden er hermed opdateret.

```
243 //Show who's at the node
244 var logqueue = [];
245 //How many things do we want in our concole log?
246 var loglines = 10;
247
248 function siteLog(logmessage, clearlog) {
249  //writes to the console log
250  //Should the log be cleared?
251  if(clearlog == true) {
252   logqueue = [];
253  } //if there is room in the log, add the message
254  if(logqueue.length < loglines) {
255   logqueue.push(logmessage);
256   //if there isn't room, remove the first one and insert at the end (it's a↩
          queue)
257  } else if(logqueue.length >= loglines) {
258   logqueue.shift();
259   logqueue.push(logmessage);
260  }
261  $("#incomming").html("");
262  //Print the log
263  for(i = 0; i < logqueue.length; i++) {
264   $("#incomming").append(logqueue[i]);
265  }
266 }
```

Listing 5.6: siteLog(logmessage, clearlog)

`siteLog(logmessage, clearlog)` er den sidste interessante ting at se på i `realgraph.js`. Denne bruges til at skrive beskeder til brugeren i "konsollen" i højre side af klienten. Den tager to parameter. Den første er beskeden, der skal skrives. Anden parameter er om alle tidligere beskeder skal slettes. Hvis

de skal dette, så sættes `logqueue = []`, altså til at være et nyt tomt array. Derefter tjekkes der om der er ledige pladser til nye beskeder ved at sammenligne `logqueue.length` med loglines. Der må højest være `loglines = 10` beskeder vist til brugeren af gangen. Hvis der er ledige pladser, så tilføjes beskeden til `logqueue`. Er der ikke ledige pladser, så shiftes `logqueue`, så det første element forsvinder og der bliver en ledig plads bagerst, hvor beskeden indsættes.
Til sidst fjernes fjernes alt indholdet i konsollen og den nye liste af beskeder skrives deri.

5.4 Server

Serveren er skrevet i C#.NET 4.0.
Serveren består ikke af så mange klasser. Der er en starter-klasse, `program`, en hovedklasse, `server`, et par hjælperklasser og nogle model-klasser, `EntityClass`, `graph` (og underklasser) og `graphPerson` (og underklasser)

5.4.1 program

Denne klasse indeholder kun `main()`-metoden:

```
1 private static void Main() {
2   Server sv = new Server();
3   while (Console.ReadLine() != "exit") {
4     Console.WriteLine("Unknown command. Exit by typing \"exit\"");
5   }
6   sv.aServer.Stop();
7 }
```

Der oprettes en instans af `server`. Derefter ventes der på kommandoen `"exit"`. Når denne kommando kommer fra brugeren, afsluttes programmet.

5.4.2 Model-klasser

Model-klasserne er ansvarlige for alt der har med data at gøre. De er ansvarlige for at indlæse data og håndtere dataobjekterne.

5.4.2.1 EntityClass.cs

`EntityClass` er en overordnet klasse, der står for at indlæse data fra XML-filerne. Den fungerer som baseklasse for både `graph` og `graphPerson`.
Klassen tager en Generic Type T med som input. Dette bestemmer hvilken slags klasse der arbejdes med. Dette gør at den kan bruges til at indlæse både `graph`- og `graphPerson`-objekter.
`EntityClass` indeholder to metoder:

`LoadFromFile(string filename)`: I en try-catch-blok indlæses filen først vha. en `FileStream`, sidenhen en `StreamReader`, og sendes til `Deserialize(string xml)`. Hvis indlæsningen fejler skrives dette i serverens konsol. Til sidst ryddes der op ved at `FileStream` og `StreamReader` disposes.

`Deserialize(string xml)`: I denne metode oprettes der en XmlSerializer, der tager modtager typen T, som defineres når `EntityBase`-klassen oprettes. XmlSerializerens `Deserialize()`-metode bruges og returværdien derfra bliver typecastet til typen T, der angiver hvilken slags klasse der arbejdes med.

5.4.2.2 graph.cs og graphPerson.cs

Disse to klasser nedarver begge fra `EntityClass`. De er ansvarlige for at indeholde data fra hhv. XML-filen med grafen og XML-filen med personer tilknyttet grafen. De indeholder gettere og settere for alle elementer og attributter i XML-filerne, f.eks.

```
16 public string name
17 {
18   get
19   {
20     return this.nameField;
21   }
22   set
23   {
24     this.nameField = value;
25   }
26 }
```

Listing 5.7: eksempel fra graph.cs

5.4.3 server.cs

`Server`-klassen indeholder al logikken for server-delen. Den gør kraftigt brug af tredjeparts-websocketserveren Alchemy.

5.4.3.1 Server()

```
19 public Server()
20 {
21  aServer = new WebSocketServer(8181, IPAddress.Any)
22    {
23      OnReceive = OnReceive,
24      OnSend = OnSend,
25      OnConnected = OnConnect,
26      OnDisconnect = OnDisconnect,
27      TimeOut = new TimeSpan(1, 0, 0)
28    };
29  aServer.Start();
30
31   //Server is started!
32  Console.WriteLine("Server is started at " + System.Environment.MachineName↩
        + ":8181");
33  Console.WriteLine("Exit by typing \"exit\" or close the program");
34 }
```

Listing 5.8: Server() i server.cs

I constructoren oprettes der en ny instans af Alchemys `WebSocketServer`. Som parameter tager den porten, som den skal lytte på, samt hvor den accepterer forbindelser fra. I dette tilfælde accepteres der indkomne forbindelser på fra alle adresser på porten 8181.
Derefter defineres hvilke metoder der skal køres ved de forskellige events `WebSocketServer` understøtter. Til sidst skrives der i konsollen, at serveren er startet.

5.4.3.2 OnReceive(UserContext context)

```
56 public void OnReceive(UserContext context)
57 {
58  Console.WriteLine("Received Data From :" + context.ClientAddress);
59  try
60  {
61   var json = context.DataFrame.ToString();
62   dynamic obj = JsonConvert.DeserializeObject(json);
63
64   switch ((int)obj.Type)
65   {
66    case (int)CommandType.getGraph:
67     this.getGraph(context);
```

```
68      break;
69     case (int)CommandType.getDFSActions:
70      speed = (int)obj.Speed;
71      this.getActions(context);
72      break;
73     case (int)CommandType.getRandomActions:
74      speed = (int)obj.Speed;
75      this.getRandomActions(context);
76      break;
77    }
78   }
79   catch (Exception e) // Bad JSON! For shame.
80   {
81    sendMessage(ResponseType.Error, new { e.Message }, context);
82   }
83  }
```

Listing 5.9: OnReceive(UserContext context) i server.cs

Denne metode køres hver gang der modtages data fra klienten. Som parameter tages en `UserContext context`. Denne indeholder information om hvor data kommer fra, hvilken type data, en datablok og meget mere. Det data der modtages fra klienten er JSON-formateret. Dette konverteres til et dynamisk objekt vha. `JsonConvert.DeserializeObject()`. Dette objekt indeholder nu en en `"type"`, der angiver hvilken handling der ønskes udført, samt en `"speed"`, der angiver hvor hurtigt handlingen skal udføres.
Ud fra `"type"` laves et swich-case-statement, der sørger for at kalde den ønskede metode.
Alt dette er pakket ind i en try-catch, der sørger for at fange elle exceptions, der måtte blive kastet underliggende metoder og boble op og sende en besked til klienten om at der opstod en fejl.

5.4.3.3 getActions(UserContext context)

Koden til denne metode er at finde som appendiks A.3. Denne metode gennemløber hele grafen vha. en dybde først-søgning, DFS, for hver person der findes. Først indlæses listen med personer vha. `graphPersons` og `EntityClass`, som beskrevet ovenfor. For hver person i listen gøres følgende:
Først oprettes der en stak, `s`. Der bruges en stak, da det er en DFS. Havde der været brugt en kø, var der blevet lavet en bredde-først-søgning. Derefter oprettes en liste af knuder, som allerede er besøgt. Dette gøres for at undgå at havne i en uendelig løkke, hvor de samme knuder bliver besøgt igen og igen.
Søgningen starter altid i den første knude. Der tjekkes om det er muligt at gå ind i den. Hvis det er, så tilføjes denne til stakken af knuder, der skal besøges.

Nu gennemløbes listen af knuder, så længe denne indeholder en eller flere knuder.

