Danmarks
Tekniske Universitet

DTU

Udvikling af diagrammeringsværktøjet "Diana"

Navn:
Lars Friberg
Studienummer:
s052423

Afleveringsdato:
16. september 2013

Vejledere:
Philip Bille, DTU
Tommy Balle, Statens Arkiver

Rapportnummer:
IMM-B.Eng-2013-23

DTU Compute
Institut for Matematik og Computer Science

STATENS ARKIVER
RIGSARKIVET

Resume

Dette projekt handler om udvikling af et diagrammeringsværktøj til at tegne ER diagrammer over en database. Jeg beskriver i denne rapport, hvordan man på en række forskellige måder, kan placere tabeller og deres relationer i en brugergrænseflade. I mine løsninger kommer jeg ind over brugen af grafteori, til at tegne diagrammet. Målet er ikke at udvikle et færdigt værktøj, men at komme med en række løsninger på hvordan problemet kan løses og vise dem i en kørende prototype.

Projektet er lavet i samarbejde med Statens Arkiver, til brug i deres software til test af databaser der afleveres til Statens Arkiver.

Forord

Denne rapport er min afsluttende eksamensopgave på Diplomingeniør IT-uddannelse på Danmarks Tekniske Universitet (DTU). Vejen til dette punkt at kunne aflevere eksamensopgave har været lang. Nu er jeg i mål. Det kan jeg bl.a. takke Rigsarkivet / Statens Arkiver for, i kraft af at jeg kunne have mit praktikophold der, i foråret 2013, samt muligheden for at lavet dette projekt hen over sommeren. Specielt vil jeg takke min vejleder Tommy Balle, som har været en god hjælp genne hele processen. Jeg vil også takke min vejlede fra DTU, Philip Bille.

København den 16. september 2013

Lars Friberg

Indhold

1. Introduktion

1.1. Indledning

Dette projekt handler om at lave et værktøj til at visualisere databaser i form af et ER-lignede diagram (Entity-relationship). Baggrunden for dette projekt er arbejdet i Statens Arkivers(SA) sektion for digital arkivering, DIARK, som i deres testværksted laver bl.a. visuelle test af databaser.
Dette værktøj ville kunne bruges til mange andre formål, hvor et diagram over databasen vil være nyttigt. Det primære fokus i dette projekt er et tegne selve diagrammet og ikke så meget, hvor og hvordan det skal bruges og det der ligger udenom. Projektet vil ikke ende ud i et færdigt værktøj, som kan det hele, men vil være en kørende prototype, som demonstrer nogle af de aspekter som ligger i opgaven. I praksis består den af en række løsninger, som hver især giver bud på hvordan problemet kan løses.

1.1.1. Sektionen for digital arkivering (DIARK)

DIARKS ansvar er at udarbejde regler, metoder, standarder og vejledninger i relation til digital arkivering, samt at modtage og teste data fra offentlige myndigheders it-systemer. En væsentlig del af modtagelsen af digitale arkivalier består i, at teste at de overholder de gældende regler for aflevering af digitale arkivalier (Kulturministeriets bekendtgørelse nr. 1007 af 24. august 2010). Dette gøres ved at lave en række test. Nogle fortages automatisk ved brug af DIARKs testprogram ADA, men da det ikke er muligt at udvikle automatiske test til alt, f.eks. om beskrivelser af tabeller og felter giver mening, gennemføres der også en række visuelle test. Det er her dette diagrammeringsværktøj kommer ind i billedet og bliver en overbygning til det værktøj jeg har udviklet i praktikperiode, kaldet ”SQLtool”.

1.1.2. SQLtool

Alle digitale afleveringer (også kaldt arkiveringsversioner) indeholder en database samt evt. dokumenter (konverteret til TIFF eller JPEG-2000), lyd, video samt geodata (GML).

Da der altid findes databaser i en arkiveringsversion, er det en væsentlig del af arbejdet i DIARKs testværksted at teste disse databaser.

I min praktikperiode i DIRAK i foråret 2013 har jeg udviklet et værktøj til at vise databasens struktur og data til brug i testen kaldet ”SQLtool”. Når afleveringen testes med ADA genrejses databasen fra XML filer til en SQLite database. Ligeledes gemmes alle metadata om databasens tabeller og felter i en anden database.

SQLtool kobler disse to databaser sammen og muliggør at se strukturer og data på en overskuelig og konsistent facon.

I dette projekt vil jeg gøre brug af SQLtool som ramme for en User Control der skal tegne et diagram over en arkiveringsversions database.

Jeg vil således gøre brug af de klasser, jeg har udviklet i praktikken til at opbevare databasens struktur, samt den overordnede brugergrænseflade. Dette er således ikke en del af løsningen i dette projekt og udgør således fundamentet for diagramdelen.

1.2. Problemformulering

Til at understøtte DIARKs arbejde med databaser skal der udarbejdes en User Control, der kan vise en databases struktur, med tabeller og deres indbyrdes relationer.

User Controllen skal automatisk tegne databasen op i form af et ER-lignede diagram. Efterfølgende skal man kunne flytte tabeller rundt på skærmen. Specielt vil fokus være på hvordan man kan tegne diagrammet op.

Der skal udarbejdes forskellige løsninger på dette problem og eventuelt kombinationer af løsningerne.

1.3. Projektplan

Projektet er af 12 ugers varighed, herudover er der indlagt 3 ugers ferie, 8-26. juni.

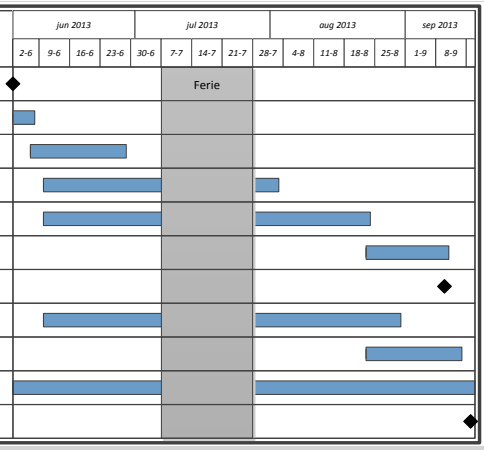
Jeg har delt projektet op i en række delopgaver, som jeg vil arbejde med sideløbende. Således vil veksle mellem at arbejde på den teoretiske del af rapporten, læse relevant litteratur, arbejde med analysedelen og arbejde med den konkrete implementering og programmeringen på baggrund af den valgte teori. Hen over projektperioden vil jeg vægte de enkelte opgaver forskelligt.

Jeg vil prøve at gribe problemet an fra forskellige vinkler og udvikle en række løsninger.

Selve implementeringen af nye ting skal være færdig sidst i august (ved starten af projektuge 10) Herefter vil jeg ikke lave flere nye løsninger, men finpudse det jeg har udført og teste applikationen samt færdiggøre rapporten.

Oversig over projektperioden

Id	Opgavenavn	Startdato	Slut	Varighed
1	Projektstart	03-06-2013	03-06-2013	0d
2	Projektopstart	03-06-2013	07-06-2013	5d
3	Analyse	07-06-2013	28-06-2013	16d
4	Design	10-06-2013	02-08-2013	40d
5	Implementering – med nye funktioner	10-06-2013	23-08-2013	55d
6	Implementering – færdiggørelse	23-08-2013	10-09-2013	13d
7	Kodestop	10-09-2013	10-09-2013	0d
8	Test løbende	10-06-2013	30-08-2013	60d
9	Test afsluttende	23-08-2013	13-09-2013	16d
10	Rapport skrivening	03-06-2013	16-09-2013	76d
11	Aflevering	16-09-2013	16-09-2013	0d



2. Analyse

I dette kapitel analyser jeg problemet. Der kigges på hvordan problemet kan løses og hvilke retninger der arbejdes ud af.

2.1. Entity-relationship model

En entity-relationship (ER) model er en måde at beskrive en samling data på. Typisk en database. Man kan beskrive data på forskelige niveauer. (Wikipedia artikel 1)

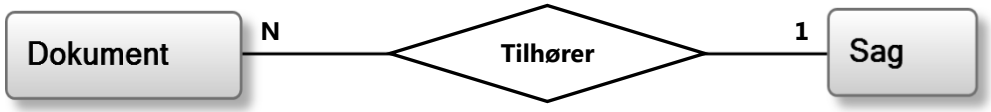
- Det øverste niveau er den konceptuelle datamodel, som er den mindst detaljerede, og beskriver i de store linjer, hvilke data som er en del af modellen. Den konceptuelle datamodel kan danne grundlag for en eller flere logiske datamodeller.
- Den logiske datamodel er en mere detaljeret model, og har flere detaljer med om data. Igen kan den logiske datamodel danne grundlag for flere fysiske datamodeller.
- Den fysiske datamodel har alle relationer og nøgler med, og kan som sådan bruge til at oprette selve databasen. En fysisk datamodel vil tit også være bundet til et bestemt databasesystem da forskellige systemer har forskellige udformninger.

Da dette projekt går ud på at tegne en model over en allerede eksisterende database, er opgaven lidt omvendt af de 3 typer modeller. Jeg starter med en database, og tegner så en model over den. Den skal ligge mest op af den fysiske model, da opgaven er at vise hvordan den konkrete databases datamodel er sammensat.

2.1.1. ER diagram typer

Siden man i 1970'erne begyndte at arbejde med sådanne modeller, er der udviklet en række forskellige måder at diagramere dem på. De fleste bruger at tegne tabellerne/entiteterne op som kasser og forbinde dem med steger. Nogle har symboler i enderne til at vise kardinaliteten mellem tabellerne, andre har tal. I mange af tilfælde er det også beskrevet med et par ord om hvordan det indbyrdes forhold mellem relationerne ser ud.

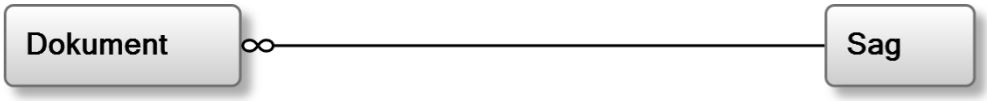
En af de første til at skrive om ER modellerne var Peter Chen som i 1976 (Chen 1976) beskrev sin model. Her er entiteterne tegnet med en kasse. Relationen mellem entiteter tegnes med en streg med en rombe på med en beskrivelse af relationen.



Figur 1: Eksempel på en Chen ER-model

Da jeg til at tegne mine diagrammer henter metadata fra databasen har jeg ikke sådanne beskrivende ord til at beskrive relationerne til rådighed, men kun navnet på fremmednøglen. Dette kunne med fordel stå sammen med relationen. Jeg tegner således fremmednøglerne med en streg mellem tabellerne. Jeg marker mange tabellen i relationen med ∞ (uendelighedstegn), der hvor stregen rører tabellen. Man kunne også sætte kardinalitet på i form af tal ved siden af stregen, men da mit fokus i dette projekt er at tegne tabeller og streger op, har jeg valgt ikke at fokusere på dette.

I eksemplet i Figur 2 ses en dokument-sag relation. Dokumenttabellen har en fremmednøgle der peger på en sag.



