

# Een collaboratief platform voor het efficiënt reconstrueren van fresco's

Nicolas Hillegeer

Thesis voorgedragen tot het  
behalen van de graad van Master  
in de ingenieurswetenschappen:  
computerwetenschappen

**Promotor:**

Prof. dr. ir. P. Dutré

**Assessoren:**

Ir. N/A

N/A

**Begeleider:**

Dr. ir. B.J. Brown

© Copyright K.U.Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail `info@cs.kuleuven.be`.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

# Voorwoord

Dit is mijn dankwoord om iedereen te danken die mij bezig gehouden heeft. Hierbij dank ik mijn promotor, mijn begeleider en de voltallige jury. Ook mijn familie heeft mij erg gesteund natuurlijk.

*Nicolas Hillegeer*

# Inhoudsopgave

<b>Voorwoord</b>	<b>i</b>
<b>Samenvatting</b>	<b>iv</b>
<b>Lijst van figuren en tabellen</b>	<b>v</b>
<b>1 Inleiding</b>	<b>1</b>
<b>2 Overzicht van het thera project</b>	<b>3</b>
2.1 De opdelingen van het thera project . . . . .	3
2.2 Reconstructie, de bestaande oplossingen . . . . .	4
<b>3 Doelen &amp; Motivatie</b>	<b>9</b>
3.1 Aspecten waarop de thesis tracht te verbeteren . . . . .	10
<b>4 Ontwerp van het project</b>	<b>15</b>
4.1 De grote lijnen . . . . .	15
4.2 Modulariteit . . . . .	19
4.3 Dataopslag, synchronizatie, ... . . . .	20
4.4 Gebruiksgemak / visuele interface . . . . .	21
4.5 Het beheer van de data . . . . .	22
4.6 Visualisatie, een manier om met de data te werken . . . . .	22
4.7 Integratie in thera project . . . . .	22
4.8 Uitbreidbaarheid . . . . .	22
<b>5 Bespreking van het database ontwerp</b>	<b>23</b>
5.1 Vereisten . . . . .	23
5.2 Het oude systeem: XML-bestanden . . . . .	23
5.3 Alternatieven . . . . .	25
5.4 SQL of niet . . . . .	25
5.5 Eerste iteratie . . . . .	28
5.6 Verschillende database systemen . . . . .	29
5.7 Externe database traag, interne database snel . . . . .	29
5.8 Slimme client of slimme server? . . . . .	30
5.9 Benchmarking . . . . .	31
<b>6 Modules</b>	<b>33</b>
6.1 MatchTileView . . . . .	33
6.2 Proof of concept: GraphView . . . . .	33

<b>7 Toekomstig werk</b>	<b>35</b>
<b>8 Besluit</b>	<b>37</b>
<b>A Numerieke resultaten</b>	<b>41</b>
<b>Bibliografie</b>	<b>43</b>

# Samenvatting

Deze abstract moet nog geschreven worden.

# Lijst van figuren en tabellen

## Lijst van figuren

2.1	Een voorbeeld Griphos tafelblad, 1 voorgesteld paar en 2 aparte fragmenten zijn reeds ingeladen . . . . .	4
2.2	Griphos kan gebruikt worden om de posities van fragmenten in hun opslagplaats te onthouden en vervolgens snel terug te vinden. Het kan ook een foto van de bak waarin ze liggen onder het virtuele tafelblad projecteren. De brokstukken die buiten de bak staan weergegeven zijn verplaatst geweest naar een andere bak. (afbeelding met toestemming gebruikt uit [1]) . . . . .	6
2.3	Browsematches in werking, de balken boven de paren duiden een validatie door de gebruiker aan. Rood betekent bijvoorbeeld “dit voorstel is zeker niet juist” . . . . .	7
3.1	Het huidige proces met de aanvullingen van de thesis in het blauw . . . .	10
4.1	Het abstracte model van de applicatie, links staat de <i>View/Controller</i> en rechts het <i>Model</i> . De controller stuurt een verzoek naar het model voor een bepaalde (sub)set van de data — al dan niet gesorteerd — en het model antwoordt met alle paren die voldoen aan de criteria . . . . .	16
4.2	Basis sorteer- en filteroperaties worden via de gebruikersinterface blootgesteld . . . . .	18
4.3	De manier van weergeven uit Browsematches werd gekopieerd naar het nieuwe platform, met uitbreidingen . . . . .	19
4.4	Een vereenvoudigde kijk op de componenten van de visualisatielaag, het hoofdscherm en het model. De componenten in het wit behoren tot de rest van het thera project en zijn niet gemaakt als deel van dit thesisproject.	20

## Lijst van tabellen

5.1	Meting van de tijden die elk programma nodig heeft om een collectie paren in te laden . . . . .	24
-----	---	----





# Hoofdstuk 1

## Inleiding

Het reconstrueren van fresco's waarvan in opgravingen fragmenten gevonden worden is een moeilijke taak. Men kan het vergelijken met het oplossen van een enorme puzzel waarvan de stukken arbitraire vormen hebben, de meesten hun originele kleur zijn verloren en er vele anderen ontbreken. Daarbovenop komt nog eens dat er heel wat stukken over de eeuwen heen een zekere vorm van slijtage hebben ondervonden, waardoor ze niet meer perfect op elkaar passen en dus confirmatie nog moeilijker maken.

[afbeelding van enkele opgegraven stukken]

De stukken van de rand van het fresco met recht afgelijnde kanten of die nog een voldoende zichtbaar geometrisch patroon bevatten, zijn in vergelijking met de anderen eenvoudig met elkaar te verbinden. De overige fragmenten die minder informatie bevatten zijn echter een nachtmerrie om aan elkaar te koppelen, gezien het menselijke visuele systeem niet gemaakt is om dat soort grillige randen te vergelijken en aan elkaar te zetten. Om deze reden is het normaalgezien noodzakelijk om vele mogelijke kandidaten visueel te onderscheiden en ze over elkaar te laten glijden om te zien of het past en waar precies.

In deze context situeert zich het **thera**<sup>1</sup> project, dat probeert om het werk van de archeoloog gemakkelijker te maken door middel van een software platform. De redenering achter het project is dat wat voor een mens repititief en tijdsabsorberend zou zijn, geautomatiseerd zou kunnen worden m.b.v. een computer. Dergelijk systeem werd in 2007 aan de *Princeton* universiteit in Amerika geconcipieerd. Sindsdien is er door verschillende onderzoekers van over de hele wereld aan gewerkt om alle noodzakelijke delen te ontwerpen, te implementeren en te integreren. [vermeld hoofdonderzoekers]. De werking van het systeem wordt in het volgende hoofdstuk nader toegelicht.

Deze thesis draait rond het maken van een uitbreiding op één van deze delen. De

---

<sup>1</sup>Thera is de oude naam voor het huidige griekse eiland genaamd Santorini, waar het project voor het eerst in de praktijk werd toegepast.

uitbreiding moet de gebruikers van het systeem in staat stellen om de beschikbare data op nieuwe manieren te gebruiken, te visualiseren, aan te passen en te delen met medeonderzoekers.

// TODO: vorige stuk gaat niet zozeer over samenwerking meer, herschrijf dit mss? De gangbare methode van manueel puzzelen van vóór het thera project leende zich wel tot een zekere vorm van samenwerking, op voorwaarde dat men op dezelfde site aanwezig was. Verder moest alle betreffende informatie van commentaren tot classificaties in een centrale plek bewaard worden zodat iedereen erbij kon. Dit ging van memobriefjes op de fragmenten zelf tot manueel aangepaste elektronische documenten. Dankzij het thera systeem kon deze manier van werken veranderd worden, alle relevante kennis kan elektronisch opgeslagen worden. Er was echter tot op heden nog geen gemakkelijke manier om deze kennis met andere archeologen te delen en al zeker niet om de bevindingen van verschillende onderzoekers te combineren. Dat is het kernprobleem dat deze thesis zal proberen uit de wereld te helpen. Kortom: de nodige aanpassingen en nieuwe implementaties zullen gedaan worden om het oude systeem collaboratief te maken, zodat het beheren, delen en combineren van data gemakkelijk en intuïtief wordt.

## Hoofdstuk 2

# Overzicht van het thera project

Zoals eerder vermeld bouwt deze thesis verder op een reeds bestaand project. Om ze beter te kunnen situeren in het geheel is het handig om eerst even het thera project te bekijken om te kunnen zien waar dit project in past.

### 2.1 De opdelingen van het thera project

Ruwweg gezien kan het reconstrueren van een fresco met behulp van de computer opgedeeld worden in 3 fasen.

**Acquisitie** Alle gevonden fragmenten worden ingescand met behulp van 3D en 2D-scanapparatuur om zo een virtueel model van elk stuk te bouwen, zie [2].

**Identificatie** Vervolgens worden deze virtuele fragmenten aan een zogenaamde *matcher* gegeven, die voor elk fragmentenpaar gaat kijken of ze mogelijk op elkaar passen. Er zijn verschillende types ontwikkeld. Eén ervan (genaamd *RibbonMatcher* [2]) kijkt enkel naar de randen van de brokstukken om te zien of ze in elkaar sluiten. Hierdoor is het in staat om passende fragmenten te vinden die voor mensen moeilijk te vinden zijn wegens te weinig distinctieve attributen zoals tekeningen. Het onderzoek gaat verder, in 2010 werd er een nieuw type ontwikkeld dat zijn analyse baseert op een combinatie van verschillende eigenschappen, zoals de sporen die een borstel kan nalaten bij het kleuren van een fresco of zelfs de beoordeling van de eerder genoemde *RibbonMatcher* [3].

**Classificatie & Reconstructie** De derde en laatste stap bestaat uit het classificeren van wat de vorige stappen produceren. Elk voorgesteld paar moet gecontroleerd worden op validiteit. Verschillende statussen kunnen toegekend worden aan een paar, zoals: *geconfirmeerd*, *misschien*, *nee*, *in conflict met een ander paar*, et cetera. De *matcher* produceert in de regel zeer veel paren van brokstukken, waarvan slechts een klein deel correct kan zijn. [BARBARA] De drempel voor het beslissen van wat een paar kan zijn en wat niet wordt in de vorige stap zo laag ingesteld omdat men wil vermijden dat twee fragmenten die toch passen genegeerd worden. Anders gezegd, de kost van “*false negatives*”

wordt hoog ingeschat. Uiteindelijk zal hieruit een gereconstrueerde fresco moeten ontstaan.