Den aktive knude hentes ud af stakken ved at "poppe" denne. Der tjekkes om denne allerede er blevet besøgt. Hvis den er det, så tages den næste knude. Er knuden ikke besøgt, så køres `Thread.Sleep(speed);`. Dette pauser eksekveringen i et givent stykke tid. Dette er for at klienten ikke modtager handlinger for hurtigt, så de ikke er til at se.
Information om hvor personen er, bliver gemt og personen tilføjes til den nye knude vha. følgende kald:

```
138 addPersonToGraph(gra.nodes.Find(t => t.node_id == n.to.node_id), person);
```

Der bruges LINQ (`.Find()`) og et lambda predicate-udtryk (`t => t.node_id == n.to.node_id`) til at hente knuden ud, der har det id, som knuden personen er på vej ind i.
Derefter skrives til serverens konsol, at personen går fra den ene knude til den anden. Dette sendes også til klienten, så denne kan vise dette.
Knuden tilføjes til listen over knuder, som er besøgt, så vi ikke besøger denne igen. Derefter kigges, vha. `listOfEdgesAllowed(graphNode node, graphPersonsPerson person)`, på alle knuder, det er muligt at komme til fra knuden, med de tilladelser personen har. Hvis de ikke tidligere er besøgt tilføjes denne til listen af knuder, der skal besøges.
Derefter starter processen forfra, ind til alle knuder er besøgt. Når de er det, så gøres det hele igen med den næste person, ind til der ikke er flere.

5.4.3.4 getRandomActions(UserContext context)

Koden til denne metode er at finde som appendiks A.4. Denne metode udvælger en tilfældig knude for hver person, og ser om denne kan nå frem til denne.
Denne metode starter på samme måde som `getActions()` med at indlæse personer, løbe dem igennem og starte med at tjekke om personen kan tilgå den første knude. Hvis personen ikke kan det, breakes der ud af den igangværende `foreach`-løkke.
Derefter udvælges der en tilfældig knude, `targetNode` via `Random().Next(int minValue, int maxValue)`, hvor min er 1 og max er antallet af knuder. Denne knude sendes også til klienten, så det er muligt at se hvilken knude der søges efter.
Der startes en tæller, `tryCounter`, som bruges til at tælle op hvor mange gange personen går til en ny knude, så der ikke ledes for evigt. Så længe den nuværende knude ikke er vores `targetNode` gøres følgende:
Først køres `Thread.Sleep(speed)` som i `getActions()`.
Via `listOfEdgesAllowed(currentNode, person)` hentes en liste af alle de kanter der kan bruges til at komme til en knude, som personen har adgang til.

Hvis der ikke er nogle kanter, der kan bruges, skrives dette til klienten og der søges med næste person. Hvis der er kanter der kan bruges, vælges en af disse tilfældigt. Personen flytter til en ny knude via kanten og dette fortælles til klienten.

Hvis den nye knude er `targetNode`, så er vi nået frem til målet. Dette sendes til klienten og der startes forfra med en ny person. Er det ikke `targetNode`, så hæver vi `tryCounter` med en. Er den 50, så opgives der at finde `targetNode` og dette sendes til klienten og der startes forfra med den næste person.

5.4.4 listOfEdgesAllowed(graphNode node, GraphPersonsPerson person)

```
257 //Get list of all edges from a node, that ends in a node, that a person has↩
        access to.
258 private List<graphEdge> listOfEdgesAllowed(graphNode node, ↩
        GraphPersonsPerson person)
259 {
260  List<graphEdge> allowedEdges = new List<graphEdge>();
261  //Get all edges from the node
262  List<graphEdge> edges = gra.edge.FindAll(e => e.source == node.node_id);
263  //Loop through them. If the edge ends somewhere good, add it to the list ↩
        of allowed edges.
264  foreach (graphEdge edge in edges)
265  {
266   if (canEnterNode(gra.nodes.Find(n => n.node_id == edge.sink), person))
267   {
268    allowedEdges.Add(edge);
269   }
270  }
271  //And return the list
272  return allowedEdges;
273 }
```

Listing 5.10: listOfEdgesAllowed(graphNode node, GraphPersonsPerson person)

Denne metode bliver brugt af `getActions()` og `getRandomActions()` til at finde kanter fra en given knude, der fører til knuder, som en person har adgang til.
Som parameter tages en knude og en person. Der oprettes en tom liste af kanter, `allowedEdges` og en liste af alle udgående kanter fra knuden, `edges`. For hver kant i `edges` kigges der på om personen har adgang til knuden, som kanten fører til. Har personen adgang så tilføjes denne kant til listen `allowedEdges`. Til sidst returneres `allowedEdges`.

Chapter 6

Videreudvikling

Det udviklede system er blot en prototype. Derfor kan systemet videreudvikles på mange punkter.

Der kunne udvikles et redigeringsmodul, så det var muligt at oprette grafer via et grafisk interface i stedet for via XML-filer. Dette kunne laves ved at videreudvikle klienten, så man kunne tilføje knuder og kanter og trække disse rundt. Man vil både kunne generere XML-filen, der bruges som datagrundlag, på klient-siden og på server-siden. Det ville være forholdsvis simpelt at sende en JSON-kodet graf fra klienten til serveren, som ville mappe denne til et `graph`-objekt. Det ville så være muligt at udvikle en serialize-funktion, der minder meget om den allerede eksisterende deserialize-funktion, til at generere en XML-fil.

Der kunne udvikles andre klienter til systemet. Da klient og server er to helt separate moduler, er det nemt at udskifte klienten med en anden klient, udviklet i f.eks. Java eller C#. Der kunne således også udvikles en smartphone-app, som kunne virke som klient til systemet. Dog kan nuværende web-baserede klient også bruges på smartphones, så længe disse understøtter HTML5.

I serveren bruges `thread.sleep()` til at forsinke gennemløbningen af grafen. I den nuværende form er det hoved-tråden for serveren, der bliver suspenderet. Dermed bliver systemet forsinket for alle klienter, ikke kun for den der er ved

at modtage handlinger. Man kunne med fordel lave serveren multi-trådet, så en ny forbindelse fra en klient spawnede en ny tråd. På denne måde ville én klient ikke være påvirket af hvad en anden klient gjorde.

Til at kommunikere imellem server og klient bruges WebSockets. Dette er ikke understøttet af Internet Explorer (endnu). For at gøre det muligt at bruge Internet Explorer, kunne der implementeres andre måder at kommunikere på. Evt. kunne Socket.IO[1] bruges til at formidle kommunikationen. Socket.IO gør brug af forskellige måder at skabe websocket-lignende kommunikation på. Hvis websockets er understøttet af server og klient, så bruges dette, ellers bruges f.eks. Afobe Flash Sockets, AJAX long polling eller persistent iframe.

Der kunne implementeres bedre logging-funktioner, der skrev log over udførte handlinger til en fil, så man senere kunne gennemgå denne. Dette ville være brugbart til senere analyse af hvordan en person kunne nå ind til et sted, som man måske ikke havde tænkt han skulle have adgang til.

[1] `\protect\T1\textbraceright`

Chapter 7

Test

7.1 Frontend test

Der fndes nogle forskellige online-validatorer, og web-klienten validderer henholdsvis HTML5 og CSS 3 igennem W3Cs http://validator.w3.org/ og http://jigsaw.w3.org/css-validator/. Disse betragtes som de offcielle (kommende) standarder og validatorer hertil. Eftersom disse validatorer kun påviser at standarden er fulgt, skal der testes andenvejs om systemet bliver vist og rent faktisk fungerer ens i de forskellige browsere. Denne test er nødvendigvis ført manuelt. Testede browsere er:

- Firefox 16+
- Google Chrome
- Safari
- Safari for iOS
- Android 4.2 standard browser

Internet Explorer er ikke understøttet. Når Internet Explorer understøtter WebSockets, så vil systemet højest sandsynlig også virker der.

jQuery overholder dog ikke helt CSS3-standarden. Dette er dog ikke af betydning for dette projekt.

7.2 Unit testing

Unittesting foregår ved at kigge på den værdi, en metode returnerer. Denne sammenlignes med en forventet værdi. Er disse ens, så er testen bestået. Et simpelt eksempel på en unittest er

```
1 public int add(int a, int b) {
2  return a + b;
3 }
4 public void testAdd(){
5  assert(add(3,3) == 6);
6 }
```

Her testes metoden `add(int a, int b)` for at se om den rent faktisk lægger tal sammen.