Figur 2: Eksempel på en relation

2.2. Kravspecifikation

Kravspecifikation for diagrammeringsværktøjet "Diana"[1]

Denne kravspecifikation tager udgangspunkt i min problemformulering, og er i store dele uddrag af den systembeskrivelse, som danner grundlag for dette projekt, som min vejleder i Statens Arkiver, Tommy Balle udarbejdede i januar 2013. (se evt. Bilag 1: Systembeskrivelse)

Nedenstående krav er det op i 4 grupper: Overordnede krav for projektet. Funktionalitet, er det som er basisfunktioner. Mulig funktionalitet, er yderlige funktioner der kan arbejdes med. Og sidst er der nogle krav til stil og dokumentation af koden.

(FK: Funktionelle krav, SK: Stil krav)

2.2.1. Overordnede krav

Der ønskes udarbejdet en applikation, der i en grafisk visning kan projicerer en relationel database på en brugergrænseflade.

Applikationen ønskes udført i programmeringssproget C# og skal kunne afvikles på en Microsoft Windows platform. Applikationen skal udføres således, at den kan etableres i en "User Control", hvor formålet med denne udformning er en højere grad af genanvendelighed.

Applikationen skal også kunne afvikles i en selvstændig udgave.

2.2.2. Funktionalitet

FK1: Applikationen skal via struktureret metadata fra en database eller anden metadatakilde, automatisk kunne projicere databasens entiteter samt kardinalitet i en grafisk visning i form af et ER-lignende diagrammeringsformat.

FK2: Databasens entiteter skal i visningen fremstå med tabelnavn. Der skal være mulighed for at kunne se tabellens feltnavne, samt hvilke felter der udgør henholdsvis primær- og fremmednøgler. Denne visning skal kunne til- og fravælges.

FK3: Alle entiteter skal frit kunne bevæges inden for den overordnede brugergrænseflades afgrænsning.

[1] "Diana" er navnet på dette diagrammeringsværktøj. Det er dog ikke brugt andre steder en her i rapporten. Navnet kommer af at systermer i Statens Arkiver har pige navne, og da diagrammering og Diana har ligheder er dette valgt.

2.2.3. Mulig funktionalitet

FK4: Ved et klik på en enkelt entitet skal data fra den bagvedliggende tabel, der repræsenterer den pågældende entitet, kunne projiceres ud i en selvstændig brugergrænseflade i en struktureret visning, eventuel i form af et dataGridView. Andre projiceringsformer kan komme på tale.

FK5: Det kunne være ønskeligt, at brugeren kan udvælge en eller flere entiteter, som efterfølgende projekteres ud på en brugergrænseflade.

FK6: En funktionalitet der procentvis op- eller nedskalerer brugergrænseflades entiteter.

FK7: En funktionalitet der kan udskrive og persistere brugergrænsefladens entiteter med tilhørende kardinalitet i et grafisk billedformat.

FK8: En mulig persistering af en aktuel visning af entiteter, således at visningen automatisk kan genoptages

FK9: At visningen af entiteter forgår efter en optimeret model, så mulige overlapninger af entiteter så vidt muligt undgås.

FK10: En funktion i forbindelse med visningen af entiteterne der kan til- og frakoble visning af datatyper.

FK11: En modulær til- og frakobling af forskellige typer af databaseservere

2.2.4. Kodestil og dokumentation af kode

SK1: Al kode ønskes dokumenteret ud fra de beskrivelser, som er beskrevet af Microsoft i relation til det valgte udviklingssprog C#[2].

SK2: Ligeledes ønskes programklasser udviklet med så løs kobling som muligt.

SK3: Det vil være ønskeligt, hvis der i høj grad gøres brug af erklærede interface mellem klasser, jævnfør Microsofts anbefalinger.[3]

SK4: Indstillinger til brug for applikationen persisteres i en konfigurationsfil jævnfør Microsofts vejledning.[4]

SK5: Der gøres brug af fejlhåndteringsrutiner således at mulige fejl så vidt muligt rapporteres til bruger på en struktureret facon.[5]

[2] http://msdn.microsoft.com/en-us/library/vstudio/b2s063f7.aspx
[3] http://msdn.microsoft.com/en-us/library/vstudio/ms173156.aspx
[4] http://msdn.microsoft.com/en-us/library/aa730869(v=vs.80).aspx
[5] http://support.microsoft.com/kb/816157

2.3. Projektafgrænsning

I det følgende vil jeg kommentere på nogle af kravene og begrunde nogle af mine prioriteringer. Målet med dette projekt er ikke at stå med den færdige løsning på alle ovenstående krav, men en kørende prototype som viser elementer til en færdig løsning.

Ad "Overordnede krav"

Det er her det grudlæggende krav for projektet er beskrevet kort. En applikation til grafisk visning af en relationel database.

Applikationen skal implementeres i C#. Jeg har brugt C# i min praktikperiode i DIARK/Statens Arkiver, hvor det er det primære udviklingssprog. Ligeledes bruger jeg .Net 4.0 som framework. I min praktik udviklede jeg værktøjet SQLtool. Brugergrænsefladen er her lavet i Windows forms (WinForms) og ved brug af User Controls, således er den delt op i flere mere eller mindre afgrænsede dele.

Diagramtegningen vil jeg også udvikle som en User Control, men hvor resten er udviklet i WinForms har jeg valgt at bruge Windows Presentation Foundation (WPF) til dette projekt. WPF blev lanceret sammen med .Net 3.0 i 2006, som en ny måde at udvikle brugergrænseflader i .Net.

WPF benytter DirectX til at optegne brugerfladen, og den giver også gode muligheder for tegning og skalering af brugerfladen. Ligeledes er paneltypen Canvas også meget nyttig til at tegne f.eks. diagrammer, som i dette projekt.

Jeg vil således ikke udvikle en komplet applikation men kun denne User Control og placere den i SQLtool. Jeg vil kun i meget begrænset omfang ændre i SQLtool.

Ad FK1: Tegning af ER-lignende diagram

Dette er hovedfunktionaliteten i denne applikation. Og er således mit hovedfokus i dette projekt.

Ad FK2: Visning af tabelnavn, felter og nøgler

Visningen af tabellernes felter vil jeg ikke bruge meget tid på. Da det er funktionaliteten i SQLtool i forvejen vil det være oplagt at gøre det muligt at komme fra en tabel i diagrammet til feltinformationer i træet i venstre side af skærmen. Jeg vil således i diagrammet kun vise tabelnavnet.

Ad FK3: Flytning af entiteter

Dette er også en væsentlig funktionalitet i dette projekt og blandt de primære opgaver, men også noget som er med til at komplicere projektet en del.

Ad FK9: Visning efter en optimeret model.

Af de mulige funktioner er FK9 for mig at se den mest interessante. Jo bedre værktøjet er til at tegne entiteter op på skærmen jo mere overskuelig bliver diagrammet. I FK9 er den optimerede visning beskrevet som at tabeller ikke overlapper. Jeg vel prøve at gå lidt videre og forsøge at tegne diagrammet således at steger mellem tabeller heller ikke overlapper hinanden unødigt.

Ad FK6: Skalering af brugergrænsefladen

Som en del af UC'en vil jeg implementer et canvas som kan skaleres hertil er det også nødvendigt med scrollbarer hvis diagrammet overskrider det tilgængelige skærmareal.

Ad FK7: Persistering af diagram i en grafisk billedfil

Denne funktionalitet har jeg også valgt at implementere da det er relativt enkelt at gemme et canvas som en grafikfil. Desuden vil jeg gøre det for at kunne dokumentere skærmbilleder på en hurtig og enkel måde. Jeg har valgt at kunne gemme i PNG filer, men da Statens Arkivers hovedstandart for grafikfiler er TIFF, har jeg også valgt at implementer dette format.

Ad SK1-5: Kodestil og dokumentation

I min udvikling vil jeg tage udgangspunkt i disse krav til kodestil, og koden dokumenteres med kommentarer og beskrivelser af metoder m.m.

Fravalg

Ligeledes er der en række funktionelle krav jeg har valg fra i dette projekt. FK4 og FK11 er funktionalitet som jeg mener primært hører til i SQLtool og som jeg således ikke vil arbejdes videre med. FK10 vil jeg heller ikke arbejde med, da jeg i forhold til FK2 ikke har valgt at vise disse oplysninger. At kunne persistere en visning, som efterfølgende skal kunne genoptages (FK8) vil jeg heller ikke arbejde med, da de ligger uden for mit fokus som er at tegne diagrammet. FK5 som handler om at tegne en del af diagrammet, vil være simpel at implementer når tegningen af diagrammet er på plads, men er nedprioriteret i dette projekt. Der er mange måde at gå videre, men dette beskæftiger jeg mig ikke med i dette projekt.

2.4. Teoretiske metoder

Jeg vil prøve at komme med forskellige bud på hvordan et ER diagram kan tegnes op. Mit udgangspunkt er et grundlæggende enkelt udseende diagram som beskrevet i 2.1.1 (Figur 2: Eksempel på en relation). Tabeller er vist som kasser og relationer som streger mellem kasserne. Dog har jeg valgt at stregerne kun tegnes som vandrette og lodrette streger, med eventuelle knæk på.

De forskellige bud – tegnemodeller – går fra at være helt simple metoder, til at brug af grafteori og specielt med fokus på graftegning.

Udgangspunktet for flere tegnemodellerne er en heuristisk løsning hvor jeg starter med den tabel med de fleste relationer og projicerer så ud fra den i forskellige retninger.

Nye modeller vil jeg så udvikle med baggrund i de andre og ved kombinere de forskellige metoder.

2.4.1. Definitioner

I mine beskrivelser af mine løsninger vil jeg komme til at omtale enkelte dele af diagrammet mange gange. For at gøre det klart hvad der menes kommer her nogle definitioner.

I diagrammet vil mine entiteter være tabeller. Jeg vil primært bruge betegnelsen tabel, men i afsnit om grafteori vil jeg bruge begrebet knude.

Streger mellem tabeller, viser relationen mellem tabellerne, i form af fremmednøgler. Som udgangspunkt vil jeg kalde disse streger for relationer, men igen vil jeg bruge begrebet kanter i afsnit om grafteori.

Når min relationer er delt i flere dele, kalder jeg de enkelte dele for segmenter.

Jeg tegner mit diagram op på et panel kaldet `Canvas`, jeg vil henvise til min tegneflade som canvaset.

Jeg bruger i dette projekt to former for modeller. Den første er datamodel, som er en beskrivelse af hvordan data er struktureret. Datamodellerne er inputtet til min User Control. En ER model er en datamodel. Den anden model jeg bruger er tegnemodeller. Det er de klasser og metoder som jeg bruger til at tegne datamodellen.

Jeg arbejder primært med udvikling af en User Control og nogle steder vil jeg forkorte dette til UC.

2.4.2. ER diagram som graf

En måde at betragte en ER datamodel på, kan være som en graf. En graf er en samling knuder(vertex), som er bundet sammen af kanter(edge). Hvis vi betragter datamodelen som en graf, hvor tabellerne er knuderne og relationerne er kanter, kan vi bruge algoritmer fra grafteorien på datamodellen. Relationer i datamodellen har retninger, og det kan kanterne i en graf også godt have. Men i det her projekt vil jeg se på dem som ikke-orienterede.