## 2.2 Reconstructie, de bestaande oplossingen

Het thesiproject besproken in deze tekst tracht de reeds bestaande hulpmiddelen van de reconstructiefase aan te vullen. Hievoor werden reeds programma's ontwikkeld, namelijk *Griphos* en *Browsematches*. Hierop volgt een bondige bespreking van beide programma's, wat ze kunnen en niet kunnen, en eventuele voor- en nadelen. Deze factoren hebben een belangrijke rol gespeeld bij het bepalen van de richting van deze thesis.

### 2.2.1 Griphos

Griphos was de eerste applicatie gericht op het weergeven en beoordelen van de resultaten van de voorgaande stappen. Het werd ontworpen met het doel een centraal zenuwstelsel te zijn voor alle informatie en met deze uiteindelijk een (zo goed mogelijk) gereconstrueerd fresco te maken. Het is mogelijk om zowel aparte fragmenten als automatisch herkende fragmentparen op een virtueel tafelblad te plaatsen en te manipuleren (zie figuur 2.1). De idee achter een dergelijke voorstelling komt voort uit een rechtstreekse vertaling naar de computer van wat een archeoloog op een werkelijk tafelblad doet.



Figuur 2.1: Een voorbeeld Griphos tafelblad, 1 voorgesteld paar en 2 aparte fragmenten zijn reeds ingeladen

Griphos wordt echter geplaagd door een paar problemen: het is traag en biedt geen goede manier om de talloze gegenereerde paarvoorstellen na te kijken. De gebruikte metafoor van het virtuele tafelblad waarin gepuzzeld kan worden stelt een belangrijk deel van het reconstructieproces voor, maar schiet tekort als men de resultaten van automatische fragmentpaarontdekking op vloeiende wijze in het proces wil betrekken. Het probleem lijkt te zijn dat er te veel informatie is en Griphos geen goede manier biedt om door de bomen het bos te zien. De aanzienlijke traagheid van sommige delen van het programma komen vooral voort uit de manier van dataopslag voor paren (XML bestanden) en de complexiteit van de visualisatie (afbeeldingen van hoge kwaliteit en/of gedetailleerde 3D-modellen). Dit zorgt soms voor een onaangename werkervaring, zelfs indien men er toch in slaagt de juiste fragmentparen te lokaliseren. Desalniettemin is het een krachtig programma dat veel functionaliteit biedt voor de detailinspectie van brokstukken.

Het heeft zijn nut al op verschillende vlakken bewezen, behalve detailinspectie is het bijvoorbeeld ook in staat om de posities van fragmenten in een bak te onthouden. Dit betekent een grote snelheidswinst wanneer men bijvoorbeeld denkt een goed paar te hebben gevonden en men wil dit met echte fragmenten verifiëren. Als beide fragmenten in dezelfde bak liggen kan het correcte tafelblad ingeladen worden, Griphos kan vervolgens de gezochte fragmenten laten oplichten. Dit is veel efficiënter dan fragmenten opsporen door vormen te vergelijken, zeker omdat de brokstukken vaak moeilijk te onderscheiden zijn zoals te zien valt in figuur 2.2.

// te technisch voor dit hoofdstuk, verplaats naar iets later: [TODO: remove] Het eerste probleem wordt vooral veroorzaakt door trage datatoegang en te complexe visualisaties. Het programma slaagt alles betreffende paren op in een (enorm) XML bestand. Dit is nog doenbaar als men slechts een paar duizend voorstellen heeft maar ondervindt reeds snel onacceptabele vertragingen eens men er meer probeert in te laden. Ter voorbeeld, een kleine voorstellenverzameling van een bepaalde opgraving heeft er reeds 50000. Gekoppeld hieraan laadt Griphos ook steeds een hoge-kwaliteits afbeelding of zelfs een volledig 3D-model in voor elk paar. De combinatie van het gebruik van grote XML-bestanden om informatie over de paren in op te slaan, en het steeds inladen van hoge kwaliteits-afbeeldingen en 3D-modellen. om alle info in op te slaan. Dit is bijzonder inefficiënt en biedt ook niet voldoende mogelijkheden tot uitbreiding. Een ander deel is het gebrek aan zoekmogelijkheden binnen de voorstellen.

### 2.2.2 Browsematches

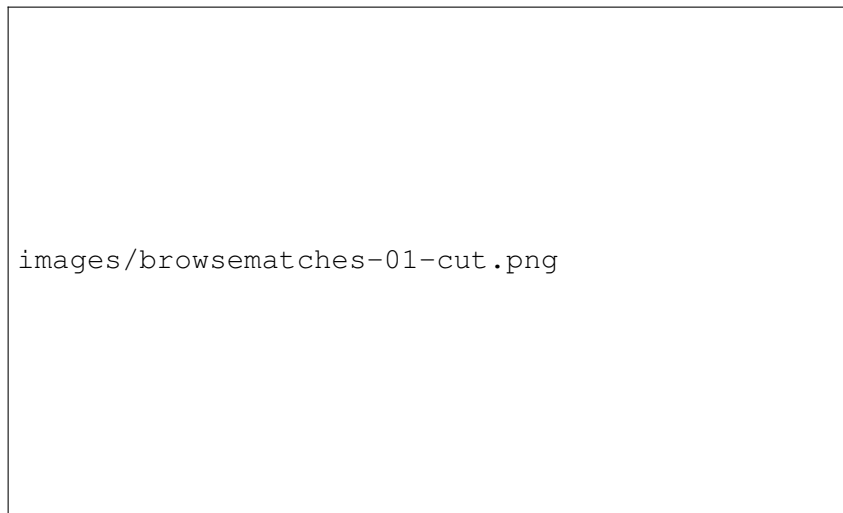
Een eerste prototype om de in Griphos ontbrekende delen aan te vullen, werd Browsematches genoemd. Het gebruikt eerst de visualisatiecapabiliteiten van Griphos om kleine afbeeldingen te nemen van elk bestaand paar met aan de linkerkant de doorsnede van hun raakvlak. Vervolgens toont het een scherm vol paren en kan men met de pijltoetsen navigeren tussen schermen.



Figuur 2.2: Griphos kan gebruikt worden om de posities van fragmenten in hun opslagplaats te onthouden en vervolgens snel terug te vinden. Het kan ook een foto van de bak waarin ze liggen onder het virtuele tafelblad projecteren. De brokstukken die buiten de bak staan weergegeven zijn verplaatst geweest naar een andere bak. (afbeelding met toestemming gebruikt uit [1])

Dit kleine programma groeide uit noodzaak: het valideren van paarvoorstellen is zo omslachtig met Griphos dat Browsematches in korte tijd werd gemaakt om het proces te stroomlijnen. Bij het begin van de thesis, om kennis te maken met het thera project, werd een verbinding gemaakt met Griphos zodat voorstellen die interessant waren in detail bestudeerd konden worden.

Browsematches erfde echter ook enkele van de nadelen van Griphos. Het inladen van de data is traag en vergt zelfs voor kleine datasets veel geheugen. Daarbij is het moeilijk om de informatie uit te breiden of deze gemakkelijk te combineren met de bevindingen van andere onderzoekers. Daarenboven is Browsematches een prototype en niet bedoeld voor algemeen gebruik. Het simpele maar succesvolle concept diende als inspiratie en basis voor deze thesis.



Figuur 2.3: Browsematches in werking, de balken boven de paren duiden een validatie door de gebruiker aan. Rood betekent bijvoorbeeld “dit voorstel is zeker niet juist”





## Hoofdstuk 3

# Doelen & Motivatie

Het hoofddoel van het thera project is om het reconstrueren van fresco's uit de oudheid zo gemakkelijk mogelijk te maken. Er moet een manier gevonden worden om een waardevolle contributie te maken aan het thera-ecosysteem zodat het zijn doel beter kan vervullen. Dit is onder andere mogelijk door delen die ontbreken aan de vorige oplossingen aan te maken of bestaande componenten veelzijdiger te maken.

Gezien de huidige stand van zaken besproken in hoofdstuk 2, valt het op dat er nog behoorlijk wat dingen kunnen toegevoegd worden op het gebied van informatie en het visualiseren ervan. De ongelooflijke hoeveelheid data die het thera project reeds heeft geproduceerd en blijft produceren kan op een betere manier behandeld worden zodat het ware potentieel ervan naar de oppervlakte komt. De laatste stap in het sterk geautomatiseerde virtuele reconstructieproces wordt gekenmerkt door een nood aan interactie met de mens die elk soort hulpmiddel moet krijgen om zo correct en snel mogelijke beslissingen te maken. Natuurlijk gaat er niets boven de feitelijke fragmenten fysisch vastnemen en ze aan elkaar te proberen zetten. Maar gezien dit een zeer tijdsroevende bezigheid is, moet dit zo veel mogelijk beperkt worden.

Er werd gesproken over Griphos en diens weinig ontwikkelde ondersteuning voor het behandelen van automatisch voorgestelde fragmentparen, alsook over de eerste poging om dit recht te trekken: Browsermatches. Deze thesis tracht de lijn van Browsermatches verder te zetten, vertrekkende van de goede ideeën en ervaring waaraan het zijn ontstaan te danken heeft. Dit houdt in dat de focus niet meer zozeer ligt op het actief puzzelen en brokstukken op een tafelblad plaatsen, maar eerder op het beoordelen van de omvangrijke verzameling voorstellen, waarvan slechts een zeer klein deel correct kan zijn. De hoop en verwachting is dat dit zeer kleine deel meteen ook een significante hoeveelheid van de werkelijk overblijvende paren voorstelt.

Daarmee valt te benadrukken dat dit project geen vervanging probeert te zijn van de bestaande software, het is eerder bedoeld als een complement. Merk op dat gevalideerde paren niet kunnen conflicteren en dus rechtstreeks in een groep op een tafelblad geplaatst kunnen worden. In de limiet, wanneer alle mogelijke

paren correct geclassificeerd zijn vloeit hieruit op natuurlijk wijze een (zo compleet mogelijk) fresco voort. Daarom valt de implementatie in de thesis te zien als een extra tussenstap, chronologisch vóór het plaatsen van de fragmenten op een tafelblad en na het uitvoeren van de herkenning algoritmen. Figuur 3.1 stelt dit visueel voor.