Ideen med at skrive unittests er, at hvis alle testene forløber planmæssigt, så er din kode fri for bugs - hvis altså alt kode bliver dækket og testene er skrevet ordentligt.

Der er udviklet unittests til serveren og klienten. Samtlige tests er gennemført med succes. Der er mange metoder i dette system, der ikke er nemme at unitteste - f.eks. er mange af serverens metoder svære at unitteste, da de venter på svar fra en websocket-klient, før de kan udføre en handling. Der kunne være udviklet en websocket-klient i C#, som kunne bruges til unittesting, men disse tests er udført manuelt.
Ligeledes er der mange metoder der ikke returnerer en værdi.

7.2.1 Klient

Til at unitteste klienten er der brugt QUnit[1]. QUnit er udviklet af holdet bag jQuery og er et Javascript unit testing framework. I første omgang blev det udviklet til at unitteste jQuery, men det kan bruges til at teste alt slags Javascript-kode.

[1] http://qunitjs.com/

Der er udviket en række QUnit tests til at teste `graphdrawer.js` og `realgraph.js`. Resultatet af tests for `graphdrawer.js` ses i tabel 7.1 på side 42 og for `realgraph.js` i tabel 7.2 på side 42. Outputtet fra QUnit-kørslen ses i figur 7.1. Disse er også at finde i ved klienten i undermappen /tests/.

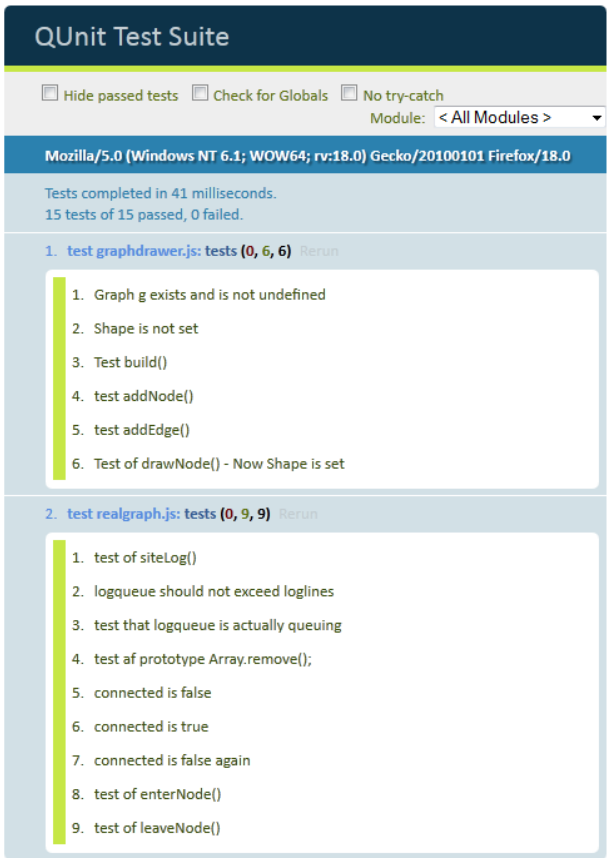


Figure 7.1: Screenshot af klient-unittests.

7.2.2 Server

Til at lave unit tests af serveren er Visual Studio Unit Testing Framework brugt. Dette er Microsofts samling af værktøjer til at lave unit tests i Visual Studio 2005 og fremefter.
Til serveren er der udviklet række tests. Disse findes i tabel 7.3. Kørslen fra Visual Studio ses i figur 7.2

Testnavn	Beskrivelse	Resultat
Graph g exists and is not undefined	Tjek om `g` er sat	✓
Shape is not set	Efter oprettelse af en knude skal den ikke have nogen shape	✓
Test build()	Tjek om der bliver oprettet den rigtige struktur ved brug af `build()`	✓
Test addNode()	Tjek at knuder bliver oprettet rigtigt	✓
Test addEdge()	Tjek at kanter bliver oprettet rigtigt	✓
Test drawNode(), now shape is set	Efter `drawNode()` bliver kørt skal der være en `shape` på knuden	✓

Table 7.1: Tests af graphdrawer.js

Testnavn	Beskrivelse	Resultat
Test of `siteLog()`	Tjek at `siteLog()` skriver til loggen	✓
`logqueue` should not exceed `loglines`	Der skal ikke kunne være flere beskeder end loglines	✓
Test that logqueue is queueing	Tjek at logqueue virker som en kø	✓
Test Array.remove()	Tjek at Array.remove() fjerner entries	✓
`connected` is false	`connected` skal være false inden der bliver forbundet til server	✓
`connected` is true	`connected` skal være false efter der bliver forbundet til server	✓
`connected` is false again	`connected` skal være false igen efter der har været forbundet til server	✓
Test of `enterNode()`	Tjek `enterNode()` virker	✓
Test of `leaveNode()`	Tjek `leaveNode()` virker	✓

Table 7.2: Tests af realgraph.js

Testnavn	Beskrivelse	Resultat
canEnterNodeTest	Tjek om canEnterNode() giver korrekt svar, given en knude der er adgangskontrolleret og en person der har adgang	✓
canNotEnterNodeTest	Tjek om canEnterNode() giver korrekt svar, given en knude der er adgangskontrolleret og en person der ikke har adgang	✓
addPersonToGraphTest	Tjek om addPersonToGraph() tilføjer person til knuden	✓
whereIsPersonTest	Tjek om whereIsPerson() returnerer den rigtige knude, når der skal findes en person	✓

Table 7.3: Tests af Server.cs

Result	Test Name	Project
Passed	canEnterNodeTest	RealTests
Passed	addPersonToGraphTest	RealTests
Passed	canNotEnterNodeTest	RealTests
Passed	whereIsPersonTest	RealTests

Figure 7.2: Screenshot af server-unittests.

7.3 Funktionel test og brugerinterface-tests

Under hele udarbejdelsen af systemet, har dettes funktionalitet løbende været testet. Dette er sket igennem direkte tests i brugergrænsefladen samt debugging/stepping, for at sikre sig, at denne opfører sig korrekt og viser data baseret på modellen/XML.

En del af disse tests kunne hjælpes på vej ved at lave automatiserede tests - ved at programmere en simulering af en user case for eksempel med udfyldning af formular og klik på links og knapper. Dette er dog ikke gjort i dette projekt, da det vil være meget besværligt at bruge disse til at teste grafens udseende.

Chapter 8

Konklusion

Det færdige system indholder prototypen på et system, der kan repræsentere et virkeligt miljø og de sikkerhedsaspekter der er i dette.

Der er udviklet en server, baseret på C#, og en klient, baseret på HTML5 og Websockets, Javascript og SVG-grafik. At klienten er implementeret i HTML5, sikrer kompatibilitet på flere platforme og styresystemer, hvilket øger brugbarheden betragteligt.

For at udveksle data imellem server og klient, bliver JSON brugt. Dette gør det nemt at udvikle andre klienter til systemet, da JSON er en åben, tekstbaseret udveksling af data.
Da klienten og serveren er helt adskilt applikationsmæssigt, er det muligt at have disse på to forskellige systemer. Dermed kan man vinde bedre performance, hvis det er meget store grafer der arbejdes med. Dog er dette system ikke synderligt datatungt, hvorfor det, i stor udstrækning, ikke er noget problem at køre server og klient på samme computer.

Til at gemme grafen og personerne, som opererer på grafen, bruges XML som datagrundlag. Disse XML-filer er dannet ud fra veldefinerede XSD-filer, der beskriver XML-filernes indhold. Alle grafer, der opfylder kravene i XSD-filen, er gyldige grafer, der kan vises i systemet.

Der er udviklet en række tests, der er med til at sikre systemets stabilitet. Derudover er systemet i stort omfang testet manuelt.