Graftegning

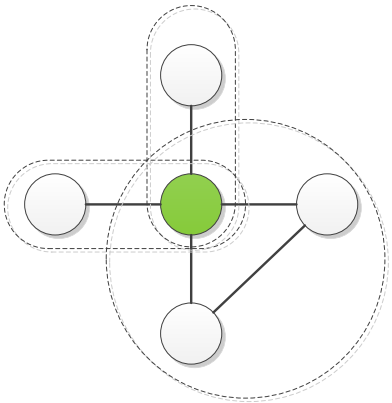
En graf i sig selv er bare beskrivelsen af knuder og deres forbindelser (kanter). Men i relation til grafteorien er der også graftegning. Det er algoritmer, der kan anvendes til at projicerer grafen. Der er mange måder at projicerer grafer op på. De kan se meget forskellige ud alt efter, hvad det er for en graf, man vil vise, og hvad man skal bruge tegningen til. Da ER modeller tit er tegnet op med vandrette og lodrette streger vil algoritmer der kan tegne dette være oplagt (ortogonal graftegning). Men andre dele af graftegning kombineret med nogle heuristiske tilgange, kan også være brugbare.

Plane grafer

Inden for graftegning beskæftiger man sig med plane grafer, som kan projiceres uden at kanterne krydser hinanden. Det ville være optimalt, hvis ER modellerne kunne projiceres op på denne måde, men det er ikke altid muligt. Hvis en graf ikke kan projiceres plant må man forsøge at gøre det med færreste krydsende kanter.

Uadskillelige komponenter

Når en graf skal projiceres kan det være en fordel at dele det op. I mange tilfælde vil nogle tabeller være samlet i gruppe af tabeller. En måde at dele den op, er i uadskillelige komponenter. En komponent er et udsnit af grafen. Når den kan betegnes som uadskillelig, betyder det at maen ikke kan dele komponenten i mindre komponenter ved kun at fjerne en knude. De knuder som kan dele en graf i komponenter kan kaldes deleknuder (cutvertices). På Figur 3 kan man se et eksempel på en graf med deleknuder og uadskillelige komponenter. I forhold til at projicere grafen kan man bruge de enkelte komponenter som grupperinger som tegnes ved siden af hinanden.



Figur 3: Eksempel på en graf. Den grønne knude er en deleknude. De stiplede linjer viser uadskillelige komponenter.

Hvis hver enkelt uadskillelige komponent er plant, vil den samlede graf være plan.

Til at finde de uadskillelige komponenter bruger jeg en algoritme som tager udgangspunkt i Hopcroft-Tarjan's udgave af Depth-First Search (DFS) (Even 2012, side 49ff). Grundlæggende går DFS algoritmen ud på at gennemløbe alle knuder og kanter og nummerer knuderne fortløbende, samt holder styr på den forgående knude. I dette tilfælde hvor jeg vil bruge algoritmen til at finde uadskillelige komponenter, er der nogle tilføjelser som kan identificere disse komponenter (Even 2012, side 52ff).

3. Design og implementering

Som beskrevet tideliger (i afsnit 1.1.2) bruger jeg mit databaseværktøj, SQLtool, som jeg har udviklet i min praktikperiode som initialapplikation for denne User Control. Således bruger jeg også den datastruktur, som jeg udviklede til SQLtool som input til min User Control.

3.1. Testmiljø

Til at arbejde med og teste tegne modellerne har jeg tilpasset SQLtool en lille smule. I stedet for at indlæse en SQLite database med data og en anden med metadata[6], har jeg valgt at hente datastrukturen fra en XML fil. I en arkiveringsversion er database gemt som en XML filer. Selve strukturen er gemt i filen tableIndex.xml. og dataindholdet i andre XML filer. Da jeg kun skal bruge strukturen, og ikke data, er tableIndex.xml nok. Jeg har tilføjet en mulighed i SQLtool, som læser alle XML filer i en bestemt mappe og ved tryk på en knap indlæses strukturen ind. Den er også sat automatisk at indlæse den XML fil ind som sidst er brugt.

En anden grund til at jeg bruger XML filer er, at det er nemt at rette i strukturen ved hjælp af tableIndex.xml editoren, som jeg også har udviklet i min praktikperiode.

Det er dog fortsat muligt at indlæse .av og .test databaserne.

3.1.1. Testdata

Til brug for test under implementeringen har jeg bruge forskellige databaser, en jeg selv har konstrueret, en anden har jeg har modificeret samt en er fra en rigtige arkiveringsversion. Databaserne spender fra nogle simple databaser med få tabeller, til en med 47 tabeler.

3.2. Opbygning af User Control

Da jeg har det omkringlæggende program til min User Control i mit testmiljø. Er det selve UC'en som jeg beskæftiger mig med. Da målet er at lave flere løsninger på, hvordan databasen kan projiceres, har jeg lavet et design hvor hoved klassen tager sig af nogle

[6] De to databaser er henholdsvis .av og .test som ADA generer ud fra XML filerne i arkiveringsversionen. Af hensyn til performance indeholder .av kun navnet på tabeller og felter, men ikke deres datatyper, alle felter er text. Hvis man skal se typer, primærnøgler, fremmednøgler m.m. skal man kigge i .test databasen.

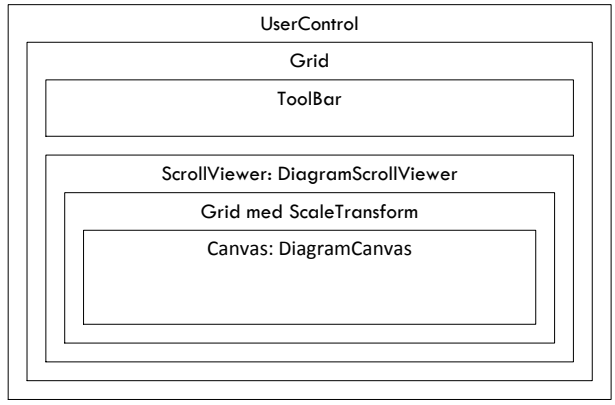
basale funktioner som at have det canvas som diagrammet tegnes på, mulighed for at flytte rundt på tabeller, og samtidig kalde de relevante metoder i tegnemodellerne.

Tegnemodellerne tager sig af at placere tabeller, tegne streger, og opdatere dem når en tabel flyttes.

User Controllen er placeret i klassen `UC_ERdiagram`.

3.2.1. `UC_ERdiagram`s opbygning

Her vil jeg kort beskrive `UC_ERdiagram`'s opbygning. På Figur 4 kan man se hvilke paneler og elementer der ligger inden i hinanden. Brugergrænsefladen i User Controllen består af en værktøjslinje i toppen, hvor man kan vælge tegnemodel, zoome, og gemme et billede af diagrammet.



Figur 4: Overigt over UserControlens opbygning

Tegneområdet som er under værktøjslinjen, er et panel af typen `Canvas`. På et canvas kan man placere andre controller og figurer der nedarver fra `UIElement` ligesom på andre paneler. Men fordelen ved `Canvas` er, at man kan placere elementer med et koordinat i forhold til kanten. I princippet til alle kanter, men i dette projekt bruger jeg kun toppen og venstre side.

For at have muligheden for at scrolle hvis diagrammet går ud over det synlige område, er der en `ScrollViewer`. Et af kravene (FK6) er at diagrammet skal kunne skaleres. I WPF er det nemt at skalere elementerne, således har jeg et panel (`Grid`) med `ScaleTransform` til at styre skaleringen af canvaset.

3.2.2. Interface: `IDrawingModel`

For at adskille hovedklassen i UC'en fra de forskellige tegnemodeller har jeg defineret et interface (`IDrawingModel`), som har nogle grundlæggende metoder som bruges i tegnemodellerne. Den første metode er Draw, som kaldes når diagrammet ønskes tegnet på skærmen. Så er der 3 metoder til flytning. En når musen trykkes ned (`StartMove`), en når musen flyttes (`OnMove`), samt en når musen slippes igen (`AfterMove`). Der er også et par properties, hvor referencen til canvaset og de tabeller der skal tegne sættes. Det er ikke alle

tegnemodeller der bruger alle metoder, men de implementere alle interfacet og `UC_ERdiagram` behøver ikke vide hvilke metoder der bruges.

3.2.3. Klasse: `DiagramTable`

Data kommer som tidligere beskrevet i form af en datamodel som jeg har lavet i SQLtool. `DiagramTable` holder en reference til tabellen, og har nogle lister over relationer, desuden er der også nogle properties der bruges af nogle tegnemodeller. Når der i UC'en refereres til en tabel er det typisk til en `DiagramTable`, hvorfra selve tabellen så kan tilgås, men også `UC_TableBox` samt relationer i form af `GraphEdge` og `ERConnector`.

3.2.4. Klasse: `GraphEdge`

Denne bruges til at beskrive relationerne mellem tabellerne. Grundlæggende består den af referencer til to `DiagramTable` objekter, som repræsentere de to tabeller, den er en forbindelse mellem. Desuden har den et navn som er constraint navnet, som relationen har. Klassen har to sammenligningsmetoder. Den ene er en override af `Equal` metoden fra `System.Object`. Den sammenligner et andet `GraphEdge` objekt med selv, ved at sammenligne `ConstraintName`, `StartTable` og `EndTable`. Den anden metode sammenligner også med en anden GraphEdge, og afgør om de begge er forbindelse mellem de samme to tabeller. Desuden er der en property, som nogle tegnemodeller bruger til at afgøre om forbindelsen har været brugt.

3.2.5. Tegning af elementer (`UC_TableBox` og `ERConnector`)

På tegneområdet (canvaset) tegnes tabeller (`UC_TableBox`) og deres relationer (`ERConnector`). Disse to klasser er implementeret som UserControls og implementere begge interfacet `IDiagramElement` (se nedenfor)

Klasse: `UC_TableBox`

Til at tegne en tabel har jeg en UserControl, som modtager en `DiagramTable` i sin constructor. På nuværende tidspunkt viser den kun tabellens navn. Men da den jo har adgang til alle informationer om felter og nøgler, kunne den også vise dette.

Klasse: `ERConnector`

`ERConnector` er den UserContol som viser relationerne, tegnet som streger mellem tabellerne. Placering af stregerne står tegnemodellerne for, ved at sætte punkterne i en property, hvorefter UC'en tegner dem op.

I den ende hvor mange tabellen af en relation er placeres et ∞ (uendelighedstegn).

Udover de metoder og properties som `IDiagramElement` definerer, er der også nogle som er specifikke for `ERconnector`. Der findes en property, som har med afstanden fra start

tabellen og slut tabellen. Til brug for at manipulere på relationen er der metoden `MoveSegmentTo`, hvor en del, hvis der er flere dele, og ellers hele relationen flyttes til siden. Metoden `GetAdjacentTable` returnerer den tabel som er modsat en given tabel. Således får men sluttabellen, hvis med giver den starttabellen.

Når relationen ønskes gentegnet har `ERConnector` et event (`Redraw`) som tegnemodellen kan lytte på, og opdatere dens rute. Eventet startes ved at kalde metoden `RedrawERConnector`.