Figuur 3.1: Het huidige proces met de aanvullingen van de thesis in het blauw

Een van de meest fundamentele verschillen tussen de centrale filosofie van Griphos en die van dit project is dus dat terwijl bij Griphos de focus ligt op aparte geplaatste fragmenten, het thesisproject eerder de automatisch gevonden paren behandelt. Het is in Griphos wel degelijk mogelijk om voorstellen van de automatische paarherkenning in te laden maar de naald in de hooiberg vinden is moeilijk.

#### 3.1 Aspecten waarop de thesis tracht te verbeteren

Hieronder staan verschillende deelaspecten waarop dit thesisproject verbeteringen tracht te maken. Bij elk aspect staat beschreven wat de vereisten zijn waar het project aan zal moeten voldoen en eventueel een voorbeeldscenario. Men kan deze aspecten desgewenst onafhankelijk bekijken maar vaak steunen ze op elkaar om werkelijk tot hun recht te komen. Een visualisatiemethode heeft bijvoorbeeld een manier nodig om de data die het wil visualiseren te krijgen. Omgekeerd is een

databaseersysteem niet bijzonder nuttig zonder de mogelijk de data te manipuleren. Om de verbeteringen en ideeën te bundelen en te testen is er ook een applicatie gemaakt die dient om reeds een voorproef te geven van wat er mogelijk is met de ontwikkelde technologie.

#### 3.1.1 Integratie

Zoals eerder vermeld staat dit thesisproject niet alleen maar maakt het deel uit van een groter geheel. Binnen de grenzen van het mogelijke zou er moeten rekening gehouden worden met de integratie van de geschreven code met die van de andere onderzoekers. Dit verhoogt de kans dat het werk aanvaard wordt en ingang vindt in andere subprojecten.

#### 3.1.2 Collaboratie

Ideaal gezien zouden archeologen steeds toegang moeten krijgen tot hun project op eender welk moment vanop eender welke plaats. Dit kan een gedeelde collectie zijn die over het internet beschikbaar wordt gesteld, of een lokale kopie. Een mogelijk scenario hierbij is dat een ervaren archeoloog in de Verenigde Staten gevraagd wordt om zijn opinie te geven over de huidige stand van zaken (reeds geïdentificeerde correcte paren, moeilijke gevallen, ...). Alle aanpassingen en commentaren die hij maakt worden automatisch ingevoegd en centraal beschikbaar gesteld voor de onderzoekers ter plekke. Op termijn moet het bijvoorbeeld zelfs mogelijk worden om amateurs te laten kijken naar de voorstellen en hun beoordeling te gebruiken om de nog na te kijken voorstellen te rangschikken.

Van cruciaal belang is dat alle verzamelde data (zoals de classificatie van voorstellen) op een robuuste manier opgeslagen, gedeeld en geïncorporeerd kan worden. De toegang naar en manipulatie van deze informatie moet efficiënt zijn. De huidige oplossingen zijn hiervoor ontoereikend en traag, zoals besproken in hoofdstuk 2. De mogelijkheid van meerdere en zelfs lokale kopieën impliceert ook de nood om te kunnen synchroniseren<sup>1</sup>. Dit is ook nodig omdat er reeds verschillende huidige verzamelingen bestaan die eventueel in het nieuwe systeem geïmporteerd en gecombineerd moeten worden. Dergelijk systeem is ook nuttig voor het maken van (kleinere) pocketversies en in gebieden waar de internetconnectiviteit niet adequaat of onbestaande is. Belangrijk is dat er steeds een manier is om waardevolle data te combineren en aan te vullen zodat niets verloren gaat.

#### 3.1.3 Schalering

De bestaande oplossingen voor het valideren van voorgestelde paren werken noodgedwongen met een sterk gereduceerde verzameling. Eén van de redenen hiervoor is het gebruik van een groot XML bestand om alles in op te slaan. Deze techniek

---

<sup>1</sup>Naar het model van de zogenaamde *Distributed Version Control System (DCVS)* systemen zoals Git, Mercurial, ...

werkt goed voor bijvoorbeeld het opslaan van tafelbladen — waar er bijvoorbeeld 50 fragmenten en hun locaties moeten opgeslagen worden — maar schiet tekort voor paarvoorstellen. Een snelle rekensom geeft de reden aan: een laag aantal fragmenten voor een specifieke opgraving is bijvoorbeeld 1500. De meeste paarherkenners gaan alle brokstukken een keer met elk andere brokstuk vergelijken en zodoende het meest waarschijnlijke aankoppelingspunt vinden. Het is zelfs mogelijk dat een fragment meerdere keren een plausibel paar vormt met eenzelfde stuk. Naar onder afgerond komen uit deze stap reeds 2 miljoen configuraties gerold, de ene wat waarschijnlijker dan de andere. Dit nummer stijgt kwadratisch in het aantal fragmenten en zelfs met hogere drempels om volgens een bepaalde maatstaf paren automatisch te verwerpen stijgt het aantal configuraties snel.

#### 3.1.4 Gebruiksvriendelijkheid

De uiteindelijke gebruikers van de applicatie zijn niet de ontwikkelaars van het thera project zelf, maar de archeologen. Om deze reden is het belangrijk dat er rekening gehouden wordt met de noodzaak van een visueel aangename en intuïtieve gebruikservaring. Om deze intuïtiviteit te bereiken moet in het programma steeds de aandacht gevestigd blijven op hetgeen het belangrijkste en meest herkenbaar is: de paren en hun fragmenten. Bij voorkeur moet elke operatie gemakkelijk ontdekbaar zijn in de context waar ze kan gebruikt worden, met eventueel een woordje uitleg erbij.

Volgens vele studies op het gebied van gebruikersinterfaces [4, 5, 6] geraken gebruikers gefrustreerd vanaf een operatie een zekere tijd duurt. Deze frustratiedrempel hangt een af van de aard van de operatie (het resultaat), de frequentie waarmee die uitgevoerd wordt, of er visuele tekenen van voortgang zijn en of er tijdens het wachten (steeds) iets anders kan uitgevoerd worden. Om deze reden is het belangrijk dit aspect in acht te nemen bij het ontwikkelen van de applicatie. Dit is vooral zo omdat veel van de acties die mogelijk moeten zijn het potentieel hebben traag te lopen en dus de gebruiker te frustreren. Een algemene vaststelling: de tijd die een operatie mag innemen is omgekeerd evenredig met de frequentie waarmee deze operatie moet uitgevoerd worden. Deze regel in acht nemend is het duidelijk dat bijvoorbeeld het inladen van een scherm vol voorstellen zoals bij Browsematches, het veranderen van een attribuut van een paar, het filteren en sorteren en dergelijke meer acties zijn die met de grootst mogelijke snelheid moeten worden uitgevoerd. Hoe functioneel ook, een programma dat sloom reageert en elke computer op z'n knieën dwingt zal zo weinig mogelijk gebruikt worden [7].

// hoort mss bij [DESIGN] De gebruikersinterface van de oude programma's was goed en kon efficiënt gebruikt worden mits enige training. Maar voor het ondersteunen van collaboratief werken moeten er natuurlijk allerhande nieuwe operaties toegevoegd worden. Tijdens het implementatieproces werd het duidelijk dat de reeds bestaande code van het Browsematches niet uitbreidbaar genoeg was en die van Griphos te complex en belangrijk. Daardoor werd de beslissing genomen om de basisinterface van Browsematches over te nemen maar alle onderliggende code te herschrijven

zodat die uitbreidbaar zou zijn. Bij het opnieuw construeren van dit alles zijn er een aantal verbeteringen gebeurd die niet meteen te maken hebben met het collaboratie aspect maar wel met de workflow van het classificeren. Dit werd gedaan om het hele programma gebruiksvriendelijker en krachtiger te maken in functie van het hoofddoel van het project.

#### 3.1.5 Uitbreidbaarheid

Het samenstellen van werken uit de oudheid is een zeer vakkennis- en ervaringsintensief proces. Hoewel men luistert naar wat de archeologen hierover te vertellen hebben — wat ze graag zouden zien of kunnen doen — zijn er vele zaken die nu nog niet duidelijk zijn maar in de toekomst zeker aan het licht zullen komen. Dit kan bijvoorbeeld zijn omdat de onderzoekers in kwestie niet goed kunnen uitleggen waar ze naar kijken of hoe ze zoeken: na zovele jaren vertrouwen ze op hun moeilijk te definiëren intuïtie. Anderzijds is het vertalen van het proces om fresco's samen te stellen naar de computer nog niet zo vaak geprobeert in het verleden. Dit betekent dat een goede werkwijze in de realiteit misschien niet zonder meer de efficiëntste is als men de transitie naar virtueel reconstrueren maakt (zoals bij Griphos). Daarbovenop zijn er nog vele kansen om innovatieve nieuwe technieken aan te wenden die niet werkbaar zijn als men enkel over fysische fragmenten beschikt.

Het is dus onwaarschijnlijk dat het laatste woord over de ideale metafoor reeds gezegd is waardoor er een grote kans is dat het platform zal moeten vervangen of herbouwd worden als het niet met uitbreidbaarheid in gedachte ontworpen wordt. Er moeten moeiteloos nieuwe delen aan de applicatie en de onderliggende lagen kunnen toegevoegd worden om snel nieuwe ideeën te incorporeren. Dit alles moet best mogelijk zijn zonder de ervaring van gebruikers met oudere versies of andere gebruikersinterfaces te degraderen.

#### Data

De eerder besproken uitbreidbaarheid die nodig is manifesteert zich op het niveau van de data bijvoorbeeld bijvoorbeeld op deze manier: een onderzoeker vindt een nieuw algoritme om fragmentparen te rangschikken op “goedheid” of men wil informatie over het dikteverschil tussen twee fragmenten opslaan. Idealiter zouden deze zaken als een attribuut bij een paar moeten kunnen toegevoegd worden zodat elke gebruiker er op kan zoeken en sorteren zonder iets extra te hoeven doen.

Enkel data die op geen enkele manier om te vormen valt naar een attribuut van een enkel paar zal een speciale module vereisen om te kunnen gebruiken. Een vereiste is natuurlijk wel dat dit geen effect mag hebben op de delen van het platform die hier geen weet van hebben.