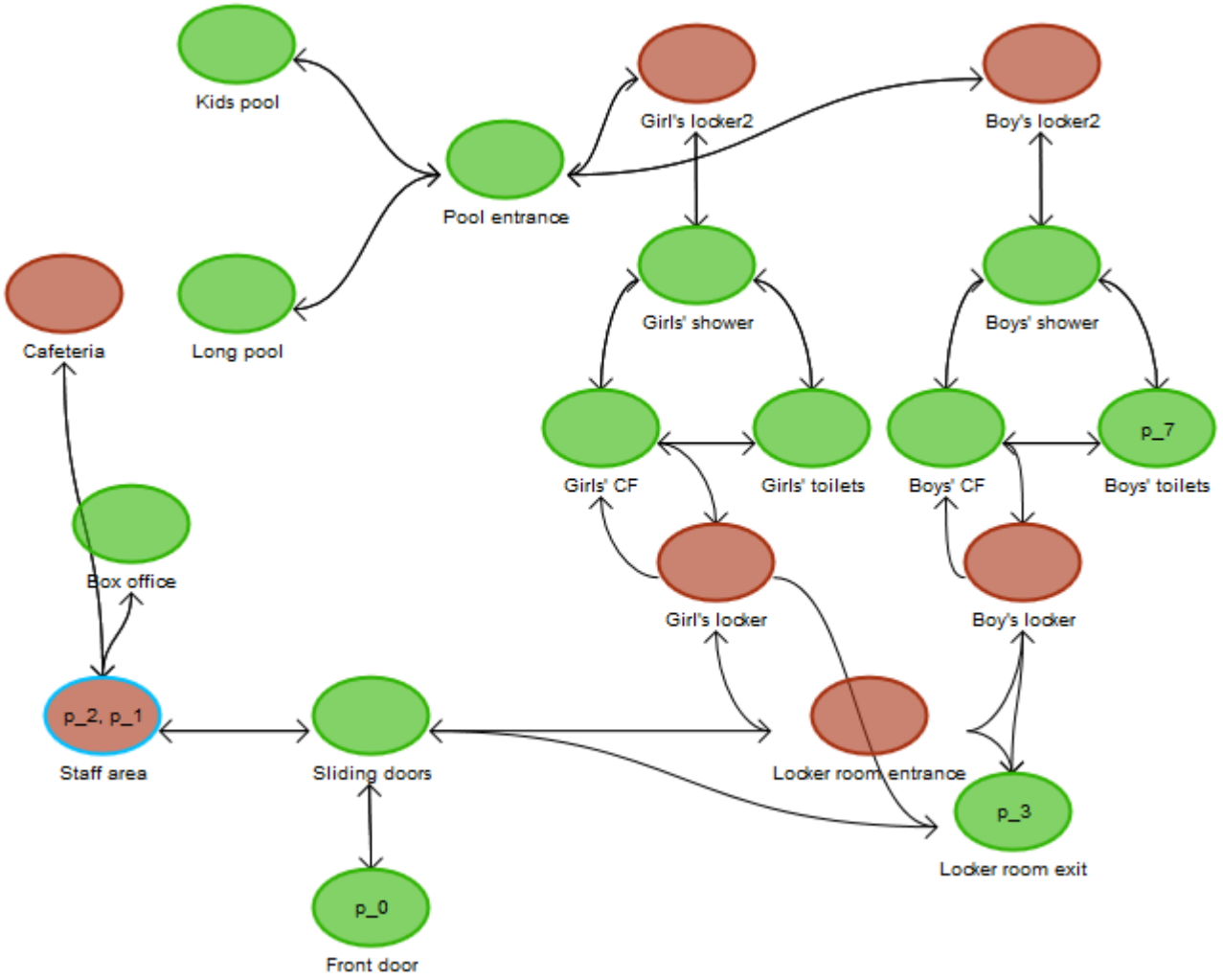
Chapter 9

Litteraturreferancer

http://en.wikipedia.org/wiki/Bézier_curve: Bézier-kurver beskrevet.
http://bit.ly/XvY7bL: SVG-paths og opbygning af disse.
http://bit.ly/9txJl: Javascript - Arrays og objekter.
http://www.w3.org/Consortium/: W3C-konsortiets hjemmeside.

Chapter 10

Brugervejledning



Brugervejledning

RealGraph

RealGraph består af to dele – en server og en klient.

Begge dele er krævet for at have et komplet system.

1. Server

1.1 Krav

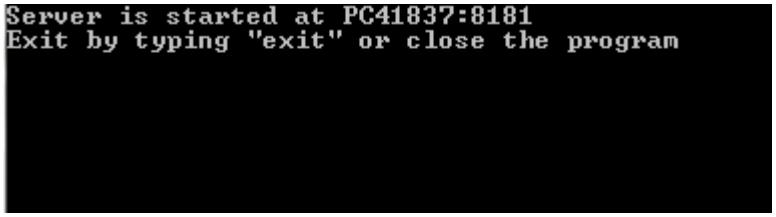
Computer med Microsoft Windows (Testet på Windows 7)

Hvis det ønskes at klienten skal kunne tilgå serveren via internettet, er en aktiv internetforbindelse med port 8181 åben påkrævet.

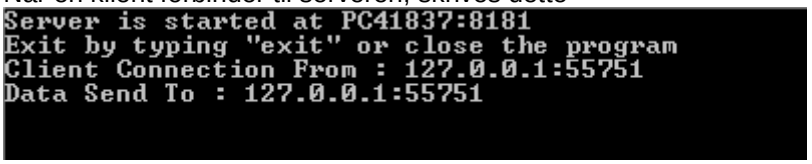
To XML-filer, der specificerer hhv. grafen og personerne i denne.
Disse skal være navngivet ”graph.xml” og ”graphpersons.xml” og ligge i samme bibliotek som ”RealGraph.exe”. Der findes to filer i forvejen, som kan bruges.
Der er to XSD-filer, ”graph.xsd” og ”graphpersons.xsd” der specificerer hvordan XML-filerne skal være formatteret.

1.2 Funtionalitet

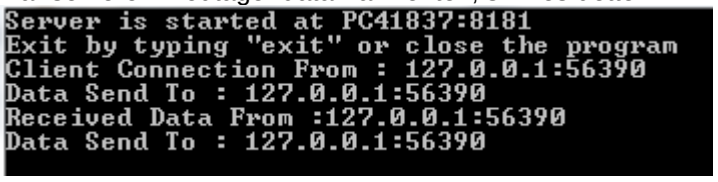
Serveren startes ved at køre filen “RealGraph.exe”



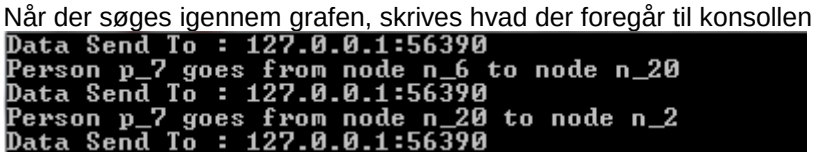
Når en klient forbinder til serveren, skrives dette



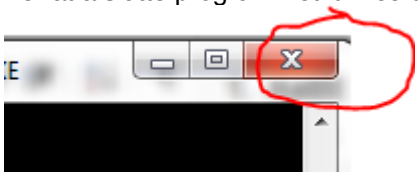
Når serveren modtager data fra klienten, skrives dette



Når der søges igennem grafen, skrives hvad der foregår til konsollen:



For at afslutte programmet lukkes dette ved at klikke på ”luk”



Eller ved at skrive ”`exit`” i konsollen

```
Data Send To : 127.0.0.1:56390
Person p_7 goes from node n_8 to node n_20
Data Send To : 127.0.0.1:56390
Data Send To : 127.0.0.1:56390
exit
```

2. Klient:

2.1 Krav

Klienten kan køre fra alle enheder, hvorpå der kører en browser, der understøtter HTML5 og WebSockets.

Dette er f.eks.