Interface: `IDiagramElement`
For at tilgå alle elementer der har med diagrammet at gøre på en ensartet måde, implementerer de begge interfacet `IDiagramElement`. Udover en række properties der har med størrelse og placering at gøre, er der en metode (`GetRects`), som returnerer en liste over rektangler, som elementet dækker. For en `UC_TableBox` er det bare et enkelt rektangel, men for `ERConnector`s er der et rektangel for hvert linjestykke(segment) på en relation. Disse rektangler bruges til at afgøre om de overlapper hinanden.

En anden metode er `MoveElement`, som kan bruges til at flytte elementet relativt i forhold til den nuværende placering. Denne metode bruger jeg, hvis jeg vil flytte diagrammet på canvas.

3.2.6. Centrering af diagram og fordeling
Tabellerne i datamodellen kan nogle gange være delt op i flere diagrammer. Dette har jeg løst ved at tegnemodellen lægge alle elementer, der relatere til en gruppe tabeller i en liste. Denne liste bruger `UC_ERdiagram` til at fordele grupperne. Efterfølgende har jeg en metode der centrer diagrammet på canvaset. Den udvider også canvaset hvis der ikke er plads nok.

3.2.7. Flytning af elementer
Jeg har valgt at man kun kan flytter tabellerne rundt. Relationerne mellem dem tegnes bagefter op automatisk. Således er det kun `UC_TableBox,` der skal kunne flyttes.

Selve implementeringen af flytning er i `UC_ERdiagram` og bygger på et eksempel af Josh Smith (Smith, 2006). Rent praktisk er flytningen delt op i 3 faser.

1. Når musen trykkes ned (`DiagramCanvasPreviewMouseLeftButtonDown`)
Når musen trykkes ned et sted på canvaset, kaldes denne metode. Hvis der er trykket på en tabel gemmes denne, som det element der flyttes på. For at tegnemodellerne også ved at der trykkes på en tabel, og kan reagere på dette, kaldes metoden `StartMove,` som findes i interfacet `IDrawingModel`. Denne metode har elementet og en liste over berørte relationer som parameter.

2. Musen flyttes (`DiagramCanvasPreviewMouseMove`)
Denne metode kaldes når musen flyttes. Hvis det er mens at musen er trykket ned flyttes elementet. Her skal der også tages højde for, at canvaset kan være skaleret, og derfor skal flytningen af elementet omregnes i forhold til størrelsesforholdet.

Der er to måder at fortælle tegnemodellen, at noget er blevet ændret. Den ene er at kalde `RedrawERConnector` i alle berørte `ERConnector` objekter som så efterfølgende starter eventet `Redraw`.

Den anden er at metoden `OnMove` i tegnemodellen kaldes, igen med elementet der flyttes og de berørte relationer som parametre.

3. Musen slippes (`DiagramCanvasPreviewMouseUp`)
Når musen slippes, fjernes referencen til det element, der er blevet flyttet, og igen kaldes der en metode (`AfterMove`) i tegnemodellen.

3.2.8. Værktøjslinjen
I værktøjslinjen er der kommandoer til at styre UC'en. Som en del af testmiljøet er der en dropdown med de tegnemodeller som man kan tegne diagrammet med. Så er der en zoom boks, hvor man kan zoome ud og ind. Til sidst er der nogle muligheder for at eksportere diagrammet til grafikfiler.

3.2.9. Styring af skalering
Som nævnt ovenfor har jeg lavet muligheden for at kunne skalere diagrammet, men det skal også kunne betjenes. Dette gøres via en dropdown på værktøjslinjen hvor man kan vælge den ønskede skalering. Valgmulighederne ligger mellem 12,5% op til 1000%. Her kunne det for brugervenligheden have været smart at man kunne trykke f.eks. "Ctrl"-tasten ned og man så scrollede på musen. Men dette har jeg ikke implementeret i denne udgave.

3.2.10. Eksport til grafikfiler
I FK7 skal diagrammet gemmes i en grafikfil, igen kommer WPF til stor hjælp her med klassen RenderTargetBitmap som generer et bitmap af stor set alle synlige elementer i WPF. I dette tilfælde vil vi gerne lave et bitmap af vores canvas. Da grafikken i WPF kan skaleres kan billedet nemt laves i højre kvalitet end det normalt ses på skærmen. Jeg har valgt at give brugeren mulighed for at eksportere til to kvaliteter af grafikfiler. En i 96 dpi, det vil sig at billedfilen vist i 100% vil se ud som i programmet. Jeg har også valgt at man kan eksporter til en høj kvalitet på 300 dpi som vil giv et pænere resultat hvis man printer grafikfilen.

RenderTargetBitmap skal som parameter have den endelige størrelse på grafikfilen, den kvalitet det skal gemmes i, samt information om pixeltypen. Når dette er sat renderer jeg mit canvas med disse indstillinger. Dette objekt kan jeg nu gemme i en fil med den ønskede encoding. Jeg har valgt både at give mulighed for PNG og TIFF filer og forskellen er primært hvilken encoding jeg bruger. For TIFF filer har jeg dog eksplicit valgt "LZW" som komprimering, så den følger kravet til TIFF filer i arkiveringsversioner. Når bitmappet er encoded, skrives den ned i en fil. Jeg har valgt at fjerne den gradient der er i baggrunden på skærmen inden jeg gemmer billedet.

I denne testapplikation har jeg valgt at den bare gemmer billederne i en undermappe til input XML filerne kaldet Exports, med oplysninger om tidspunkt for genereingen.

3.3. Tegning af diagrammet

At tegne diagrammet op er hovedproblemet i dette projekt. Det kan gøres på flere simple måder, som måske ikke er så effektive, men som virker for simple databaser. Udfordringen med at tegne diagrammerne kan deles op i 3 hovedområde. Først og fremmest hvordan tabellerne skal placeres i forhold til hinanden, dernæst hvordan stregerne mellem dem skal tegnes. Og da man også skal kunne flytte på tabellerne efter de er tegnet, skal stregerne tegnes op igen så de passe til den nye placering.

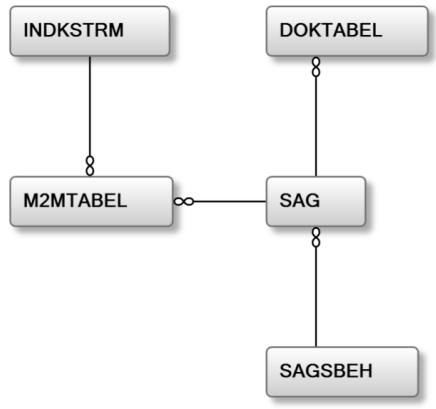
Jeg vil prøve at løse problemet på forskellige måder. Nogle modeller har jeg implementeret, andre har jeg enten ikke implementeret, eller kun dele af metoden. Jeg vil i begge tilfælde beskrive hvordan de kan implementeres.

3.3.1. Tegnemodel 1

Implementeret i DrawingModel1

Den første model jeg har skrevet er ret simpel. Først sorteres alle tabellerne efter hvor mange relationer tabellen har. Den med flest relationer behandles først. Her tælles jeg det samlede antal relationer med, altså både indgående og udgående relationer.

Til at tegne diagrammet bruges en rekursiv metode (`DrawTableWidthNeighborTable`), som parameter tager den tabel der skal tegnes (en `DiagramTable`), et x og y koordinat hvor tabellen skal placeres, samt en retning. Retningen bekriver i hvilken retningen den tidligere tabel er placeret.



Figur 5: Tegning af simpel database med tegnemodel 1

Når tabellen er tegnet, fjernes den fra en liste over de tabeller der skal tegnes. Så gennemløbes alle relationerne, som tabellen har, hvis de tabeller relationerne peger på, ikke allerede er tegnet, beregnes positionen, og metoden kalder sig selv igen. Næste tabel placeres i en 90 graders vinkel med uret i forhold den forgående. I Figur 5 se en simpel database tegnet op. "SAG" er den tabel med flest relationer og tegnes først. Herefter tegnes "DOKTABEL" ovenfor. "M2MTABEL" tegnes til venstre og "INDKSTRM" i en 90 gradres vinkel. Hvis der havde været en relation mere til "M2MTABEL" havde den være placeret til venstre.

Når alle tabeller er tegnet op, gennemløbes alle tabellerne igen og deres relationer, men denne gang kun udgående relationer, altså hvor tabellen har en fremmednøgle der peger på en anden tabel. Hvis tabeller er placeret således, at de er lige over eller lige ved siden af hinanden, tegnes der en enkelt lodret eller vandret streg mellem tabellerne. Hvis tabellerne ikke er placeret på denne måde tegnes stregen således at den starter på enten den venstre eller højre side og ænder så i toppen eller bunden af den anden tabel, med et knæk undervejs.

Når en tabel flyttes, kaldes metoden `RedrawERConnector` i alle de relationerne som er forbundet til tabellen. Den kalder et event og tegnemodellen gentegner relationen som tidligere beskrevet

Denne model fungere fint på meget små databaser, men meget hurtigt vil der komme problemer med at tabeller og streger ligge oven i hinanden.

Forbedringsmuligheder

En forbedring til denne model kunne være at ændre på hvordan tabellerne placeres i forhold til den foregående. Der kunne også testes for om tabeller og streger, ligger over hinanden, for så efterfølgende at flytte dem.

3.3.2. Tegnemodel 1b

Implementeret i DrawingModel1b

Denne model er en videreudvikling af model 1, og fungerer grundlæggende på samme måde. Men her er der justeret på, hvor næste tabel placeres. Som første valg placeres en tabel i samme retning som den foregående, og efterfølgende til siderne.

Test for overlap

Et gentagende problem ved at tegne på denne måde er at tabellerne og relationerne kommer til at ligge helt eller delvist over hinanden. Jeg har delt problemet op i forskellige typer af overlap.

- `UC_TableBox` over `UC_TableBox` (rød)
- `UC_TableBox` over `ERConnector` (`UC_TabelBox`: blå, `ERConnector`: blå)
- `ERConnector` krydser `ERConnector` (grøn)
- `ERConnector` ligger over en anden `ERConnector` et stykke af connectoren (lilla)
- To `ERConnector` ligger helt over hinanden (rød)

Som beskrevet i 3.2.5, implementer jeg interfacet `IDiagramElement`, som har en metode, der hedder `GetRects`. Den returnerer en liste af `Rect` objekter. Structen `Rect` (`System.Windows.Rect`) kan holde information om et rektangel, placering og størrelse. I denne sammenhæng er metoden `Intersect` interessant.

Når jeg vil vide om et `UC_TableBox` eller `ERConnector` overlappes af et andet element, kalder jeg metoden `IntersectWith` med elementet som parameter. Jeg gennemgår nu alle elementer på canvaset, og hvis det er af den type jeg tester for, testes de to elementers `Rect`s mod hinanden. Metoden returnerer en enum passende til en af ovenstående situationer.

I denne model tester jeg kun for overlap, men jeg flytter ikke noget. Men jeg farver stregerne og tabellernes kanter i forhold til, om der er overlap. Farverne kan ses i listen ovenfor.

Forbedringsmuligheder

Denne model vil helt klar blive bedre, hvis den også handler på baggrund af testen for overlap. Desuden kan måden diagrammet tegnes på også optimeres noget endnu. Modellen vil heller ikke være særlig overskuelig ved databaser med mange tabeller.