#### Visualisatie

Er zijn vele visualisatiemanieren denkbaar die elkaar kunnen aanvullen bij het volbrengen van het zoek- en classificeerproces. Zo stelt men zich bij de uitdrukking “fresco’s samenstellen” waarschijnlijk een grote puzzel voor waar men ten alle tijde het overzicht kan behouden en stukken proberen te passen.<sup>2</sup> Deze puzzel visualisatie is visueel aantrekkelijk en biedt het menselijke patroonherkenningsvermogen<sup>3</sup> de mogelijkheid om zich van zijn beste kant te laten zien. Echter, door de grote hoeveelheid aan fragmenten en dus mogelijke paren is het moeilijk om hier aan te beginnen. Maar, hoe meer reeds geconfirmeerde paren er zijn, hoe duidelijker het globale beeld kan worden. Dit staat toe om een overzicht te krijgen van de vooruitgang en gericht te zoeken naar stukken die nog ontbreken. Dit is een voorbeeld van een macro-perspectief.

Een alternatief is te beginnen door te kijken naar waar de computer wél goed in is: fragmenten aan elkaar passen en rangschikken naar kans/overeenkomst/et cetera. Door een gemakkelijk navigeerbare lijst op te stellen van alle voorstellen die de reconstructie algoritmes hebben gedaan, kunnen er snel op elkaar passende fragmenten geïdentificeerd worden. Dit kan men zien als een micro-perspectief of *bottom-up* manier om fresco’s te reconstrueren. Het is ook de aanpak die in Browsematches gebruikt wordt. Nadat men bijvoorbeeld met deze manier een deel acceptabele paren heeft geïdentificeerd, kunnen deze bijvoorbeeld weergegeven worden als een grote puzzel en dienen zij als beginpunt om verder te puzzelen. Dit maakt het globale beeld veel informatiever: stel dat duidelijke "gaten" ontstaan in een resem goede fragmentparen, dan kan er gericht gezocht worden naar een fragment dat erin past door te zoeken naar een fragment dat met elk van deze insluiters past (indien het gevonden werd bij de opgraving).

Kortom, het is duidelijk dat een visualisatie die in alle gevallen de meest geschikte is niet bestaat. De beschikbare informatie moet soms gewoon op andere manieren worden weergegeven. Om deze reden is het wenselijk om het mogelijk te maken snel nieuwe visualisaties in te bouwen die kunnen communiceren met andere delen van de applicatie en eventueel de informatie manipuleren.

---

<sup>2</sup>Het Griphos programma gaat uit van het idee van kleine beheersbare stukken van de reuzenpuzzel (elk tafelblad zou een verzameling kunnen zijn van stukken die gerelateerd waren, bijvoorbeeld door hun vindplaats)

<sup>3</sup>Iets waar computers de mens nog altijd niet in evenaren. Een ander therapeutisch subproject onderzoekt wel de mogelijkheid om zogenaamde ‘clusters’ te identificeren en te gebruiken voor reconstructie.

## Hoofdstuk 4

# Ontwerp van het project

Eenmaal de belangrijkste vereisten gekend zijn kan er een ontwerp opgetekend worden. Daarom even de grote lijnen die te concluderen vallen uit hoofdstuk 3:

- De data en het visuele aspect van de applicatie moeten zo ontkoppeld mogelijk zijn, elk moet apart uitbreidbaar zijn
- Zeer grote hoeveelheden data moeten vlot kunnen behandeld en genavigeerd worden: zoeken, filteren, sorteren, aanpassen, ...
- De data moet zowel centraal als decentraal toegankelijk zijn en synchronisatie toestaan
- Compatibiliteit met de reeds geschreven onderdelen van het project moet indien mogelijk bewaard blijven
- Om het ontwerp te verifiëren moet een werkend programma gemaakt worden, gebruiksvriendelijkheid, functionaliteit en snelheid zijn hierbij belangrijk

### 4.1 De grote lijnen

Van alle vereisten die rechtstreeks invloed kunnen uitoefenen op de architectuur, is de splitsing van de data en de visualisatie waarschijnlijk de meest fundamentele. Een beproefde aanpak om dit te realiseren is het ontwerppatroon genaamd *Model-View-Controller (MVC)*<sup>1</sup>.

Het doel is een ontwerp te maken waarin we een visualisatiemethode kunnen verbinden aan de juiste databron. Enkele zaken moeten echter nog vastgelegd worden, namelijk: wat is de basiseenheid van data? Waar zal een visualisatiemodule achter vragen? Zoals in figuur 3.1 te zien is, ligt de focus bij dit project op koppelingen van fragmenten in plaats van fragmenten op zich. Er zijn op het eerste zicht twee alternatieven om dit te modelleren. De eerste mogelijkheid is een soort van *MatchedFragment*

---

<sup>1</sup>Een algemeen overzicht kan men bijvoorbeeld bekomen op <http://en.wikipedia.org/wiki/Model-view-controller>

die een fragment beschrijft plus een lijst met alle fragmenten die er potentieel aan gekoppeld kunnen worden en op welke locatie. De tweede mogelijkheid is om elk paar een apart object te laten voorstellen (bvb. genaamd *FragmentPair*). Het eerste alternatief lijkt het voordeel te hebben dat het gemakkelijk is om na te kijken of een fragment reeds “bezet” is. Ook zou het dan mogelijk zijn om bijvoorbeeld een grafe op te stellen door van brokstuk naar brokstuk te springen. Echter, dit soort opstelling bevat op het eerste zicht veel redundantie, een fragment zal op die manier een verwijzing met attributen naar een fragment bevatten, en dit fragment zal op zijn beurt een identieke omgekeerde verbinding hebben. De redundantie vermijden en een verwijzing als een apart object voorstellen waar beide fragmenten naar kunnen verwijzen is eigenlijk niets anders dan de tweede optie (een *FragmentPair*). Op die manier wordt de situatie omgedraaid en kan een fragmentenpaar verwijzen naar de fragmenten die het opbouwen. Daarbij kan het probleem van hoe de “bezetting” van een object te weten te komen (alsook de grafe, zoals later zal aangetoond worden) opgelost worden door de vereiste zoekfunctionaliteit van het datamodel te benutten.

Eenmaal de basiseenheid van informatie gekozen is, valt de kern van de applicatie volgens MVC uit te beelden als in figuur 4.1.



Figuur 4.1: Het abstracte model van de applicatie, links staat de *View/Controller* en rechts het *Model*. De controller stuurt een verzoek naar het model voor een bepaalde (sub)set van de data — al dan niet gesorteerd — en het model antwoord met alle paren die voldoen aan de criteria



### 4.1.1 Verzoeken

#### Opvragen

Het concept achter het opvragen van fragmentparen is dat er steeds wordt begonnen vanuit de volledige beschikbare verzameling. Deze kan dan **gereduceerd** en **gesorteerd** worden.

Sorteren is eenvoudig en kan op eender welk attribuut in stijgende of dalende zin gebeuren. Filters gebruiken een simpele maar krachtige syntax die volledig gelijk is aan die van de *WHERE*-clausule van een SQL zin (voorbeeld: broncode 4.1). Geavanceerde gebruikers kunnen op deze manier zelf filters verzinnen, veelgebruikte filters en sorteeroperaties kunnen echter best als knoppen en invoervelden blootgesteld worden, zoals in figuur 4.2.

De voorwaarde die aan filters gesteld wordt is dat ze enkel attributen beschrijven die werkelijk bestaan, filters die naar een onbestaand attribuut refereren worden genegeerd.

Er kunnen meerdere filters op een model actief zijn, deze werken dan conjunctief. Op die manier kunnen meerdere modellen filters plaatsen zonder met elkaar in conflict te komen.

Broncode 4.1: Een voorbeeld van hoe een model kan gebruikt worden in een applicatie

```
IMatchModel *model = ...; // een model

model->sort("error", Qt::AscendingOrder); // alles moet gesorteerd worden op het "error"
    attribuut, van laag naar hoog
model->filter(
    "duplicates_filter",
    "duplicate = 0"
); // eis dat het "duplicaat"-attribuut van een paar gelijk is aan 0, wat betekent dat het geen
    duplicaten heeft
model->filter(
    "probability_filter",
    "probability > 0.8 OR volume < old_volume"
); // de probabilliteit moet groter zijn dan 0.8 of het volume moet kleiner zijn dan het oude
    volume
model->filter(
    "my_new_status_filter",
    "status IN (1,2)"
); // het paar moet reeds "goedgekeurd" of "waarschijnlijk goed" zijn

...

int max = model->size(); // de hoeveelheid paren die aan de filters voldoen

for (int i = 0; i < model->size(); ++i) {
    IFragmentPair &pair = model->get(i);

    // doe iets met het paar
```

}

---



Figuur 4.2: Basis sorteer- en filteroperaties worden via de gebruikersinterface blootgesteld

### Aanpassingen

Veranderingen aan de inhoud van de database Een verzoek...

Er zijn twee grote klassen Er kan gesorteerd worden op attributen en er kan gefilterd worden o

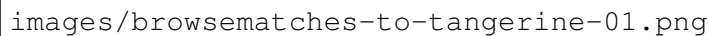
#### 4.1.2 Fragmentenparen

Het zoeken naar de gewenste verzameling van paren gebeurt via een verzoek en wordt aan het model overgelaten, maar wat kan een component doen met de paren die dan ter beschikking komen? Op zijn minst bestaat een paar uit twee fragmenten en een resem eigenschappen. Die eigenschappen of attributen kunnen gaan van een maatstaf voor de fout toegekend door de automatische herkenner, tot de validatie die een onderzoeker aan het voorstel heeft gegeven. De objecten die geproduceerd worden door de automatische paarherkenners van het theraproject voldoen aan deze twee voorwaarden. Er bestaan reeds veel componenten die hiervan gebruik maken dus dezelfde interface recycleren zou de integratie van het project een heel deel vergemakkelijken. Alle aanpassingen die zich niet kunnen beperken tot één paar kunnen via het model afgehandeld worden.