Mozilla Firefox
Google Chrome
Apple Safari

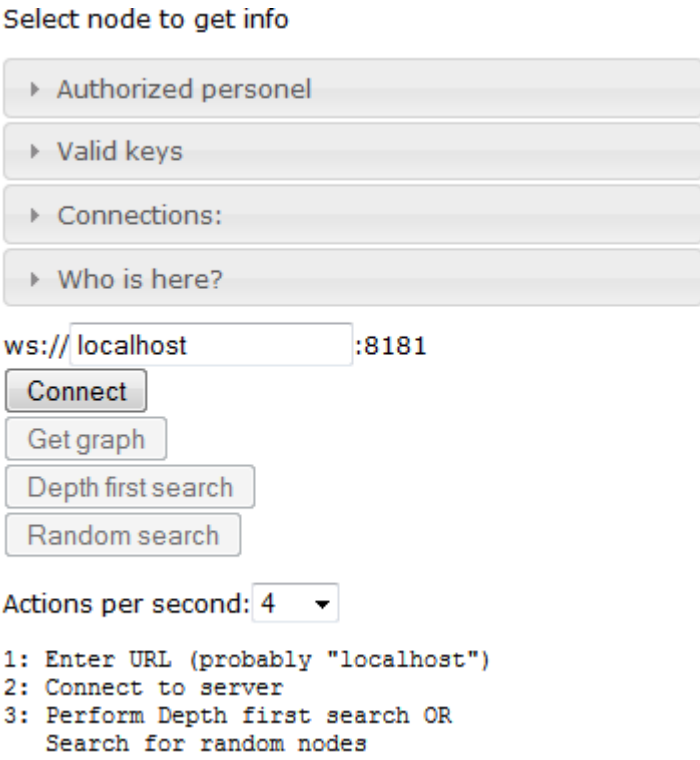
Det kan bade være på Windows, Mac, Linux, Android og iOS.

Hvis klienten kører fra en anden computer end serveren, kræves en netværksforbindelse til denne.

2.2 Funktionalitet

Klienten startes ved at åbne “index.html” i en understøttet browser.

Når klienten startes, vises dette:



Info-området i toppen kan ikke vise noget endnu.

Der indtastes adressen på den server, der ønskes at forbinde til – hvis serveren kører på samme maskine som klienten er dette ”localhost” – og trykkes ”Connect”

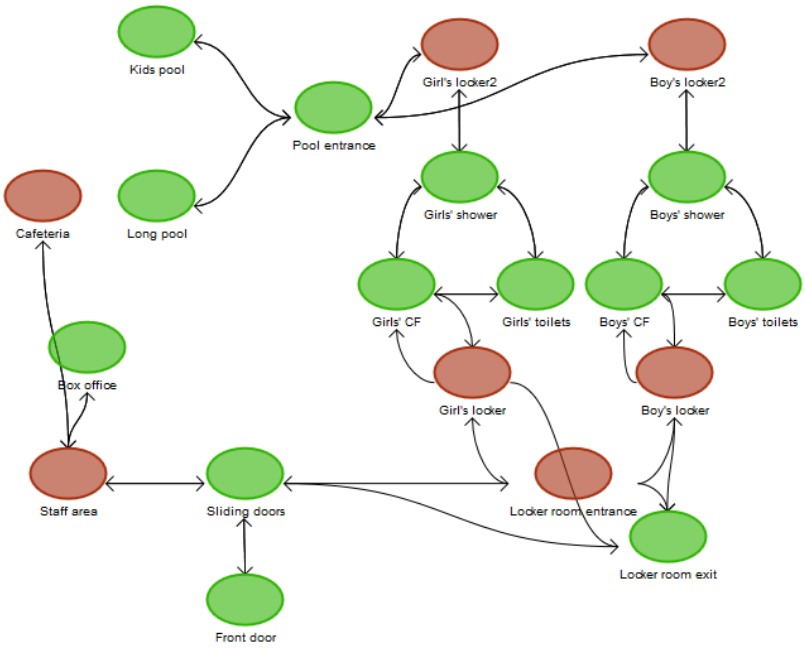
Klienten forbinder nu til serveren og dette skrives i konsollen:

```
connecting to server ws://localhost:8181/..
.. connection open
```

Grafen kan nu hentes ved tryk på ”Get Graph”.
Grafen hentes og vises:

Lyngby Svømmehal

Graph updated at 2013-01-11T10:48:23.1259556+01:00 by Mikkel Kaas Pedersen.
Graph came from server at PC41837



Select node to get info

- Authorized personel
- Valid keys
- Connections:
- Who is here?

ws:// localhost :8181

Disconnect

Get graph

Depth first search

Random search

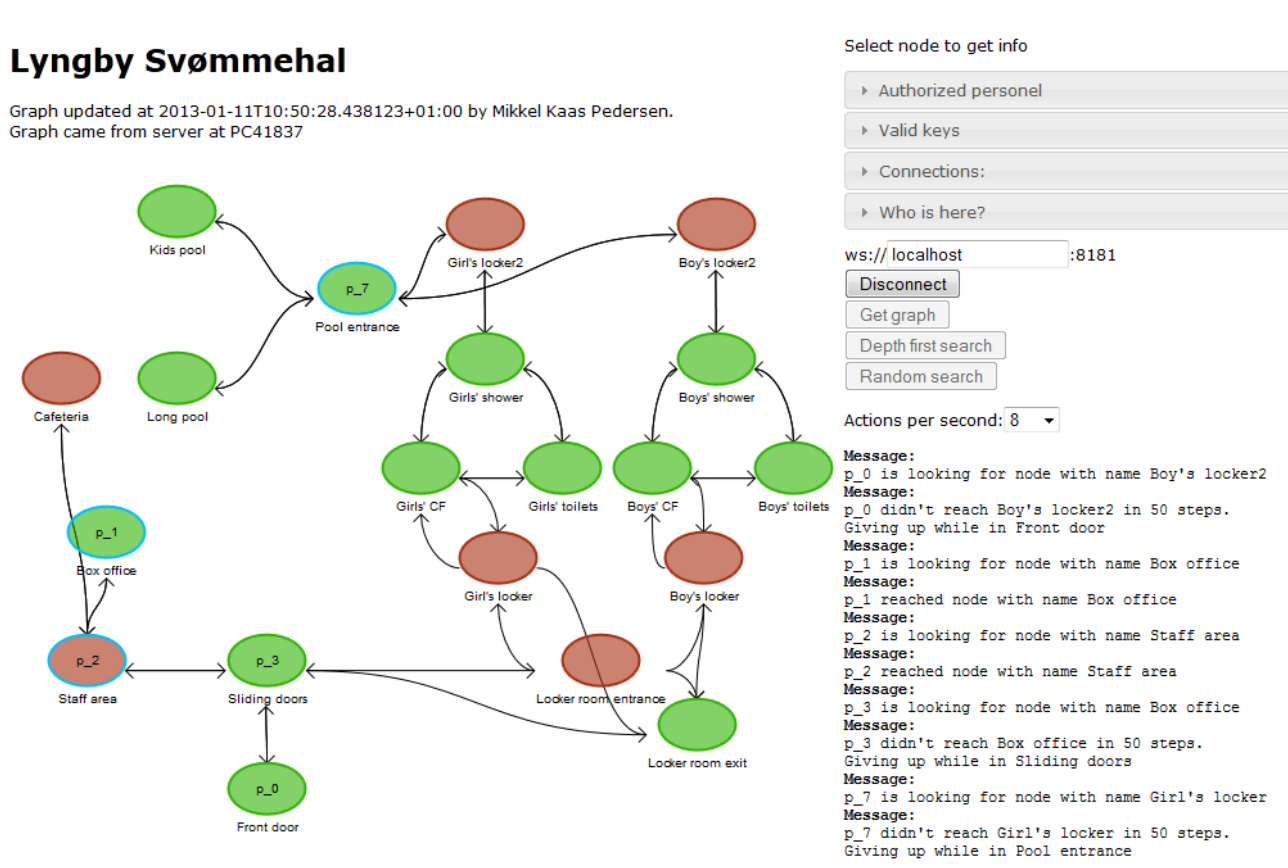
Actions per second: 4

```
.. connection lost/closed
connecting to server ws://localhost:8181/..
.. connection open
.. recieved graph
.. drawing graph
```

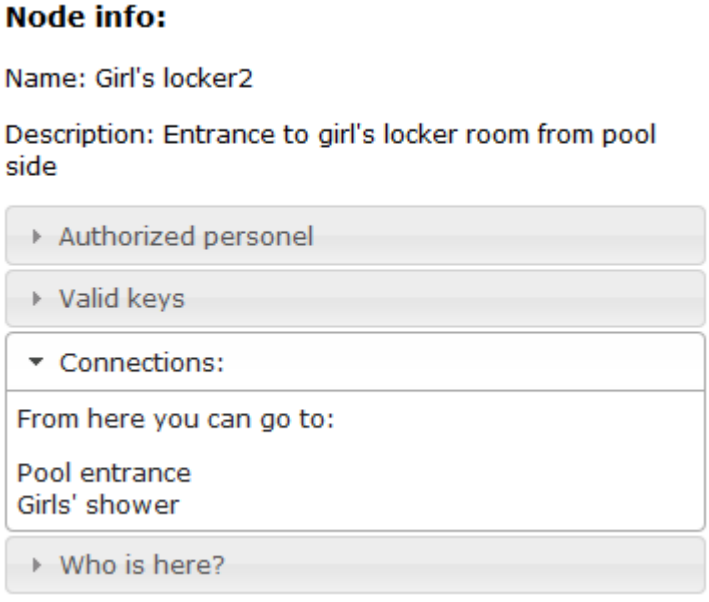
Ved tryk på ”Depth first search” gennemføres en dybde først-søgning igennem grafen.
Ved tryk på ”Random search” vælges en tilfældig knude og der søges efter denne.