3.3.3. Tegnemodel 2

Implementeret i DrawingModel2

Igen bygger jeg videre på den foregående model tegnemodel 1b. Det første problem denne model løser, er forholdet til tabeller som overlapper hinanden. Når jeg prøver at tegne en tabel, testes der, om den ligger oven på en anden tabel. Så længe det er tilfældet flyttes tabellen, retningen er afhængig af hvilken retning, den tidligere tabel befandt sig i. Er det vertikalt flyttes tabellen horisontalt og omvendt. Denne løsning sikrer, at tabellerne ikke ligger over hinanden, men det er ikke sikkert, at diagrammet efterfølgende særlig logisk eller overskuelig. Man kan risikere, at enkelte af tabellerne kommer langt fra deres rette sammenhæng.

Da jeg i tegnemodel 1b testede for muligt overlap returnerede metoden bare den første type af overlap. Dette er ikke tilstrækkeligt, hvis man vil vide, om en bestemt type overlap opstår. Derfor har jeg udviklet klassen `Intersection`. Klassen initialiseres når `Draw` kaldes i tegnemodellen. Nå der så ønskes testet for et overlap kaldes metoden `Intersection.Detect`, med et `IDiagramElement` som parameter. Metoden returnerer true, hvis der findes et overlap. Efterfølgende kan en liste med `IntersectionElements` hentes i `IntersectedElements`. Klassen `IntersectionElements` har to properties en med typen af overlap, samt det element der er overlap med. Metoden `HasIntersectionOfType` kan bruges til at teste for om en bestemt type af overlap forekommer.

Streger gennem tabeller

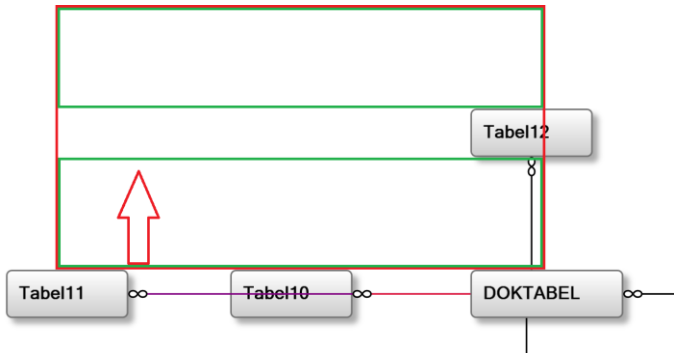
Det andet problem jeg prøver at løse i denne model er når en `ERConnetor` der ligger over en `UC_TabelBox`.

Figur 6: Eksempel på en streg gennem en UC_TableBox

Problemet kan opstå i flere situationer, men den situation jeg prøve at løse her, er når en tabel findes placeret på række med andre tabeller og skal forbindes til en tabel på den anden side af en anden tabel. Eksemplet ses i Figur 6. Den tidligere løsning tegner, som man kan se, en lige linje gennem den eller de mellemliggende tabeller. Når jeg har prøvet at tegne den lige linje, og testet og fundet frem til at den overlapper, skal jeg finde en anden vej udenom "Tabel10". Da "Tabel11" og "DOKTABEL" ligger på helt lige linje med hinanden har jeg defineret at jeg vil forsøge at tegne en vej ovenover "Tabel10". Først vil jeg undersøge hvor der er plads i området.

Til at finde ledigt område bruger jeg metoden `FindSpace`. Metoden tager 4-6 parametre. De første to `from` og `to` definer det interval, som skal undersøges. I dette tilfælde er der vandret, som der skal søges i. `start` er der, hvor søgningen skal starte, og sidst er den retning, der søges i. Her er det `Directions.North`. `FreeSpace` beregner et `Rect` for hvilket område der skal søges i. Den retning der ikke er defineret på forhånd, forlænges til 30 pixels udover kanten af diagrammet. Den 5 parameter som er valgfri, er muligheden for at tilføje en liste over `ERConnectors`, som skal ignoreres i søgningen. Det vil i dette tilfælde sige, dem der er forbundet med den tabel, vi flytter på. Den sidste parameter giver mulighed for at begrænse området til en bestemt afstand. Når området er beregnet, gennemløbes alle elementer på canvaset for de elementer der findes inden for området, og efterfølgende gemmes de i en liste. Her bruges elementernes `Rects` som `GetRect` returnerer. De elementer der måtte være i `ignoreList`, tilføjes ikke i listen.

`FreeSpace` returnerer en liste af `Inteval` objekter. `Inteval` er en generisk klasse som jeg har skrevet. Den har to properties start og end. Datatypen på intervallet er ikke defineret på forhånd, og sættes i <> efter navnet, derfor er den generisk. I dette tilfælde skal jeg bruge datatypen `double`. Så `FreeSpace` returnerer `List<Interval<double>>`.



Figur 7: Det røde område er det områder der søges i. De grønne områder er hvor den ikke har fundet nogle

Når der ikke findes nogle elementer i området, returneres hele området. Ellers kaldes metoden `DetectFreeArea`, den får det `Rect` som definerer området der søges i, retningen vi vil have det frie områder i (vandret/lodret), samt de elementer der findes i området.

`DetectFreeArea` starter med at skabe en liste til intervaller. Nu gennemløbes alle de elementer der befinder sig i området enkeltvis. Først søges der i listen efter det interval, hvor elementet starter, og ligeledes findes det interval, hvor elementet slutter. Hvis både starten og slutningen ligger i det samme interval, oprettes et nyt interval der starter hvor elementet slutter, og slutter hvor det oprindelige interval sluttede. Det oprindelige interval opdateres med slutposition i elementets startposition. I Figur 7 kan man se at området er blevet delt i to intervaller. Hvis ikke det er det samme interval, fjernes alle intervaller, der ligger mellem elementets start og slut. Og de intervaller, som start og slut ligger i, opdateres.

Når `FindSpace` returnerer listen med intervaller, kan jeg finde det sted, hvor der er frit til at den vandrette segment af relationen kan tegnes. I Figur 7 har jeg markeret, hvilke intervaller der er frie.

Denne løsning er ikke fuldt dækkende. For nok er det vandrette segment af relationen frit, men de lodrette segmenter kan forsat godt overlappe andre elementer. Der kunne søges i området over en tabel for at finde en forbindelse op til det vandrette stykke. Dette har jeg ikke implementeret i dette projekt. Med den nuværende forbedring vil relationen ikke ligge over tabellen direkte, og eventuelle parallel streger vil ligge ved siden af hinanden.

Ovenstående løsning sørger for, at relationer mellem tabeller, som ligger lodret eller vandret ud for hinanden, bliver forbundet af en anden vej. Denne løsning virker ikke for

tabeller, der er forbundet af en simpel knækket forbindelse. Her vil det forsat forekomme overlapninger. Dette problem vil kunne løses på en lignende måde. Dette har jeg dog valgt ikke at prioritere at implementerer.

Fordeling af streger

En anden problemstilling der kan nævnes er, at relationer forbundet til en tabel, ikke bliver fornuftigt fordelt. I mange tilfælde vil relationerne ligge oven i hinanden. Til at løse dette, har jeg skrevet metoden `AdjustConnectors`. Som parameter tager den en `UC_TableBox`. Det første den gør, er at finde de `ERConnector`s, der findes på en given side af tabellen. Disse deles så op i tre grupper, ud fra hvilken retning de er orienteret i. Efterfølgende sorteres de tre gruppers relationer efter, hvor langt der er fra startpunkt til slutpunkt.

Nu beregnes den indbyrdes afstand mellem de enkelte relationer således, at de fordels jævnt. For eksemplets skyld tager jeg udgangspunkt i en overside. Først placeres de relationer, som peger til venstre, så dem der peger lige op, og til sidst dem der peger til højre. Det skal dog bemærkes, at den sidste gruppe tages i omvendt rækkefølge. Således at dem der er længst, bliver justeret først.

Denne metode er heller ikke helt færdig implementeret, da den kun virker for oversiden. Desuden mangler den at tage højde for at tabeller placeret over den aktuelle tabel, kan være forskudt i forhold til hinanden, og at stregen således ikke ramme den anden tabel. Dette problem kan løses på forskellige måder. Hvis der findes god plads på siden, kan fordelingen ændres således, at de kommer til at passe. Men i de tilfælde hvor der er mange streger på én side, vil det måske være bedre at lave en `ERConnector` med et eller måske to knæk på. I sidste ende kunne man også flytte lidt på tabellerne, men dette kan efterfølgende give andre problemer. Når metoden implementeres, så den virker for alle sider, skal der findes en måde, så to tabeller ikke modarbejder hinanden.

Gentegning af streger

I de tidligere modeller gentegnes relationerne, når en tabel flyttes. Dette fungerer også udmærket, men da stregerne tit vil ligger oven i hinanden, og der ikke tages hensyn til dette, opstår der ikke nye problemer, når tabellen flyttes. Men efter at jeg placerer streger i forhold til hvor der er fri plads, får jeg problemer med den metode, jeg tidligere har brugt. Hvis jeg forsat tegnede dem op en efter en, når jeg flyttede en tabel, vil de flyttes længere og længere væk i nogle tilfælde. Problemet opstår ved, at når der søges efter et fri område, vil metoden finde det udenfor de allerede tegnede streger, det samme vil den næste og den næste igen. På den måde rykker den længere og længere væk. Jeg har derfor valgt at ændre den måde der gentegnes på. I stedet for at tegne den fra bunden igen, vil jeg prøve at rette på det oprindeligt tegnede layout. Når tabellen flyttes, forsøger jeg om en af segmenterne i relationen kan forlænges eller forkortes således, at den passer. Hvis dette ikke er muligt,

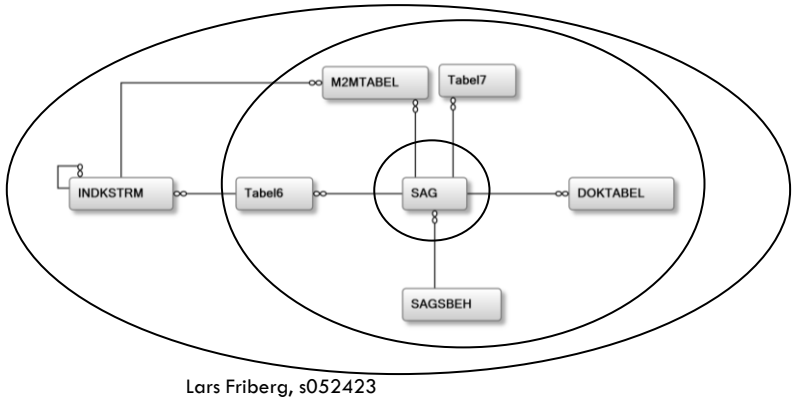
må der tegnes en ny fra bunden. Fordelen ved denne metode er, at den holder sin plads, og forbindelsen flytter ikke pludselig et andet sted hen. Metoden her til at opdaterer relationerne virker til et vist punkt efter hensigten, men mangler muligheden for at forlænge på flere segmenter på samme tid.