## 4.2 Modulariteit

Om het geheel uitbreidbaar te maken is een pluginsysteem gemaakt. Er is een hoofdapplicatie (codenaam “Tangerine”) die de connectie maakt met de databebeerlaag en verschillende visualisatieplugins kan laden. Het basissysteem zonder modules bestaat uit een manier om een fragmenten en paren-database in te laden en te kiezen welke module op te starten.

De eerste en meest uitgewerkte daarvan werd *MatchTileView* gedoopt. Het geeft de paren op dezelfde manier weer als het Browsematches prototype, maar is natuurlijk uitgebreid qua mogelijkheden. De bespreking van de toegevoegde functionaliteiten komt aan het bod in het hoofdstuk over modules.

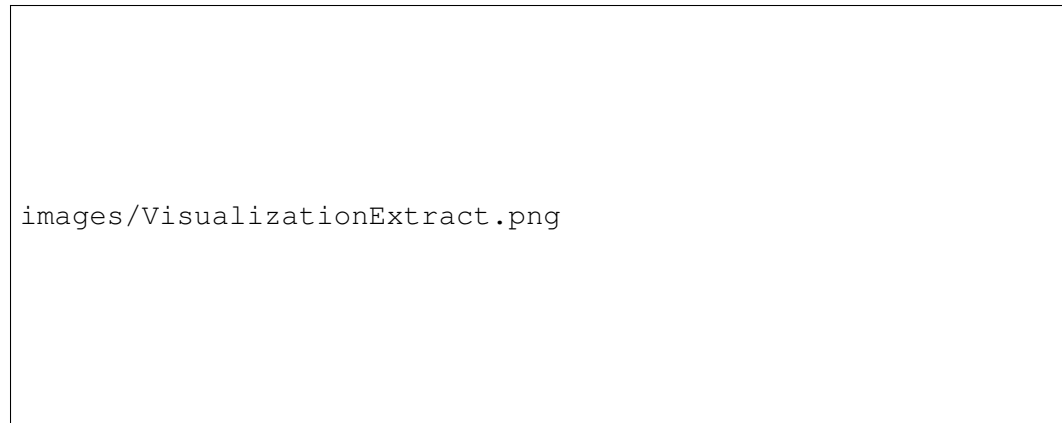


images/browsematches-to-tangerine-01.png

Figuur 4.3: De manier van weergeven uit Browsematches werd gekopieerd naar het nieuwe platform, met uitbreidingen

Elke module krijgt van de applicatie een model toegewezen waar het fragmentparen uit kan opvragen. Dit kan een nieuw model zijn zonder criteria of een gedeeld model. Een gedeeld model behoort niet exclusief tot een module en betekent bijvoorbeeld dat als er één beslist om te sorteren op een attribuut zoals “het verschil van de dikte tussen twee fragmenten”, alle modules die gebruikmaken van ditzelfde model opeens over een gesorteerde dataset beschikken. Via speciale signalen worden zij hiervan op de hoogte gebracht, zodat ze kunnen beslissen of het nodig is een actie te ondernemen. Dit kan handig zijn voor pure visualisatieplugins die geen zoekmogelijkheden aan de gebruiker blootstellen, het kan dan vertrouwen op andere modules om data aan te leveren.

Indirecte communicatie via het model is (voorlopig) de enige manier waarop modules elkaar kunnen beïnvloeden. De structuur van de componenten ziet er uit als in figuur 4.4. Merk op dat er een plugin is (*DetailView*) die geen gebruik maakt van fragmentparen maar eerder van een virtueel tafelblad net als Griphos. Zoals eerder aangehaald kunnen fragmentparen automatisch op een tafelblad gezet worden. Dit



Figuur 4.4: Een vereenvoudigde kijk op de componenten van de visualisatielaag, het hoofdscherm en het model. De componenten in het wit behoren tot de rest van het thera project en zijn niet gemaakt als deel van dit thesisproject.

tafelblad kan dan in 3D weergegeven worden door *DetailView*, waarover later meer.

### 4.3 Dataopslag, synchronizatie, ...

De oplossing om aan de vereisten van centrale en decentrale toegang, grote hoeveelheden data en navigatie te voldoen wordt in hoofdstuk 5 besproken. Het is echter reeds duidelijk dat er een database zal moeten gebruikt worden. Om twee van deze databases te verenigen is er een component ontwikkeld die een aantal fasen doorloopt. De mogelijkheid bestaat om er later nog meer te maken maar voorlopig zijn er 3 fasen: **Gebruikers**, **Paren** en **Attributen**. In elke fase krijgt de gebruiker een scherm te zien met de verschillen tussen de 2 databases die een algoritme heeft gedetecteerd en krijgt ze de mogelijkheid om aanpassingen te maken alvorens naar de volgende fase over te stappen. Het is belangrijk gebruikers keuzes te laten maken waarin ze geïnteresseerd zijn en ze niet lastig te vallen met keuzes die ze niet willen maken (beide zaken negeren leidt tot frustraties [7]). Het algoritme probeert steeds een standaardkeuze in te vullen aan de hand van heuristieken (dit hangt af van de fase), de gebruiker kan deze indien gewenst natuurlijk veranderen. Tegelijkertijd moet de gebruiker geïnformeerd worden van alle stappen die het algoritme zal ondernemen.

#### 4.3.1 Paren

Overeenkomsten met verschillende id    Overeenkomsten met dezelfde id    Conflicten met dezelfde id    Geen conflict, geen overeenkomst

Deze subcomponent laat aan de volgende fasen weten welke acties er zijn ondernomen, meer bepaald welke id-transformatie een bepaalde object heeft meegemaakt

### 4.3.2 Attributen

### 4.3.3 Geschiedenis

## 4.4 Gebruiksgemak / visuele interface

De gebruikersinterface en de achterliggende bibliotheken werden gemaakt met behulp van de Qt toolkit [8]

Vergelijking van wachttijden tussen Griphos/Browsematches en Tangerine

Geheugengebruik BM na grote db: 350 MB (geen afbeeldingen)

Griphos: tonen van een lijst met aparte fragmenten om uit te kiezen (geen equivalent in BM/Tang) Griphos: tonen van een lijst met alle paren (zoals te zien in figuur ...): 7 minuten 18 seconden <— NOOT: Griphos gebruikt dezelfde manier van fragmenten inladen als Browsematches, maar laadt ook op voorhand alle afbeeldingen in. Het geheugengebruik na het openen van 49471 paren is 1.5 GB RAM, het heropenen van dit scherm neemt evenveel tijd in beslag. Het lijkt erop dat het geheugen niet vrijgemaakt noch herbruikt wordt, want bij de tweede keer moest het besturings-systeem het programma afsluiten wegens te veel geheugengebruik. Browsematches: opstarten en een lijst van alle paren tonen: 49471 paren = 160 seconden en 350MB Thesisproject:

	Griphos	Browsematches	Thesis (MySQL)
<b>Opstartsnelheid</b>	1 sec	16 sec	1 sec
<b>Fragmenten inladen</b>	1 min 35 sec	-	-
<b>~4000 paren laden</b>	-	16 sec	0 sec
<b>~50000 paren laden</b>	7 min 18 sec	2 min 47 sec	0-3 sec*
<b>~250000 paren laden</b>	niet getest	niet getest	0-15 sec*

\* Deze getallen zijn afhankelijk van de cache die op de moment aanwezig is, enige vorm van recente activiteit op de database zal ervoor zorgen dat de responstijd rond de 100 milliseconden ligt. De reden dat het soms lang duurt is omdat het programma vraagt achter het totaal aantal fragmenten, en dit is voor de geteste types van databases geen snelle operatie (alles moet geteld worden). Het cijfer verschilt ook per type database

Alle informatie die alleenstaande brokstukken toebehoort — naam, 3D-voorstelling, contour, ... — wordt opgeslagen als een grote verzameling mappen in het bestands-systeem. Hoewel niet alles meteen wordt ingeladen neemt dit soms toch wat tijd in beslag. Hetzelfde geldt voor een collectie paarvoorstellen waarvan een eerste scherm moet ingeladen worden. Weer een andere bron van vertraging bij het opstarten is het inladen van alle modules, vooral degene die gebruikmaken van OpenGL om fragmenten in 3D weer te geven. Deze stappen Om deze reden is er voor gezorgd dat deze operaties parallel kunnen lopen, zodat de gebruiker zo weinig mogelijk moet wachten en steeds informatie kan krijgen over de vooruitgang.

De gemiddelde opstarttijd voor Browsermatches gemeten op een AMD Athlon X2 5600+ met 2 GB RAM is 10 seconden, waardoor de gebruiker reeds snel gaat denken dat er iets misgelopen is. Bij Browsermatches duurde het bijvoorbeeld is nog een. Alle modules kunnen acties zoals knoppen en menu's Elk programma dat Bij het initialiseren

### 4.5 Het beheer van de data

Redenering: Alles vloeit voort uit de paren, het is voorspelbaar (simpel voor te stellen) en ondubbelzinnig

Het zelf kunnen maken van paren is van secundair belang (volgens de thesis), zij kunnen door de HumanMatcher ingevoerd worden in de grote database

Griphos is zeer nuttig -> identificatie locatie van fragmenten in de bak, etc. -> Tangerine is het ontbrekende middendeel!

Dit betekent echter niet dat beide perspectieven elkaar uitsluiten, integendeel.

[TODO: verhuizen naar ontwerp van database]

### 4.6 Visualisatie, een manier om met de data te werken

#### 4.6.1 Model-View-Controller

#### 4.6.2 UML diagramma

[maak UML diagramma]

### 4.7 Integratie in thera project

Gebruik van bibliotheken, ... Extensie door refactoring: FragmentConf -> IFragmentConf ==> SQLFragmentConf | FragmentConf Hierdoor is het nodig om de reeds bestaande code van het thera project om te zetten naar het gebruik van IFragmentConf waar mogelijk maar het aanmaken van FragmentConf anderzijds of FragmentConf ==> SQLFragmentConf Alternatieve oplossing, geen veranderingen in thera code maar SQLFragmentConf sleept dan veel onnodige ballast mee van FragmentConf

### 4.8 Uitbreidbaarheid

[afbeeldingen invoegen]

[afbeelding invoegen, selectie naar grafe transit!]