Antallet af handlinger pr. sekund kan vælges i drop down-menuen.

Her ses resultatet af en ”Random search”

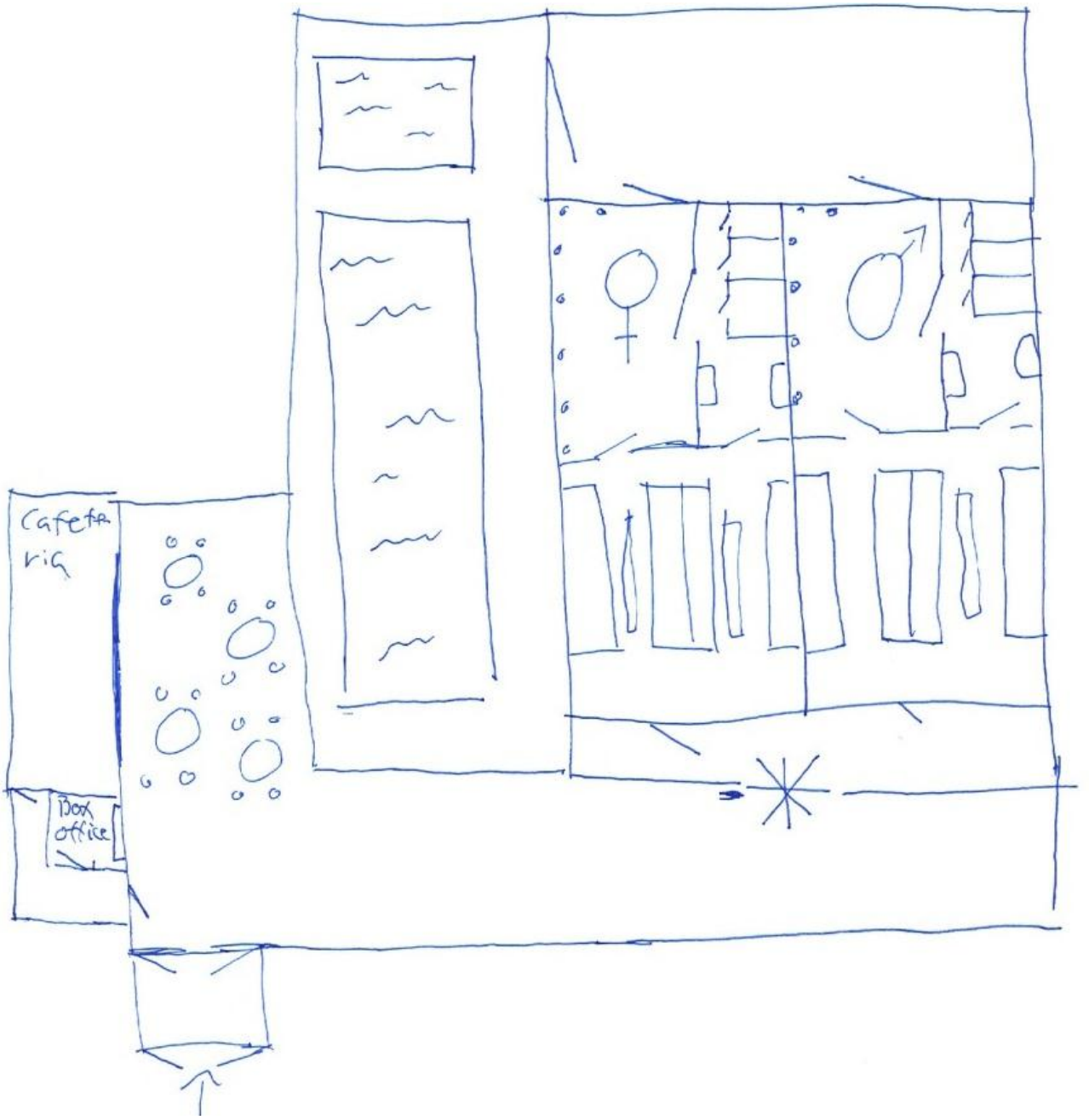


Når der trykkes på en knude, er det muligt at se data om denne knude:



Klienten afsluttes ved at lukke websiden.

Standard-grafen er lavet ud fra en plantegning af Lyngby Svømmehal:


Cafeteria
Box office

Appendix A

Appendiks

A.1 connection(obj1, obj2)

```
Raphael.fn.connection = function Connection(obj1, obj2) {
  var selfRef = this; // create and return new connection
  var edge = {
    draw: function() { // get bounding boxes of target and source
      var bb1 = obj1.getBBox();
      var bb2 = obj2.getBBox(); // coordinates for potential connection ↩
          coordinates from/to the objects
      var p = [ // NORTH 1
      {
        x: bb1.x + bb1.width / 2,
        y: bb1.y
      }, // SOUTH 1
      {
        x: bb1.x + bb1.width / 2,
        y: bb1.y + bb1.height
      }, // WEST 1
      {
        x: bb1.x,
        y: bb1.y + bb1.height / 2
      }, // EAST 1
      {
        x: bb1.x + bb1.width,
        y: bb1.y + bb1.height / 2
      }, // NORTH 2
      {
        x: bb2.x + bb2.width / 2,
        y: bb2.y
```

```
27        }, // SOUTH 2
28        {
29          x: bb2.x + bb2.width / 2,
30          y: bb2.y + bb2.height
31        }, // WEST 2
32        {
33          x: bb2.x,
34          y: bb2.y + bb2.height / 2
35        }, // EAST 2
36        {
37          x: bb2.x + bb2.width,
38          y: bb2.y + bb2.height / 2
39        }]; // distances between objects and according coordinates connection
40        var d = {},
41          dis = [];
42
43        // find out the best connection coordinates by trying all possible ↩
              ways
44        // loop the first object's connection coordinates
45        for(var i = 0; i < 4; i++) { // loop the seond object's connection ↩
              coordinates
46          for(var j = 4; j < 8; j++) {
47            var dx = Math.abs(p[i].x - p[j].x);
48            var dy = Math.abs(p[i].y - p[j].y);
49            if((i == j - 4) || (((i != 3 && j != 6) || p[i].x < p[j].x) && ((↩
                  i != 2 && j != 7) || p[i].x > p[j].x) && ((i != 0 && j != 5)↩
                   || p[i].y > p[j].y) && ((i != 1 && j != 4) || p[i].y < p[j↩
                  ].y))) {
50              dis.push(dx + dy);
51              d[dis[dis.length - 1]] = [i, j];
52            }
53          }
54        }
55        //Now we know where to put the arrows. Hooray!
56        //Lets build our bezier curve
57        //Math.min.apply(Math, dis) gets the min value of dis
58        var res = dis.length == 0 ? [0, 4] : d[Math.min.apply(Math, dis)];
59
60        var x1 = p[res[0]].x,
61          //Start X
62          y1 = p[res[0]].y,
63          //Start Y
64          x4 = p[res[1]].x,
65          //End X
66          y4 = p[res[1]].y,
67          //End Y
68          dx = Math.max(Math.abs(x1 - x4) / 2, 10),
69          dy = Math.max(Math.abs(y1 - y4) / 2, 10),
70          x2 = [x1, x1, x1 - dx, x1 + dx][res[0]],
71          y2 = [y1 - dy, y1 + dy, y1, y1][res[0]],
72          x3 = [0, 0, 0, 0, x4, x4, x4 - dx, x4 + dx][res[1]],
73          y3 = [0, 0, 0, 0, y1 + dy, y1 - dy, y4, y4][res[1]];
74
75        var path = ["M" + x1, y1, "C" + x2, y2, x3, y3, x4, y4];
76
77        //Now lets build our arrow
78        //magnitude, length of the last path vector
79        var mag = Math.sqrt((y4 - y3) * (y4 - y3) + (x4 - x3) * (x4 - x3));
80        // vector normalisation to specified length
81        var norm = function(x) {
82            //6 is the length of the legs in the arrow
83            return(-x * 6 / mag);
84          };
85        // calculate array coordinates (two lines orthogonal to the path ↩
              vector)
```

```
86        var arr = [{
87          x: (norm(x4 - x3) + norm(y4 - y3) + x4),
88          y: (norm(y4 - y3) + norm(x4 - x3) + y4)
89        }, {
90          x: (norm(x4 - x3) - norm(y4 - y3) + x4),
91          y: (norm(y4 - y3) - norm(x4 - x3) + y4)
92        }];
93        //Add the arrow to the path
94        path = path + ",M" + arr[0].x + "," + arr[0].y + ",L" + x4 + "," + y4↩
              + ",L" + arr[1].x + "," + arr[1].y;
95        // lets attatch the path with some attributes
96        edge.fg = selfRef.path(path).attr({
97          stroke: "#000",
98          fill: "none"
99        }).toBack();
100     }
101   }
102   //draw the edge. Woohoo!
103   edge.draw();
104   return edge;
105 };
```