3.3.4. Tegnemodel 3
Ikke implementeret model

I denne tegnemodel, vil jeg komme med et andet bud på hvordan tabellerne kan placeres. I de foregående modeller er placeres en tabel, og en relation følges. Så tegnes denne tabel og den forsætter til man ikke kan komme længere. Efterfølgende vendes tilbage til tidligere tabeller, og deres relationer følges. Denne måde at gennemgå datamodellen er meget lig algoritmen Depth-First Search(DFS), som beskrives nærmere i næste afsnit (Tegnemodel 4).

I denne tegnemodel vil jeg bruge en anden algoritme hedder kaldes Breadth-First Search eller bare BFS, og bygger på en algoritme Moore forslog i 1957 (Even 2012, side 12) Da jeg først sent i projektet, har tænkt denne algoritme ind som en løsning på dette problem, har jeg ikke haft den fornødne tid til at implementere den. Men jeg vil beskrive princippet i den og hvordan den kunne implementeres.

Det grundlæggende i denne algoritme er en kø, lad os kalde den Q, og en liste som vi kalder T, som er til alle de tabeller, der er tegnet. Begge kan indeholder tabeller (`DiagramTable`). Der vælges en starttabel, dette kunne som i foregående modeller være den med flest relationer. Denne tabel placeres om den første, og repræsenterer den første ring eller bølge af tabeller. Starttabellen lægges i T og i Q. Nu køres en løkke som forsætter indtil der ikke er flere tabeller i køen (Q). Det første der gøres i løkken er, at fjerne den første tabel i Q, første gang er det jo starttabellen. Nu gennemgås alle relationer til denne tabel og hvis de tabeller som peges på, ikke allerede er tegnet og dermed er i T, tegnes de rundt om tabellen som er udgangspunktet. Den nye tabel lægges i listen T, og i køen Q. Således vender algoritmen tilbage til tabellen, når alle med samme afstand til starttabellen er tegnet. I Figur 8kan ses, hvordan en datamodel kunne være tegnet med denne algoritme. Jeg har markeret "bølgerne" med cirkler.



Figur 8: Datamode der kunne være tegnet med BFS algoritmen

3.3.5. Tegnemodel 4

Delvist implementeret i DrawingModel4

I denne model vil jeg betragte min datamodel som en graf, og bruge nogle algoritmer fra grafteorien. Således vil jeg også bruge begreber fra grafteorien. Så i stedet for at snakke om tabeller vil jeg kalde dem knuder, og relationerne mellem tabeller, kalder jeg kanter. Jeg arbejder med kanterne i klassen `GraphEdge`.

Det første jeg vil gøre, er at dele grafen op i uadskillelige komponenter, samt finde de deleknuder som deler grafen op. Den algoritme jeg bruger er, som udgangspunkt Hopcroft-Tarjan's udgave af Depth-First Search (DFS) med nogle tilføjelser (Even 2012, side 52ff).

I Hopcroft-Tarjan's udgave af DFS er omdrejningspunktet en `while` løkke, som kører så længe at knuden, den til enhver tid er i, har flere nye kanter eller at den er i en knude som har en faderknude (dvs. at det ikke er startknuden).

Jeg har valgt at bygge min implementering af DFS op omkring en rekursiv metode (DFS), samt `foreach` løkker gennem knudernes kanter.

Før jeg kalder `DFS` første gang, vælger jeg hvilken knude jeg starter ved. Dette kan i princippen være hvilken som helst knude. Jeg vælger at tage den med flest kanter (som i de foregående tegnemodeller). Knuden får nummer 1, og har jo ikke nogen faderknude. Til at holde styr på hvilket nummer den er kommet til, gemmes det næste nummer altid i `dfsIncrementNumber`. Som en del af denne algoritme, lægges hver ny knude i en `Stack`, hvor den sidst besøgte knude ligger øverst. Nu kaldes `DFS` for første gang, med den valgte startknude. Det første `DFS` gør, er at fjerne knuden fra listen over alle knuder. Så genereres en liste over alle nye kanter. Til at holde styr på om en kant har været brugt, har den en property kaldet `DFSNew`. Som udgangspunkt er alle kanter nye.

Med en `foreach` løkke gennemløbes alle kanter en efter en. Først testes for om kanten er ny, hvis den er det markers den som brugt. Dernæst identificeres den knude som er i den anden ende af kanten, med metoden `GetAdjacentTable`. Nu testes for om knuden er i listen over knuder som ikke er besøgt. Hvis dette er tilfældet lægges knuden på stacken, knudenummeret og faderknude opdateres. Til brug for at finde deleknuderne, gemmer jeg også et lavpunkt (`DFSLowpoint` i `DiagramTable`). I første omgang sættes dette til det aktuelle knudenummer. Knudenummeret tælles en op. Nu kaldes `DFS` igen med den nye knude. Da `DFS` kaldes igen inden vi komme tilbage til `foreach` løkken kommer den igennem alle der er forbundet gennem denne knude, inden den forsætter med de andre kanter.

Når alle kanter i en knude er gennemløbet, går algoritmen videre til at identificere om det er en deleknude. Dette gøres kun hvis knuden den er i, ikke er startknuden. Hvis denne knudes lavpunkt er større end faderknuden nummer har vi en uadskillelig komponent. Men først skal vi have gemt deleknuden. Deleknuden er faderknuden til den aktuelle knude, og gemmes i listen `seperatingTables.` Men hvis faderknuden er startknuden, og denne ikke har flere nye kanter er der ingen deleknude, men komponenten er den sidste i grafen.

Nu hentes alle knuder på stacken ud, ned til og med den aktuelle knude og de gemmes i en liste. Disse knuder samt faderknuden, udgør en uadskillelig komponent. Og listen gemmes i listen `nonSeparableComps.`

Hvis denne knudes lavpunkt, ikke var større end faderknuden, som der blev testet for før, beregnes et nyt lavpunkt for faderknuden, som værende det lavest af den aktuelle knudes lavpunkt og faderknudens lavpunkt.

Når `DFS` er kørt færdig, og er tilbage hvor den startede, har den være forbi alle knuder og kanter. Listen `nonSeparableComps` indeholder lister over knuder der er en del af uadskillelige komponenter og listen `seperatingTables` har alle deleknuderne. Hvis grafen er uadskillelig vil der kun være et komponent, og ingen deleknuder.

Ovenstående er implementeret i `DrawingModel3`, men på baggrund af prioritering mellem andre dele af projektet, i forhold til tiden, har jeg valgt ikke at implementere nedenstående, men jeg vil beskriver hvordan diagrammet vil kunne tegnes op. Nu vil jeg prøve at beskrive hvordan jeg vil tegne tabellerne op. Så når jeg konkret beskriver hvordan de skal tegnes, bruger jeg tabeller og relationer.

De uadskillelige komponenter kan nu bruges når jeg skal tegne grafen op. En måde kan være at starte med et komponent, f.eks. det største og tegne dette. Dette kunne f.eks. være med metoden fra tegnemodel 2. Når den så er tegnet op, kan man så gennemgå alle tabeller som er deleknuder (listet i `seperatingTables`), og tilføje tabellerne fra de komponenter. Således kommer disse tabeller til at ligge i umiddelbare nærhed af hinanden. Udfordringen ved denne løsning vil være at få plads til tabellerne, da tegnemodellen jo ikke har taget højde for at der skal flere tabeller ind. Den plads kan skabes på flere måder.

- Man kan starte med at tegne tabellerne med større afstand. I de foregående modeller har jeg brugt en horisontal afstand på 150 pixels og 100 pixels vertikalt. Dette kunne man forøge og dermed få mere luft.

- En anden metode kunne være at flytte tabeller, hvis der ikke er plads. Man kunne forstille sig at man finder tabeller som ligger i en bestemt retning, og flytter dem et stykke væk.

Behovet for plads afhænger jo også meget af hvor mange tabeller der skal tilføjes. I de tilfælde hvor en uadskillelig komponent består af to tabeller, hvoraf den ene jo er deletabellen, skal der jo ikke så meget plads til. Men måske er deletabellen, eller den anden tabel, deletabel for flere komponenter. Her kunne man evt. finde de deletabeller, som optræder flere gange i listen og danne nogle større komponenter. De vil ikke længere være uadskillelige, men de vil stadig kun have en tabel til fælles med andre komponenter. Når man har disse komponenter, kan man prøve beregne hvor meget plads der skal bruges. Enten kan man lede efter et frit område, med metoderne `freeSpace` eller `DetectFreeArea`, eller man kan prøve at skabe mere plads som beskrevet ovenfor.

Som jeg skrev ovenfor, har jeg ikke forsøgt at implementere disse metoder, da jeg har prioriteret andre dele af projektet. Men for at få en fornemmelse af, hvad algoritmen kan, har jeg brugt tegnemodel 2 vise dette. I tegnemodel 4a tegnes hele datamodellen op, og efterfølgende markeret de tabeller, der er deletabeller, med en grøn baggrund.

I tegnemodel 4b tegnes kun det største, af de uadskillelige komponenter op. Igen farves deleknude grønne. Jeg `DFS` på datamodellen og sender den største af komponenterne videre til tegnemodel 2. Efterfølgende markerer jeg alle deletabeller. I metoderne, `StartMove`, `OnMove` og `AfterMove`, kalder jeg de tilsvarende metoder i tegnemodel 2. Således er alt tegning og gentegning overladt til den tegnemodel, og det er kun farvningen denne model ændre.

3.3.6. Tegnemodel 5

Som i tegnemodel 4 vil jeg forsætte med at se datamodellen som en graf.

I dette afsnit kommer jeg ikke med en færdig løsning, men med nogle input til hvordan, grafen kan behandles, således at en fornuftig tegning af grafen er mulig..

En forudsætning for en række grafalgoritmer er, at den er uadskillelig, som jeg har beskrevet i tegnemodel 4. Desuden skal grafen også være uden parallelle kanter, samt uden "self loop", hvor en knude er direkte forbundet til sig selv. Til denne del har jeg implementeret klassen `SimplifyGraph`. Klassen tager en graf løber den igennem og finder de steder, hvor der er flere parallelle kanter mellem to knuder. Den fjerner også kanter som forbinder en knude med sig selv. Alle de kanter der fjernes bliver gemt i en liste, som så efterfølgende kan tilføjes grafen igen, når den ønskede algoritme er udført. Selve rekonstruktionen er ikke en del af `SimplifyGraph` klassen. Da jeg ikke har implementeret

de algoritmer, som skulle bruge den simplificerede graf, har jeg valgt ikke at lave denne rekonstruktion.

En plan graf

En graf er plan, når den kan tegnes så ingen af kanterne krydser hinanden. Det vil være optimalt hvis en graf er plan, at jeg også kunne tegne den sådan. Dette er dog ikke helt enkelt. Men først vil jeg kigge lidt på algoritmer til at teste om en graf er plan. Hopcroft og Tarjan kom i 1974 som de første med en algoritme, der kunne teste om en graf var plan, i lineær tid (Even 2012, s. 168). Deres algoritme bygger på, at de tilføjer stier, en af gangen. En sti (path) er en vej gennem grafen. De starter deres algoritme med et simpelt kredsløb, dvs. en sti gennem grafen hvor knuder og kanter kun indgår en gang, og som ender i den samme knude som den startede. Efterfølgende tilføjer de stier mellem to knuder i grafen. Alle stier tilføjes således at de ikke krydser tidligere kanter. Nogle gange, er man nødt til at flytte og vende hele stykke af grafen for at få det til at lykkes. Hvis det ikke er muligt at tilføje en sti, er grafen ikke plan. Den anden algoritme går ud på at tilføje en knude afgangen. Knudens kanter tegnes også, men hvis den knude de ender ved ikke er tilføjet endnu, efterlades de uden forbindelse i enden. Også i denne algoritme er det til tider nødvendigt at flytte rundt på allerede placerede knuder. Begge algoritmer er temmelig komplicerede at implementere, og det er ikke lykkedes mig at implementere dem inden for tidsrammen af dette projekt og samtidig løse andre af opgaverne. Således er jeg heller ikke kommet dertil, hvor jeg kan tegne en plan graf ved hjælp af grafteorien.