## Hoofdstuk 5

# Bespreking van het database ontwerp

De visuele delen van het platform bekommeren zich met de **de manier waarop** er moet weergegeven worden, de data laag die nu besproken wordt is verantwoordelijk voor. Vóór dit kan gebeuren moet de gebruiker kiezen waar de paren vandaan moeten komen, hoe ze gesorteerd moeten zijn, aan welke voorwaarden ze moeten voldoen en of er eventueel extra informatie moet weergegeven worden. Dit alles neemt de data laag voor zijn rekening.

### 5.1 Vereisten

De hoofdpunten waaraan deze laag moet voldoen werden reeds opgesomd in hoofdstuk 3, namelijk: snelheid, universele toegang, synchroniseerbaarheid en maximale flexibiliteit voor data-analyse. Eerst wordt een analyse gemaakt van de structuur van de data. Vervolgens wordt er gekeken naar de huidige methode die het therapeut project hiervoor hanteert. Enkele alternatieven worden besproken en op basis daarvan wordt een keuze gemaakt.

#### 5.1.1 Ontleding van de data

De kern van de data is een verzameling fragmentparen, ...relationeel van aard? => SQL database Het Eigenschap-patroon (Property pattern)  
Huidige situatie = Properties pattern?

### 5.2 Het oude systeem: XML-bestanden

De fragmentparen en hun attributen worden door automatische herkenners opgeslagen in een XML bestand zoals in figuur 5.1. Dit formaat is uitermate geschikt voor het overbrengen van de data naar andere subsystemen en is leesbaar door een mens. Het is minder geschikt als een permanent opslag- en zoekformaat. Dat laatste is echter de manier waarop het nu gebruikt wordt, met trage applicaties tot gevolg.

Om een redelijke snelheid te behouden tijdens het zoeken of sorteren moeten Griphos en Browsesmatches het XML bestand volledig in het werkgeheugen plaatsen. Voor een document met 50000 paren neemt dit reeds een goede 300MB in beslag voor paren met elk 11 attributen (zonder afbeeldingen of 3D-modellen). Dit extrapoleren naar een miljoen paren geeft 5GB aan vereist werkgeheugen, hetgeen bij de meeste systemen van vandaag leidt tot het wegschrijven van data naar het bestandssysteem of eerder nog een geheugenallocatiefout.

Het inladen van dit XML bestand is tijdrovend. Tabel 5.1 laat zien dat het inladen minutenlang kan duren. Griphos en Browsesmatches gebruiken dezelfde manier om data in te lezen, maar vertonen desalniettemin een groot verschil in laadtijd. De reden is dat Griphos meteen ook de afbeeldingen van elk fragment ophaalt. Meer nog, na het laden van 50000 paren neemt het Griphos proces 1.5GB RAM in beslag. Het lijkt erop dat dit geheugen niet vrijgemaakt noch herbruikt wordt, want een tweede keer de lijst met fragmentparen openen zorgde er op het testsysteem met 2GB RAM voor dat Griphos automatisch werd afgesloten wegens teveel geheugenverbruik. Browsesmatches (en de thesisapplicatie) laden de afbeeldingen pas wanneer ze eigenlijk nodig zijn en besparen op die manier veel geheugen. Het thesisproject gaat in feite nog verder en laadt de fragmenten zelf pas in wanneer ze nodig zijn, dit komt later aan bod.

	Griphos	Browsesmatches	Thesis
~4000 paren laden	54 sec	16 sec	0 sec
~50000 paren laden	7 min 18 sec	2 min 47 sec	0-3 sec*
~250000 paren laden	niet getest	niet getest	0-15 sec*

Tabel 5.1: Meting van de tijden die elk programma nodig heeft om een collectie paren in te laden

\* De tijden voor de oplossing van de thesis variëren afhankelijk van wat er reeds in het geheugen geladen is, en de achterliggende database. De maximale tijden komen enkel voor indien de database net opgestart is, de gemiddelde laadtijd is ongeveer 100 milliseconden.

Eenmaal geladen, is het niet zo eenvoudig om te navigeren in deze collectie

Als het echter aankomt op het aanwenden van de data om in te zoeken schiet het formaat tekort.

XML bestand volledig in het geheugen ingeladen —> 300 MB voor 50000 paren, dus voor 2 miljoen paren. . .

Eén van de belangrijkste stappen op weg naar een collaboratieve applicatie, is het omvormen van

Het ontwerp van het database model en de implementatie details zijn groot genoeg



**Broncode 5.1:** Uittreksel van een fragmentpaarbestand (hier worden slechts 2 paren getoond)

```
<!DOCTYPE matches-cache>
<matches version="1.0" >
  <match status="0" error="0.187266" old_volume="0.143461" conflict="... 3007 2991 ..."
    num_conflicts="18" tgt="WDC1_0030" overlap="0.401502" volume="1.02974" id="
    2677" src="WDC1_0037" xf="9.96e-01 -8.49e-02 ... 1.00e+00 " />
  <match status="0" error="0.21624" old_volume="0.183338" conflict="... 2710 2666 ..."
    num_conflicts="24" tgt="WDC1_0030" overlap="0.288536" volume="2.51007" id="
    2678" src="WDC1_0037" xf="9.99e-01 -1.62e-02 ... 1.00e+00 " />
</matches>
```

om hun eigen hoofdstuk te verdienen, zij worden later in hoofdstuk [...] besproken.

## 5.3 Alternatieven

Het moet gratis zijn! (niet overdreven veel middelen in het thera project en “enterprise” oplossingen zijn vaak zeer prijzig) ==> open-source of tenminste gratis.

NoSQL (Cassandra, ...) <http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model> — <http://nosql.mypopescu.com/post/573604395/tutorial-getting-started-with-cassandra> (statische sort == slecht voor onze doeleinden) SQL XML database (oud) (zie wiki) XQuery, ...

XML dismissed -> volgende sectie is kiezen tussen SQL en NoSQL

Redenering!

### Paren

#### Attributen van paren

**Complexe informatie die niet in een simpel attribuut past**

#### 5.3.1 De vereisten van een database ontwerp

Zoals in sectie [doelen] omschreven moet de applicatie in staat zijn om data op een uitbreidbare manier uit te lezen en te veranderen, liefst zonder

#### 5.3.2 Pagination

-> beperken van datatransfer -> het niet steeds opnieuw vragen van hoeveel fragmenten er voldoen aan de criteria

## 5.4 SQL of niet

Er is dus een idee van wat er moet opgeslagen worden. Er is een grote variëteit aan oplossingen mogelijk

De zogenaamde “NoSQL” oplossingen schieten als paddenstoelen uit de grond. De meerderheid hiervan functioneren als sleutel-waarde opslag. Door de veelheid aan oplossingen en hun (relatief) jonge leeftijd is het niet evident een volledige vergelijking te maken. Nogal wat artikels over NoSQL oplossingen lijken zonder meer te evangeliseren, waardoor het moeilijk is een realistisch beeld te krijgen van hun specifieke sterktes en zwaktes.

NoSQL, lichte uitleg over NoSQL (Cassandra, ...)

De belofte van object serialisatie en schemaloze<sup>1</sup> opslag is verleidelijk. De basisvereisten van dit thesisproject qua data gaan in eerste aanleg niet verder dan het opslaan/serialiseren van objecten (*User*, *FragmentPair*, *History*). Het niet weten welke attributen er in de toekomst gaan belangrijk zijn lijkt goed te passen in het schemaloze aspect.

Een goede vergelijking van beide aanpakken kan als er gekeken wordt naar: kan de implementatie het soort operaties dat nu nodig is aan (op performante wijze? Kan de implementatie voorziene/niet-voeziene uitbreidingen aan (many-to-many)?

Vergelijking van NoSQL tegenover SQL oplossingen waar die toepasselijk zijn op de vereisten van het project

Berkeley DB, Tokyo Cabinet, CouchDB, MongoDB, Cassandra

Er zijn vele argumenten over wat wel en niet geschikt is voor elk type database, waarvan een heel deel tegenstrijdig zijn. Uit dit kluwen komt eigenlijk slechts één goede raad naar voren: probeer het zelf. Het is echter niet zo evident noch de beste besteding van tijd om twee verschillende maar complete subsystemen uit te bouwen om te zien welke de bovenhand haalt. Omdat er nog verschillende gebruiksscenario's niet gekend zijn werd er gekozen voor een SQL oplossing, waarvan de auteur meer zekerheid had dat het de taak en alle toekomstige taken aankon.

	SQL	
<b>Portabiliteit/Uniformiteit</b>	Hoog (gestandaardiseerde taal)	
<b>Ondersteuning in de toekomst</b>	Hoog	Laag (vele projecten die een

Beide systemen bezitten voord- en nadelen. Wat echter duidelijk is, is dat bij NoSQL databases er op voorhand moet geweten zijn welk type query het meest frequent gaat zijn (Cassandra...) Omdat er geen 100% zekerheid is over de uitbreidingen die gaan komen aan het datamodel

---

<sup>1</sup>In een traditionele relationele database is steeds een schema aanwezig dat precies specificeert wat een object (= rij in een tabel) is. Het is minder eenvoudig om de definitie van object aan te passen door bijvoorbeeld een attribuut (= kolom) toe te voegen, maar niet onmogelijk.

Beide modellen voor dataopslag kunnen in principe dezelfde soort operaties aan, de verschillen bevinden zich in de gemakkelijheid van uitvoeren en de performantie. Het is echter nog steeds niet evident welke uiteindelijk de optimale keuze is. De wereld van het databeheer bevindt zich in een stroomversnelling en op elk gegeven moment kan er een fantastische nieuwe oplossing uit de bus komen. Het valt op te merken dat op de manier er nu van een XML database gemigreerd zal worden naar een andere technologie, dit later opennieuw kan gebeuren als er een duidelijk betere oplossing zich voordoet. Alle operaties op de objecten die in dit project reeds zijn geïmplementeerd, zijn vanuit het perspectief van de gebruiker — archeoloog of ontwikkelaar — database-agnostisch. De schijnbare uitzondering op deze regel lijkt het filteren te zijn, waar een valide (vereenvoudigde<sup>2</sup>) SQL WHERE gebruikt wordt om condities op de eigenschappen van paren te zetten. Deze zijn echter niet zo complex (zie 4.1) en dus gemakkelijk om te bouwen naar iets wat een willekeurige onderliggende database kan begrijpen. De WHERE-syntax kan dus ook, ook al verstaat de database het niet zonder meer. Indien dit niet genoeg is kunnen er gewoon meerdere filtertypes zijn, een SQL-type, een CouchDB-type, enzovoort. Een implementatie van een model zal dan zelf moeten uitzoeken hoe het alles moet vertalen of gewoon de filter weigeren.