Listing A.1: connection(obj1, obj2)

A.2 draw_graph_json(graphjson)

```
1  function draw_graph_json(graphjson) {
2      var width, height, i, ac, keys, persons, j;
3      $("#canvas").empty();
4      //we create a new graph.
5      g = new Graph();
6      //What the size of is our canvas?
7      width = $(document).width() - 500;
8      height = $(document).height() - 160;
9      //add some info from the graph to the page
10     $("#title").html(graphjson.name);
11     $("#header").html("Graph updated at " + graphjson.last_updated_at + " ↩
           by " + graphjson.last_updated_by + ". Graph came from server at " ↩
           + graphjson.server);
12     //Loop through all the nodes in the graph
13     for(i = 0; i < graphjson.nodes.length; i++) {
14         //is it access controlled?
15         ac = graphjson.nodes[i].access_controlled == true ? true : false;
16         keys = new Array;
17         persons = new Array;
18         //If it's access controlled, we add the keys and persons
19         if(ac) {
20             for(j = 0; j < graphjson.nodes[i].keys.length; j++) {
21                 keys.push(graphjson.nodes[i].keys[j].key_id);
22             }
23             for(j = 0; j < graphjson.nodes[i].persons.length; j++) {
24                 persons.push(graphjson.nodes[i].persons[j].person_id);
25             }
26         }
27         //add the node
28         g.addNode({
29             label: graphjson.nodes[i].name,
30             description: graphjson.nodes[i].description,
```

```
31              access_controlled: ac,
32              atnode: new Array,
33              keys: keys,
34              persons: persons,
35              x_pos: graphjson.nodes[i].x_pos,
36              y_pos: graphjson.nodes[i].y_pos,
37              id: graphjson.nodes[i].node_id,
38              edges: []
39          });
40      }
41      //and and the edge
42      for(i = 0; i < graphjson.edge.length; i++) {
43          g.addEdge(graphjson.edge[i].source, graphjson.edge[i].sink);
44      }
45      // set up the renderer on the div with id "canvas"
46      renderer = new Graph.Raphael("canvas", g, width, height);
47      //and look how pretty it is :D
48      renderer.draw();
49      siteLog(".. drawing graph<br />");
50  }
```

Listing A.2: draw_graph_json(graphjson)

A.3 getActions(UserContext context)

```
1   private void getActions(UserContext context)
2   {
3    //Load our persons
4    GraphPersons per = GraphPersons.LoadFromFile("graphpersons.xml");
5
6    //Loop through them
7    foreach (GraphPersonsPerson person in per.person)
8    {
9     //We are creating a DFS, so we use a stack
10    Stack<VisitedHelper> s = new Stack<VisitedHelper>();
11    //And a helper so we avoid an infinite loop
12    List<graphNode> visited = new List<graphNode>();
13    //Can we enter the first node? If we can, we push it to the stack
14    if (canEnterNode(gra.nodes[0], person))
15    {
16     s.Push(new VisitedHelper() { to = gra.nodes[0], from = null });
17    }
18
19    //Then we go through the stack
20    while (s.Count > 0)
21    {
22     //This one knows where we go from and to
23     VisitedHelper n = s.Pop();
24
25     //We shouldn't visit it, if we have allready visited it. Lambda ↩
           predicates HOORAY!
26     if (!visited.Contains(gra.nodes.Find(te => te.node_id == n.to.node_id)))
27     {
28      //Take a nap, so we can see what's happening
29      Thread.Sleep(speed);
30
31      //Where is the person now?
32      graphNode from = whereIsPerson(person);
```

```
33
34      //Add person to the new node
35      addPersonToGraph(gra.nodes.Find(t => t.node_id == n.to.node_id), person↩
            );
36      Console.WriteLine("Person " + person.p_id + " goes from node " + (from ↩
            == null ? "NULL" : from.node_id) + " to node " + n.to.node_id);
37
38      //Say it to the client
39      actiontransferer transfer = new actiontransferer() { to = n.to.node_id,↩
             from = (from == null ? null : from.node_id), person_id = person.↩
            p_id };
40      sendMessage(ResponseType.FetchDFSAction, transfer, context);
41      //Remember that we visited this node
42      visited.Add(n.to);
43
44      //Now wee will look at the edges, so we can create a DFS.
45      //int edgecounter = 0;
46      foreach (graphEdge edge in gra.edge)
47      {
48       //We only look at edges exiting this node
49       if (edge.source == n.to.node_id)
50       {
51
52        //edgecounter++;
53        //We shouldn't stack the node if we already visited it
54        if (!visited.Contains(gra.nodes.Find(x => x.node_id == edge.sink)))
55        {
56         //We haven't been here, so maybe we can stack it?
57         graphNode newnode = gra.nodes.Find(x => x.node_id == edge.sink);
58         //Is the person allowed to enter the node?
59         if (canEnterNode(newnode, person))
60         {
61          //He is, so stack it.
62          s.Push(new VisitedHelper() { from = n.to, to = newnode });
63         }
64        }
65       }
66      }
67     }
68    }
69   }
70  }
```

Listing A.3: getActions(UserContext context)