4. Test

I dette kapitel vil jeg gennemgå mine tegnemodeller og vise hvordan de hver især kan tegne forskellige datamodeller op. Grundet nogle tekniske problemer mangler billederne i testafsnittet.

4.1. Test af tegnemodeller

Jeg vil i denne test, prøve alle datamodellerne på alle tegnemodeller. Jeg vi gennemgå resultat af en række af testene her i rapporten. Mens nogle er placeret kan ses som bilag. Grunden til at de er lagt som bilag er, enten at de ikke viser noget nyt i forhold til tidligere test.

Jeg har udvalgt fire datamodeller jeg vil bruge til min test. Nedenfor er de kort beskrevet.

Test datamodel 1

Den første datamodel, er taget fra Statens Arkivers eksempelafleveringversion (AVID.SA.18000) Det er en database til et sags og dokumenthåndteringssystem. Det er en lille simpel database, som består af 5 tabeller.

Test datamodel 2

Denne datamodel er en udvidelse af datamodel 1, der er tilføjet en række ekstra tabeller

Test datamodel 3

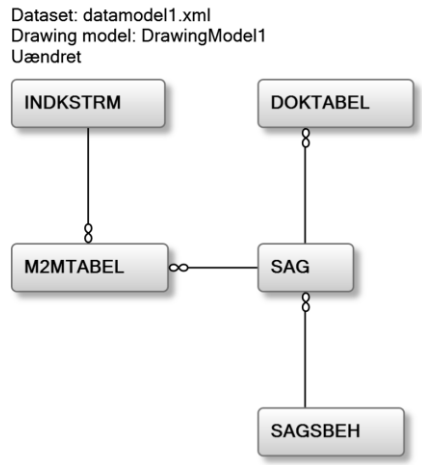
Her har jeg konstrueret en lille datamodel med 4 tabeller. Den har flere tilfælde af parallelle relationer og den har et self loop. Denne er primært lavet til test af klassen `SimplifyGraph`.

Test datamodel 4

Den sidste datamodel er taget fra en arkiveringsversion som er afleveret til Statens Arkiver. Den har 47 tabeller, og et hav af relationer mellem dem.

I testen vil jeg henvise til datamodellen som DM1, DM2 osv. Ligeledes henviser jeg til tegnemodellerne som TM1, TM1b, osv.

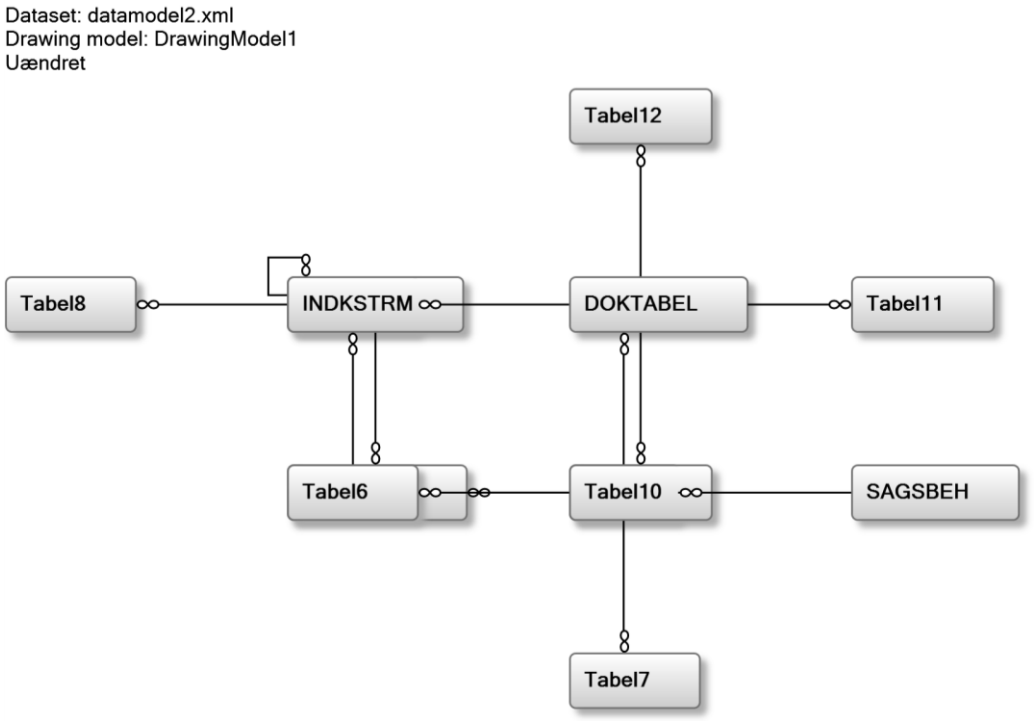
4.1.1. Tegnemodel 1



Figur 9 Tegnemodel 1 med datamodel 1

Optegning af diagrammet

Denne tegnemetode er meget simpel, og har derfor også meget kraftige begrænsninger. Som det kan ses på Figur 9, virker den fint til denne lille simple datamodel (DM1). Den starter med den tabel med flest relationer, og tegner de øvrige tabeller uden om. Da den drejer tabellerne for hver gang kommer den ret hurtigt til kort. Dette kan ses på en mere omfattende DM 2. Her kommer tabeller og relationer til at ligge over hinanden. En datamodel som DM 3 er også svær at aflæse da relationerne ligger oven i hinanden. Den sidste datamodel er helt uoverskuelig at aflæse, da mængden af tabeller resultere i at de ligger over hinanden. Det skal dog bemærkes at den fint sørger for at tegne tabeller, som ikke er forbundet til diagrammet, fint op i bunden af diagrammet.

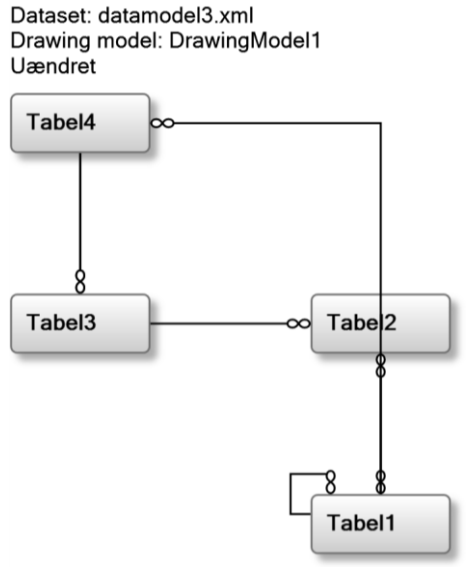


Figur 10: Tegnemodel 1 med datamode 2

Gentegning af relationer

Tegnemodellen har de to, grundlæggende måder at tegne relationer op på. Direkte forbindelse og med et enkelt knæk. Ved flytning, gentegnes relationerne fra bunden. Dette fungerer efter hensigten, men da der jo ikke testes for overlap, tegnes de henover tabeller og andre relationer.

Ved DM 3 skal det dog bemærkes at man nu kan se begge relationer mellem "Tabel1" og "Tabel2". Da de er defineret til at gå fra side til top/bund. Den dobbelte relation mellem "Tabel3" og "Tabel4" ligger forsat over hinanden.



Testkonklusion

Denne simple tegnemetode virker efter hensigten og tegner fint diagrammet op, så længe at der er tale som meget små og enkelte datamodeller, men den har sine tydelige begrænsninger.

4.1.2. Tegnemodel 1b

Denne tegnemodel er en videreudvikling af tegnemodel 1, men den adskiller sig på 2 punkter. Tabellerne tegnes i samme rækkefølge som i tegnemodel 1, men breder sig lidt mere, da den som førstevalg altid fortsætter i den retning den kommer fra. Dette kan f.eks. ses i fig x, hvor DM1 bliver mere aflang. Datamodel 2 tegnes rimeligt pænt op, med kun få tabeller der overlapper. På DM 4, er der fortsat rigtig mange tabeller der ligger over hinanden, men de er alligevel spredt mere ud.

Den store forskel i denne tegnemodel er at tabeller og relationer har fået farver hvis de overlapper. Således kan man nu se, hvor der ligger tabeller og relationer over hinanden. Dog er der to problemer ved denne løsning. For det første får elementet farve efter den først element søgning finder der overlapper, der kan godt være flere typer overlap. Den anden, og nok større fejl opstår når der er flere grupper af elementer i diagrammet. Dette ses f.eks. i DM4, hvor der er to tabeller der ikke har relationer til resten. Disse er farvet, men har ingen overlap. Fejlen opstå da tabellerne oprindeligt bliver tegnet hen over resten af diagrammet, og flyttes så efterfølgende væk fra hinanden. Selve testen for overlap køres før

grupperne af tabeller flyttes. Den rigtige løsning vil være at metoden (`IntersectWith`), der tester kan skelne de forskellige grupper af tabeller. Farvningen bliver korrekt når tabellen flyttes manuelt. Det skal også bemærkes at tabellerne er markeret røde, for overlap med andre tabeller, men ikke blå for overlap for med relationer. Dette skyldes igen tidspunktet for hvornår de farves, som er før relationerne tegnes.

Gentegning af relationer

Dette er ikke lavet om i forhold til tegnemodel 1, dog kan man jo nu se hvis der er relationer over hinanden.

Testkonklusion

Denne tegnemodel udfører det, som den er lavet til. Den er ikke lavet til at kunne håndtere mange tabeller. Dog er der ikke helt styr på farvelægningen.

4.1.3. Tegnemodel 2

Som udgang tegnes tabellerne op på samme måde, som i tegnemodel 1b. Men hvor TM1b kun farvede tabellerne ved overlap, flytter tegnemodel 2 tabellerne ved overlap. DM1 tegnes præcis som med tegnemodel 1b, så den kigger vi ikke nærmere på. Datamodel 2 kan vi se forskellen. "Tabel11" som i TM1b ligger oven på "Tabel10" er nu flyttet op over. Her skal dog bemærkes at relationen mellem "DOKTABEL" og "Tabel11", fejlagtigt er farvet grøn. Dette skyldes metoden `AdjustConnectors,` som køres efter farvelægningen har flyttet relationen op til ”Tabel12”, således at de ikke krydser hinanden mere.

Udover de relationer, som de foregående tegnemodeller har, er der også en ny i denne tegnemode. Den fremkommer når en direkte forbindelse krydser andre tabeller, så føres relationen udenom. i fig x ses dette i relationen mellem ”SAG” og ”M2MTABEL”. Der søges efter et frit område ved siden af tabellerne. Det findes først uden for ”SAGSBEH”. Således er den midterste del fri for tabeller og relationer, men der tages ikke højde for om der er tabeller på vejen derud.