Niet alleen is objectpersistentie nodig (de sterkte van key-value NoSQL databases), maar de manier om de objecten te vinden moet zo flexibel mogelijk zijn. NoSQL databases bieden verschillende hulpmiddelen aan om dit te doen (MapReduceCombine, CouchDB Views), ...

Een lichte voorkeur voor de SQL oplossingen is niet verwonderlijk. Er zijn verschillende implementaties met uniforme taal. De auteur is reeds bekend met de werking en als men moeilijk kan voorspellen wat voor een queries zullen nodig zijn (data-mining) blijken SQL databases sneller te zijn. SQL lijkt gemakkelijker te gebruiken.

Een van de belangrijkste voordelen van NoSQL type databases is de ingebouwde schaleerbaarheid over meerdere machines. Het kost werkelijk geen moeite om deze implementaties over meerdere machines te laten samenwerken en hun performantie gaat zo goed als lineair met het aantal machines omhoog. Zullen de datavereisten van het thera project ooit zo groot worden dat een enkele krachtige machine niet meer voldoet? Op het eerste zicht lijkt het van niet, hedendaagse . Daarenboven is het daarom niet onmogelijk om SQL databases in een cluster te laten werken, gratis open-source implementaties als MySQL en PostgreSQL hebben beide ondersteuning voor clustering [9, 10]. Het verschil is de eenvoudigheid waarmee dit uit te bouwen is.

De syntax van NoSQL is niet simpeler, ook niet voor de specifieke queries om paren op te halen (zie infographic). Een andere vaststelling is dat de bibliotheek die door het thera project en dus ook dit thesisproject gebruikt wordt, Qt, ingebouwde

---

<sup>2</sup>Hoewel er niet gecontroleerd wordt of er geen SQL subqueries in de modelfilter aanwezig zijn, is het gebruik ervan niet ondersteund (hoewel het voor de meeste SQL-databases geen probleem is deze filters uit te voeren).

ondersteuning heeft voor SQL databases. Het nalezen van artikels en conversaties op het internet en private correspondentie met ervaringsdeskundigen heeft geleerd dat NoSQL databases hun snelheid laten blijken bij veel transacties en een beperkte klasse van verzoeken, die moet echter op voorhand gekend zijn. In essentie moet de data ingedeeld worden zodat het soort verzoeken dat verwacht wordt snel kan afgehandeld worden [REFERENTIE]. Hoewel een deel van de verzoeken die nodig zijn in dit project daadwerkelijk gekend zijn, is één van de uitgangspunten net dat er een grote variëteit aan verzoeken moet mogelijk zijn.

Om al de redenen is er gekozen om het stabiele SQL pad te kiezen, met de deur open naar eventuele alternatieven wanneer die een overtuigend voordeel bieden.

[figuur mogelijke NoSQL uitbreiding, TokyoMatchModel, CouchDBModel, ...]

Een voordeel van SQL systemen is dat er eenvoudig tussen verschillende implementaties kan gemigreerd worden als de vereisten veranderen. Een broekzakversie die bij momenten geen connectie met het internet heeft kan op het compacte SQLite vertrouwen terwijl een krachtige server een heel grote database kan draaien met MySQL/PostgreSQL/Oracle/.... Al deze verschillende implementaties kunnen met dezelfde subsystemen aangesproken worden. Bij vele NoSQL systemen is dit soort flexibiliteit niet mogelijk (een uitzondering is bijvoorbeeld <http://fallabs.com/kyotocabinet/>).

<http://stackoverflow.com/questions/5438500/example-of-a-task-that-a-nosql-database-cant-handle-if-any> (auto-sync...) <http://stackoverflow.com/questions/2403174/is-there-any-nosql-database-as-simple-as-sqlite> Supergoede argumenten voor en tegen NoSQL/SQL: <http://buytaert.net/nosql-and-sql>

Dat gezegd zijnde, behalve de query interface

## 5.5 Eerste iteratie

### 5.5.1 De database laag

UML-achtig schema met de Database, het MatchModel, SQLFragmentConf, de interoperabiliteit

—> hieruit vloeit de requirement van een matches tabel en attributen

Op deze manier wordt de meest eenvoudige van een *FragmentPair* ongeveer zo geschreven:

Broncode 5.2: Een versimpelde interface voor een *FragmentPair* en diens implementatie met directe verzoeken aan de database

```
class FragmentPair {
    bool hasProperty(string propertyName) { /* query database to see if the property is available
        to this object */ }
    value getProperty() { /* query database to retrieve property */ }
    void setProperty(string propertyName) { /* query database to set property */ }
    void deleteProperty() { /* query database to delete property */ }
};
```

---

### 5.5.2 Een grote tabel of vele verschillende

Om de attributen van een paar te modelleren, kan er gekozen worden tussen deze allemaal op te slaan in de hoofdtabel (*matches*) of voor elk attribuut een andere tabel aan te maken.

Indien het mogelijk zou zijn dat een (grote) subset van paren een attribuut gewoonweg niet bezit, kan het

Hybride aanpak:

History altijd in verschillende tabellen

Zijn de meeste attributen gedefiniëerd voor alle paren?

Voordelen denormalizatie: de index wordt niet ettelijke keren redundant herhaald

Voordelen normalizatie

Simpele attributen, database query voor elk attribuut (dit bleef in stand tot er op een externe server geprobeerd werd) Directe write-through is nooit een echt probleem omdat dit minder vaak gebeurt, op minder fragmenten... men zal veel vaker van venster wisselen dan ineens alle statussen omgooien

Dependency analyse voor filters

### 5.5.3 Complexe informatie voorstellen als een attribuut

Meta-attributen, VIEWS → duplicates (conflicts?)

### 5.5.4 Geschiedenis

De geschiedenis van een attribuut bijhouden heeft twee voordelen: het is nodig voor samenvoegalgoritmen + het is een weer een bron van data (welke paren zijn veel van status veranderd?) + sommige attributen worden op die manier nuttiger,

## 5.6 Verschillende database systemen

### 5.6.1 SQLite

Aan de start van het project leek

### 5.6.2 MySQL

MySQL werd eerst gekozen wegens de alomtegenwoordige

### 5.6.3 De late toevoeging van PostgreSQL

Dankzij de flexibiliteit die nodig was om zowel SQLite als MySQL het grootst mogelijke deel van de code te laten delen

### 5.7 Externe database traag, interne database snel

Het grote probleem dat duidelijk werd na het rechtstreeks werken met externe databases, is dat zelfs op een lokaal netwerk de verzoeken een zeer grote vertraging opleverden.

De redenen hiervoor waren natuurlijk de vele queries die elk object kan versturen om zijn attributen op te halen of te veranderen.

Nog een groot voordeel bij het maken van dergelijk systeem is dat er gebruik kan gemaakt worden van niet-destructieve veranderingen. Op het einde van een werksessie kan men de volledige lokale database ofwel in de hoofd database invoegen ofwel gewoon verwijderen.

Het design moest transparant zijn, zowel een rechtstreekse connectie als een gebufferde connectie zouden voor de eindgebruiker en de ontwikkelaar aan de buitenkant hetzelfde lijken.

Idee: Vervang object ID met object hash voor snellere match-to-match identificatie zonder een globale id nodig te hebben

### 5.8 Slimme client of slimme server?

Het programma bevat alle nodige functionaliteit en de server is in dat opzicht gewoon de specifieke database waarnaar het een verwijzing heeft.

Voor het gebruik op kleine apparaten (zoals tablets) zijn de mogelijkheden voor computationeel zware activiteiten niet zo uitgebreid. Daarenboven moet ervoor gezorgd worden dat de batterijduur van dit apparaat niet te hard beknot wordt door het gebruik van de applicatie. Daarom zal de transitie naar zeer mobiele platformen enkel mogelijk worden indien er een simpele client applicatie kan ontwikkeld worden die op de server vertrouwd om de juiste berekeningen te maken.

In de thesisperiode is geen tijd gevonden om dit concept uit te werken maar er zijn wel plannen gemaakt waardoor dergelijke applicatie op een efficiënte manier zou kunnen ontwikkeld worden. De gebruikersinterface Tangerine voert alle functies met betrekking tot de achterliggende data uit met behulp van een grote bibliotheek van klassen die op zich niets met de interface te maken hebben. Er zou dus een alternatief programma gebouwd kunnen worden die in plaats van met een grafische interface kan bediend worden via een zelfgemaakt protocol. Dit soort ontwerp wordt vaak aangetroffen in de UNIX wereld, waar er een enkele server-variant van een programma bestaat en meerdere mogelijke clients die ervan gebruik maken. Het bekendste is misschien wel de X server.

Materialized views!!!

Model change batching!!! (startBatch/endBatch)

Incompatibiliteiten tussen \*SQL

Analyse van datatoegangspatroon: vooral SELECT, ORDER BY, GROUP BY —> veelvuldig gebruik van indexes

Metadata preloading (na het testen van de snelheid van het ophalen van metadata over een internetconnectie, werd besloten om...)

De gevaren van een stale cache! (en ook de gevaren van een stale window, ook een soort cache)). Oplossing altijd een request sturen om te dubbelchecken?! —> te traag

optimize voor fast reads —> inserts kunnen lokaal gedaan worden, updates hopelijk niet zo veel of lokaal

Recheck om de X seconden: doenbaar indien materialized views met indices! (eigenlijke window query < 1 msec). Systeem zou 1000'en gebruikers kunnen ondersteunen, zeker met een grotere cache

Cache = write-through

Vele disciplines van de computerwetenschappen

dynamisch zoekopdrachten genereren

SQLite formaat maakt database gemakkelijk te delen (USB-stick) <- geen internet connectie

Detecteren van incompatibiliteiten EN veranderingen met reguliere expressies:

Aanpakken voor synchronizatie: moeilijk: changelogging functionaliteit, triggers, ... (mogelijk maar moeilijk cross-db en error-prone) “gemakkelijk”: Maak gewone queries zeer goedkoop

Vereisten: Scaleerbaarheid (XML schaaft NIET) Voordeel van XML is echt wel dat het een mooi outputformaat is voor matchers... import capaciteit voorzien ==> import naar temp SQLite db en merge

Ondersteuning voor “meerdere snelheden”, minder modules hebben is niet erg, elke additie is uitbreiding. Zo ook minder/geen versie van db-schema problemen dependency scanning!