A.4 getRamdomActions(UserContext context)

```
1   private void getRandomActions(UserContext context)
2   {
3    //Load persons
4    GraphPersons per = GraphPersons.LoadFromFile("graphpersons.xml");
5
6    //Loop through all persons
7    foreach (GraphPersonsPerson person in per.person)
8    {
9     //Check if the person can enter the first node
10    if (canEnterNode(gra.nodes[0], person))
11    {
```

```
12      //If he can, send to the client that the person moves from NULL to the ↩
            first node
13      actiontransferer transfer = new actiontransferer() { to = gra.nodes[0].↩
            node_id, from = null, person_id = person.p_id };
14      sendMessage(ResponseType.FetchRandomAction, transfer, context);
15    }
16    else
17    {
18      //If he can't. Say it and break the loop, so we can look at the next ↩
            person
19      string temp = person.p_id + " could not enter start node";
20      sendMessage(ResponseType.SendConsoleMessage, temp, context);
21      break;
22    }
23    //Get a random node from all nodes
24    Random rnd = new Random();
25    int randInt = rnd.Next(1, gra.nodes.Count);
26    //And set it as the target
27    graphNode targetNode = gra.nodes[randInt];
28    //And tell the target to the user
29    sendMessage(ResponseType.SendConsoleMessage, person.p_id + " is looking ↩
          for node with name " + targetNode.name, context);
30    //We start at the first node
31    graphNode currentNode = gra.nodes[0];
32    //We count how many moves we make - we won't go on forever.
33    int tryCounter = 0;
34    //While we are NOT at the target.
35    while (targetNode != currentNode)
36    {
37      //Sleep a bit, so we can see what happens
38      Thread.Sleep(speed);
39      //Find all edges going out from the current node, that we are allowed to↩
            use.
40      List<graphEdge> edges = listOfEdgesAllowed(currentNode, person);
41      //If there are node edges we can use we are stuck. Dammit!
42      if (edges.Count == 0)
43      {
44        sendMessage(ResponseType.SendConsoleMessage, person.p_id + " is stuck ↩
              at node with name " + currentNode.name + "<br />Failed to reach ↩
              target node", context);
45        break;
46      }
47      //If there are node edges OR we can't enter any of the outgoing edges we↩
            are stuck. Dammit!
48      if (edges.Count == 0 || canEnterAnyEdge == false)
49      {
50        sendMessage(ResponseType.SendConsoleMessage, person.p_id + " is stuck ↩
              at node with name " + currentNode.name + "<br />Failed to reach ↩
              target node", context);
51        break;
52      }
53      //Lucky for us, we aren't stuck. So take a random edge, that we can use,↩
            from the node.
54      int randIntEdge = rnd.Next(edges.Count);
55      graphEdge goViaEdge = edges[randIntEdge];
56      //Set the new node
57      graphNode newNode = gra.nodes.Find(n => n.node_id == goViaEdge.sink);
58      //if the person is allowed to enter the node
59      if (canEnterNode(newNode, person))
60      {
61        //Then tell it to the client
62        actiontransferer transfer = new actiontransferer() { to = newNode.↩
              node_id, from = currentNode.node_id, person_id = person.p_id };
63        sendMessage(ResponseType.FetchRandomAction, transfer, context);
64        //And set the new node
```

```
65      currentNode = newNode;
66     }
67     //If the target is the new node
68     if (targetNode == currentNode)
69     {
70      //Speak!
71      sendMessage(ResponseType.SendConsoleMessage, person.p_id + " reached ↩
            node with name " + currentNode.name, context);
72     }
73     //If not
74     else
75     {
76      //We will not go on foreever. Stop after 50 tries.
77      tryCounter++;
78      if (tryCounter == 50)
79      {
80       sendMessage(ResponseType.SendConsoleMessage, person.p_id + " didn't ↩
             reach " + targetNode.name + " in " + tryCounter + " steps.<br />↩
             Giving up while in "+currentNode.name, context);
81       break;
82      }
83     }
84    }
85   }
86  }
```

Listing A.4: getRandomActions(UserContext context)

A.4.1 graph.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSpy v2011 rel. 2 (http://www.altova.com) by Test (Test)↩
       -->
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:altova="http↩
       ://www.altova.com/xml-schema-extensions" elementFormDefault="qualified↩
       " attributeFormDefault="unqualified">
4   <xs:element name="graph">
5    <xs:annotation>
6     <xs:documentation>Root node element</xs:documentation>
7    </xs:annotation>
8    <xs:complexType>
9     <xs:sequence>
10     <xs:element name="name" type="xs:string"/>
11     <xs:element name="last_updated_at" type="xs:dateTime">
12      <xs:annotation>
13       <xs:documentation>Ex: 2012-09-23T09:30:10+01:00</xs:documentation>
14      </xs:annotation>
15     </xs:element>
16     <xs:element name="last_updated_by" type="xs:string"/>
17     <xs:element name="server" type="xs:string"/>
18     <xs:element name="nodes"  minOccurs="1" maxOccurs="1" >
19      <xs:complexType>
20       <xs:sequence>
21        <xs:element name="node" maxOccurs="unbounded">
22         <xs:complexType>
23          <xs:sequence>
24           <xs:element name="description" type="xs:string"/>
25           <xs:element name="name" type="xs:string"/>
26                    <xs:element name="x_pos" type="xs:int"/>
```

```
27                    <xs:element name="y_pos" type="xs:int"/>
28                    <xs:element name="access_controlled" type="xs:boolean"↩
                          />
29          <xs:element name="keys" minOccurs="0">
30           <xs:complexType>
31            <xs:sequence>
32             <xs:element name="key" minOccurs="0" maxOccurs="unbounded">
33              <xs:complexType>
34               <xs:attribute name="key_id">
35                <xs:simpleType>
36                 <xs:restriction base="xs:string">
37                  <xs:pattern value="k_[0-9]+"/>
38                 </xs:restriction>
39                </xs:simpleType>
40               </xs:attribute>
41              </xs:complexType>
42             </xs:element>
43            </xs:sequence>
44           </xs:complexType>
45          </xs:element>
46          <xs:element name="persons" minOccurs="0">
47           <xs:complexType>
48            <xs:sequence>
49             <xs:element name="person" minOccurs="0" maxOccurs="unbounded">
50              <xs:complexType>
51               <xs:attribute name="person_id">
52                <xs:simpleType>
53                 <xs:restriction base="xs:string">
54                  <xs:pattern value="p_[0-9]+"/>
55                 </xs:restriction>
56                </xs:simpleType>
57               </xs:attribute>
58              </xs:complexType>
59             </xs:element>
60            </xs:sequence>
61           </xs:complexType>
62          </xs:element>
63         </xs:sequence>
64         <xs:attribute name="node_id" use="required">
65          <xs:simpleType>
66           <xs:restriction base="xs:string">
67            <xs:pattern value="n_[0-9]*"/>
68           </xs:restriction>
69          </xs:simpleType>
70         </xs:attribute>
71        </xs:complexType>
72       </xs:element>
73      </xs:sequence>
74     </xs:complexType>
75     <xs:key name="node_key">
76      <xs:selector xpath="node"/>
77      <xs:field xpath="@node_id"/>
78     </xs:key>
79    </xs:element>
80    <xs:element name="edge"  minOccurs="0" maxOccurs="unbounded">
81     <xs:complexType>
82      <xs:attribute name="source" use="required">
83       <xs:simpleType>
84        <xs:restriction base="xs:string">
85         <xs:pattern value="n_[0-9]+"/>
86        </xs:restriction>
87       </xs:simpleType>
88      </xs:attribute>
89      <xs:attribute name="sink" use="required">
90       <xs:simpleType>
```

```
91          <xs:restriction base="xs:string">
92           <xs:pattern value="n_[0-9]+"/>
93          </xs:restriction>
94         </xs:simpleType>
95        </xs:attribute>
96       </xs:complexType>
97      </xs:element>
98     </xs:sequence>
99    </xs:complexType>
100   <xs:keyref name="sink_node_ref" refer="node_key">
101    <xs:selector xpath="edge"/>
102    <xs:field xpath="@sink"/>
103   </xs:keyref>
104   <xs:keyref name="source_node_ref" refer="node_key">
105    <xs:selector xpath="edge"/>
106    <xs:field xpath="@source"/>
107   </xs:keyref>
108  </xs:element>
109 </xs:schema>
```

A.5 graphpersons.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSpy v2011 rel. 2 (http://www.altova.com) by Test (Test)↩
       -->
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="↩
       qualified" attributeFormDefault="unqualified">
4   <xs:element name="GraphPersons">
5    <xs:complexType>
6     <xs:sequence>
7      <xs:element name="Persons" minOccurs="1" maxOccurs="1">
8       <xs:complexType>
9        <xs:sequence minOccurs="0" maxOccurs="unbounded">
10        <xs:element name="person">
11         <xs:complexType>
12          <xs:sequence minOccurs="0" maxOccurs="unbounded">
13           <xs:element name="key">
14            <xs:complexType>
15             <xs:attribute name="k_id">
16              <xs:simpleType>
17               <xs:restriction base="xs:string">
18                <xs:pattern value="k_[0-9]+"/>
19               </xs:restriction>
20              </xs:simpleType>
21             </xs:attribute>
22            </xs:complexType>
23           </xs:element>
24          </xs:sequence>
25          <xs:attribute name="p_id">
26           <xs:simpleType>
27            <xs:restriction base="xs:string">
28             <xs:pattern value="p_[0-9]+"/>
29            </xs:restriction>
30           </xs:simpleType>
31          </xs:attribute>
32         </xs:complexType>
33         <xs:unique name="key">
34          <xs:selector xpath="key"/>
```

```
35            <xs:field xpath="@k_id"/>
36          </xs:unique>
37        </xs:element>
38      </xs:sequence>
39     </xs:complexType>
40    </xs:element>
41   </xs:sequence>
42  </xs:complexType>
43 </xs:element>
44 </xs:schema>
```

Bibliography

[1] Christian W. Probst and René Rydhof Hansen. An extensible analysable system model. 2008.

[2] René Rydhof Hansen Christian W. Probst and Flemming Nielson. Where can an insider attack? 2007.