I denne tegnemodel er metoden `AdjustConnectors` en forbedring. Metoden fordeler relationerne på en side af en tabel. Den er kun implementeret til oversiden af tabellerne, og det kan også ses her at den ikke er helt optimal. Ved tabellen ”INDKSTRM”, mangler der at tages højde for self loops. Hvor dette ændre, fejlagtigt. I denne datamodel (DM2) er det ikke så udpræget, men i DM4 kan man virkelig se hvordan den virker. Relationerne til tabellerne (på oversiden) fordeles jævnt. Dette kan ses i fig x hvor jeg kigger på en enkelt tabel. Der er dog også plads til forbedring her. hvis man føler op til første knæk, kan man se at rækkefølgen ikke er helt optimal, da to relationer mellem de samme to tabeller, krydser hinanden. Men alligevel er dette langt bedre end tidligere. En anden fejl er, hvor der er relationer mellem to tabeller uden nogen form for knæk. Her betyder den aktuelle

implementering, at nogle relationer ender ude i ingenting. Og så mangler metoden jo også at blive implementeret til de øvrige sider.

Gentegning af relationer

I denne tegnemodel er der også ændret på hvordan relationerne opdateres, når tabeller flyttes. Som beskrevet i tidligere, prøver jeg at opdatere relationen, frem for at gentegne det fra bunden. Metoden virker desværre ikke helt igennem endnu. Da den ikke er helt i stand til at korte sig af på det rette måde, og for at den var optimal, skulle den også kunne forlænge de enkelte segmenter af relationen. Desuden er der en decideret fejl som til tider opstår, hvor der blive tegnet forkerte streger. På fig xx kan ses DM2, hvor en relationerne, opføre sig på en noget uventet måde. Fejlen kan nok kobles sammen med de problemer modellen har med at korte relationerne af.

Testkonklusion

Alt i alt der mange gode funktioner i denne tegnemodel, men ikke alle virker helt efter hensigten. Mange ting kunne løses ved videre udvikling, men er valgt fra, til fordel for andre dele af projektet.

4.1.4. Tegnemodel 3

Denne model er som tidligere beskrevet ikke implementer så jeg vil ikke lave en test af den her.

4.1.5. Tegnemodel 4

Denne tegnemodel er implementeringen af Depth-First Search algoritmen, som finder uadskillelige komponenter. Selve tegningen af tabeller m.m. tager TM 2 sig af. Da anden del af denne tegnemodel, hvor komponenterne tegnes sammen igen, ikke er implementeret, vil jeg prøve at vise hvordan algoritmen virker på flere forskellige måder. Derfor er tegnemodel 4 delt op i en 4a, hvor hele diagrammet tegnes, men deletabellerne farves grønne. TM4b tegner den største af de uadskillelige komponenter op. Igen med markering af deletabeller markeret grønne.

I fig x ses DM1 tegnet med TM4a. Denne datamodel vil med DFS blive delt op i 4 komponenter af hver to tabeller. I DM2 kan man ligeledes se hvilke tabeller, der er deletabeller. Af en eller anden grund, som jeg ikke har undersøgt nærmere, tegnes diagrammet på en anden måde end når den tegnes med TM2 alene. DM3 er i sig selv uadskillelig og derfor sker der ingen ændringer med den. Med DM4 ser diagrammet næsten ud som før, udover at deletabellerne er grønne. Det er ikke umiddelbart muligt at se hvad, der er uadskillelige komponenter, men det er her TM4b kommer ind i billedet. Den tegner kun den største komponent og udelader resten. De øvrige DM vil jeg ikke kigge nærmere på her, se evt. bilag x.

Testkonklusion

Denne tegnemodel tegner ikke så meget i sig selv, da den bruger

4.1.6. Tegnemodel 5

Denne tegnemodel er ikke rigtigt implementeret, bortset for klassen `SimplifyGraph` som jeg vil teste her. Jeg vil bygge den ovenpå TM4b, da det viser tydeligst hvordan den virker. DM1 og DM2 kigger jeg ikke på her, se bilag x. Men DM3 er lavet primært til denne test. Den består af to gange to parallelle relationer samt en enkelt self loop. Disse relationer går SimplifyGraph i og fjerne. I DM4 bliver diagrammet også simplere at se på. Men som beskrevet i afsnit 3.3.6, er det ikke i forhold til visningen, at jeg laver denne reduktion af relationer, men som en forstadie til algoritmer til at teste for om grafen er plan og evt., tegne den plant.

Testkonklusion

På baggrund af de to diagrammer i fig x og y vil jeg sige at simplifyGraph gør det den er lavet til.

4.2. Samlet Testkonklusion

Generelt virker de enkelte tegnemetoder godt, set i forhold til hvor meget der implementeret. Der er enkelte ting der ikke virker efter hensigten. Det drejer sig om farvelægning af tabeller og relationer, ved overlap. Der er også problemer med gentegning af relationer ved flytning af tabeller. Men alt i alt virker tingen rimelig god, i forhold til hvor meget der implementeret af de enkelte modeller.

5. Konklusion

Jeg har i dette projekt udviklet en User Control, som tager en datamodel som input og projicere den op på en brugergrænseflade. Dels har jeg udviklet den grundlæggende funktionalitet, som muliggør en egentlig tegning af diagrammet. Generelt er det lykkedes at løse de problemer jeg valgte i min afgrænsning (afsnit 2.3). Det skal dog bemærkes at det er en kørende prototype. Så implementeringerne er ikke fuldstendige Helt overordnet kan jeg tegne datamodellerne op som ER-diagrammer. Når det kommer til at kunne flytte tabellerne efterfølgende er dette også implementeret, men en optimal løsning for, hvordan relationerne skal gentegnes, har jeg ikke fået færdigimplementeret. Ligeledes er det mulig at skalere diagrammet og eksportere det til grafikfiler.

En afgrænsning vil også medføre ting man ikke vælger at løse da jeg har begrænset til. Og der kunne gøres meget mere.

Samlet set er det lykkedes at udvikle den User Control som var målet med dette projekt var. Min vurdering er at med videre udvikling kan den blive et rigtigt nyttigt værktøj.

Jeg har i projektet været omkring følgende ting.

- Jeg har lavet overvejelser om udformning af diagrammet
- Jeg har implementeret en del af de løsninger jeg er kommet frem til
- Prototypen er i stand til at tegne diagrammer med forskellige tegnemodeller og med flere datamodeller.
- Man kan flytte tabeller rundt på canvaset
- Diagrammet kan skaleres
- Diagrammet kan eksportres til grafikfiler

6. Kildehenvisninger

Wikipedia artikel 1: "Entity–relationship model" Wikipedia, the Free Encyclopedia, 13. September 2013. http://en.wikipedia.org/w/index.php?title=Entity%E2%80%93relationship_model&oldid=571907469

Chen, Peter Pin-shan. "The Entity-Relationship Model: Toward a Unified View of Data." ACM Transactions on Database Systems 1 (1976): 9–36. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.123.1085

Even, Shimon. Graph Algorithms. Edited by Guy Even. 2nd ed. Cambridge: Cambridge University Press, 2011. http://ebooks.cambridge.org/ebook.jsf?bid=CBO9781139015165

Smith, Josh "Dragging Elements in a Canvas", 2. Sep. 2006, http://www.codeproject.com/Articles/15354/Dragging-Elements-in-a-Canvas

7. Bilag

Bilag 1: Systembeskrivelse

Systembeskrivelse af afsluttende praktikantopgave udført for Statens Arkiver efter afsluttet praktikophold.

Systembeskrivelse af diagrammeringsværktøjet "Diana"

Overordnede ønsker

Der ønskes udarbejdet en applikation, der i en grafisk visning kan opmærke en relationel database på en brugergrænseflade.

Applikationen ønskes udført i programmeringssproget C# og skal kunne afvikles på en Microsoft Windows platform. Applikationen skal udføres således, at den kan etableres i en "User Control", hvor formålet med denne udformning er en højere grad af genanvendelighed.

Applikationen skal også kunne afvikles i en selvstændig udgave.

Den færdige applikation med tilhørende kildekode, tilhører i fællesskab Lars Friberg og Statens Arkiver, som hver især frit kan disponere over applikationen.

Funktionalitet

- Applikationen skal via struktureret metadata fra en database eller anden metadatakilde, automatisk kunne projektere databasens entiteter samt kardinalitet i en grafisk visning i form af et ER- lignende diagrammeringsformat.
- Databasens entiteter skal i visningen fremstå med tabelnavn. Der skal være mulighed for at kunne se tabellens feltnavne, samt hvilke felter der udgør henholdsvis primær- og fremmednøgler. Denne visning skal kunne til- og fravælges.
- Alle entiteter skal frit kunne bevæges inden for den overordnede brugergrænseflades afgrænsning.

Mulig funktionalitet

- Ved et klik på en enkelt entitet skal data fra den bagvedliggende tabel, der repræsenterer den pågældende entitet, kunne projekteres ud i en selvstændig brugergrænseflade i en struktureret visning, eventuel i form af et dataGridView. Andre projekteringsformer kan komme på tale.
- Det kunne være ønskeligt, at brugeren kan udvælge en eller flere entiteter, som efterfølgende projekteres ud på en brugergrænseflade.
- En funktionalitet der procentvis op- eller nedskalerer brugergrænseflades entiteter.
- En funktionalitet der kan udskrive og persistere brugergrænsefladens entiteter med tilhørende kardinalitet i et grafisk billedformat.
- En mulig persistering af en aktuel visning af entiteter, således at visningen automatisk kan genoptages
- At visningen af entiteter forgår efter en optimeret model, så mulige overlapninger af entiteter så vidt muligt undgås.
- En funktion i forbindelse med visningen af entiteterne der kan til- og frakoble visning af datatyper.
- En modulær til- og frakobling af forskellige typer af databaseservere

Kodestil og dokumentation af kode

- Al kode ønskes dokumenteret ud fra de beskrivelser, som er beskrevet af Microsoft i relation til det valgte udviklingssprog C#[7].
- Ligeledes ønskes programklasser udviklet med så løs kobling som muligt.
- Det vil være ønskeligt, hvis der i høj grad gøres brug af erklærede interface mellem klasser, jævnfør Microsofts anbefalinger.[8]
- Indstillinger til brug for applikationen persisteres i en konfigurationsfil jævnfør Microsofts vejledning.[9]
- Der gøres brug af fejlhåndteringsrutiner således at mulige fejl så vidt muligt rapporteres til bruger på en struktureret facon.[10]

18. januar 2013

Tommy Balle
Specialkonsulent

---

[7] http://msdn.microsoft.com/en-us/library/vstudio/b2s063f7.aspx
[8] http://msdn.microsoft.com/en-us/library/vstudio/ms173156.aspx
[9] http://msdn.microsoft.com/en-us/library/aa730869(v=vs.80).aspx
[10] http://support.microsoft.com/kb/816157

Bilag 2: Kildekode

Kildekode og hele projektet er uploadedt som bilag.

Der er også en kørende version af prototypen.