## 5.9 Benchmarking

De query cache staat af Om de variabiliteit van de filesystem (nederlands) cache een beetje buiten spel te zetten zijn al deze routines “opgewarmd”

Pagination Late row lookup

Ideaal = minder dan een seconden voor elke gegeven query vanuit een gebruikersinteractie standpunt (System Response Time and User Satisfaction pagina 5)

Effect van DB configuratie (veel geheugen...)

Suggested workaround voor het text probleem -> sphinx, restrict fragment names (niet ZO gemakkelijk), string + nummer Suggested workaround voor het indexing probleem (zoals gezien voor status IN (...)) -> force een index?! dunno... hij pakt in ieder geval de verkeerde!

'High Performance MySQL', Second Edition, O'REILLY, ISBN: 978-0-596-10171-8 MySQL Reference Manual for version 5.1

<http://nlp.stanford.edu/IR-book/html/htmledition/permuterm-indexes-1.html> (dit is hoe wildspeed werkt) (LIKE performance lijkt niet zo slecht in Postgres, het is de sorting eerder. . . ) <http://www.slideshare.net/techdude/how-to-kill-mysql-performance>  
[http://stackoverflow.com/questions/1540590/how-to-speed-up-like-operation-in-sql-postgres-preferably](http://stackoverflow.com/questions/1540590/how-to-speed-up-like-operation-in-sql-postgres-preferably-use-trigrams-fail) <— use trigrams (fail), MAAR BETER IN 9.1 (future research) <https://cgsrv1.arcc.csiro.au/views-for-postgresql/>



## Hoofdstuk 6

# Modules

Hier komen de modules!!!!

Pure grafische plugins die op andere plugins vertrouwen voor data-selectie: Om de werkbaarheid van dit systeem uit te testen werd een voorbeeldmodule ontwikkeld die alle huidige paren in een grafe plaatst en deze met een ontwaringsalgoritme probeert te plaatsen, zodat men een globaler beeld kan krijgen van de huidige selectie.

### 6.1 MatchTileView

### 6.2 Proof of concept: GraphView



## Hoofdstuk 7

# Toekomstig werk

Met dit project is ook een verdere stap gezet naar mobiele toepassingen. Het data-luik staat bijvoorbeeld toe om applicaties voor tablets te schrijven die beroep kunnen doen op een externe database om een groot deel van het zware werk over te nemen. Deze soort programma's kunnen het manueel verifiëren van paren sneller en aangenamer maken. De huidige werkwijze is als volgt: nadat er een resem waarschijnlijke paren zijn geïdentificeerd, kan men de kisten met fragmenten uit de opslagruimte halen en nakijken welke er écht passen. Gezien de grote hoeveelheid brokstukken is het niet mogelijk om ze allemaal bij de hand te houden. Daardoor duurt het altijd even voor de gewenste fragmenten gevonden worden. In het slechtste geval wordt er gewerkt op een (krachtige) desktop. Hierdoor is het nodig is om ofwel de namen en locaties van de fragmenten te onthouden, of een heleboel afbeeldingen af te drukken. Na het fysisch testen van de fragmenten moet men dan terug naar de desktop om de bevindingen in te geven. Gelukkig behoort een laptop ook tot de mogelijkheden hoewel applicaties als Browsematches en Griphos niet bepaald licht zijn. Het performantieprobleem wordt natuurlijk reeds een deel verholpen door het invoegen van een externe database. Ook kan men nu gemakkelijker met meerdere mensen en laptops tegelijkertijd werken aan de validaties wegens de automatische synchronisatie. Een stap verder zou zijn om een tablet te gebruiken. Er zijn reeds in het verleden experimenten geweest binnen het thera project om aanraakgevoelige omgevingen te maken en de hoop is dat tesamen met de resultaten van deze thesis er in de toekomst iets concreets van gemaakt kan worden.

Het versturen van afbeeldingen over het netwerk! (hoe dit efficient te doen? niet opnieuw uitvinden van het wiel, gebruik een goede codec)



## Hoofdstuk 8

# Besluit

Dit project heeft een lange weg afgelegd. De initiële beschrijving van de thesisopdracht ging over het maken van (innovatieve) gebruikersomgevingen om te helpen met het vinden van passende fragmentparen. De kernvraag luidde: “welke informatie is nodig om te kunnen zeggen of een paar wel of niet past? Hoe kan die weergegeven worden? Kan dit bijvoorbeeld voor 1000 paren tegelijkertijd?” Het zoeken naar dé nieuwe weergave van fragmentparen die een gebruiker in staat zou stellen om op efficiëntere wijze naar de enkele correcte paren in een zee van incorrecte paren te zoeken leverde lange tijd niets op. Niettemin zou het kunnen dat er inderdaad een visualisatie bestaat die een optimale relevante informatiedensiteit bezit. Een voorbeeld van een dense voostelling is een intensiteitsmap van zoveel mogelijk fragmentparen (stel 1 pixel per paar) waar de intensiteit gelijkgesteld wordt aan de beoordeling van een automatische herkenner of een combinatie van meerdere herkenners en menselijke input (cfr. Machine Learning). Maar hier kan een archeoloog niet zoveel meer uit leren dan gewoon te sorteren op deze karakteristiek en in die volgorde paar na paar na te kijken. Als een karakteristiek werkelijk een onderscheid kan maken tussen correcte en niet-correcte paren zal dit in ieder geval moeten gebeuren. Dit soort dense visualisaties heeft dus vooral nut voor de ontwikkelaars van de karakteristieken zelf, zij kunnen een beter overzicht krijgen van de verdeling binnen de set van alle paren. Het passen van fragmenten is een complexe taak, maar niet zo complex dat het onmogelijk lijkt om een getal te plaatsen op de relatieve “goedheid” van een paar. Het zoeken naar zo’n getal of karakteristiek is daarom op zich een waardevolle bezigheid. Dat was echter niet het onderwerp van deze thesis, maar van een andere die in dezelfde tijd aan de K.U.Leuven werd gemaakt binnen het thera project [referentie thesis yassine].

Vele evidente en minder evidente weergaven waren reeds te vinden in het thera project. Men kon de fragmenten op allerlei manieren bekijken: behalve de gewone weergave in kleur kan men ook de achterzijde bekijken, de randen alleen, de dikte van de fragmenten, zelfs de normalen van het oppervlak konden gevisualiseerd worden om een beter idee te krijgen van het reliëf (zo kunnen onder andere krassen en borstelafdrukken zichtbaarder worden). Tevens kon de doorsnede van de plaats waar

beide fragmenten elkaar raken bekeken worden, dit bleek zo nuttig te zijn dat het in Browsematches standaard aan de linkerkant van een paar werd weergegeven. Eén mogelijke uitwerking van de thesis zou dus geweest zijn nog een voorstelling te vinden die snelle en accurate beslissingen toelaat en bij voorkeur compact is. Helaas is het op dit vlak qua ideeën nooit verder gekomen dan incrementele verbeteringen op de huidige resultaten (bvb. niet-lineaire doorsneden van fragment visualiseren). Andere beloftevolle insteken, zoals het gebruiken van contextinformatie en menselijke input om een virtuele classifier te trainen worden reeds onderzocht door onderzoekers Antonio Garcia Castaneda (Clusters) en Tom Funkhouser (Machine Learning) respectievelijk.

Geen revolutionaire ideeën voor visualisaties die een thesis kunnen vullen en een heleboel reeds lopende projecten op alle vlakken, wat nu gedaan? Uit een kijk op de deficiënties van het thera project bleek dat er aan nieuwe delen werd gewerkt maar de infrastructuur niet optimaal was om alles aan elkaar te knopen, om de resultaten bij de gebruiker te brengen. Het project klaarmaken voor een eventueel nieuw paradigma en het toegankelijker maken van de informatie die reeds beschikbaar was, werden het nieuwe doel.

Net zoals een gebroken fresco in feite een puzzel is, kan men het thera project zien als de som van vele delen die in elkaar passen. Dit thesisproject is bedoeld als een stuk dat een ander perspectief biedt op het geheel en het in staat moet stellen om meer en sneller resultaten te boeken. Het complementeert de bestaande aanpakken en zorgt ervoor dat de resultaten van de automatische paarherkenning nog nuttiger gebruikt kunnen worden. Omdat het finale validatiewerk noodzakelijk door mensen moet gebeuren, is het geproduceerde werk waardevol: het kan niet zonder meer opnieuw door een algoritme gegenereerd kan worden. De voor dit thesisproject gemaakte componenten proberen er onder andere voor te zorgen dat het verlies van informatie zo min mogelijk voorkomt door robuuste dataopslag en synchronisatie mogelijk te maken.

Op het vlak van ontginning van nuttige informatie met nieuwe visualisaties en nieuwe manieren om de juiste patronen te ontdekken zijn er natuurlijk nog steeds vele opportuniteiten. Want — zoals opgemerkt in een recente paper over het thera project [citatie siggraph submission 2011] — het vinden van de juiste paren is zoals zoeken naar een naald in een hooiberg. Met elke nieuwe toevoeging aan de mogelijkheden van het platform is er de kans dat deze een manier is om de hooiberg te verkleinen, door te lichten met X-stralen of gewoonweg op een grote krachtige magneet in een windtunnel te plaatsen. Naar deze laatste methode is iedereen natuurlijk op zoek. Tot dan is het zeker belangrijk dat men gemakkelijk kan experimenteren alsook bijhouden en opvragen welk deel van de berg reeds doorkamt is, waar de gevonden naaldrijke aders zitten en wat hun eigenschappen zijn. Misschien zijn de naalden immers niet van metaal...

# Bijlagen





## Bijlage A

# Numerieke resultaten

Nog niets hier



# Bibliografie

- [1] C. Toler-Franklin, B. Brown, T. Weyrich, T. Funkhouser, and S. Rusinkiewicz, “Needles in a haystack: Finding matches among fresco fragments,” in *ACM Transactions on Graphics (Proc. SIGGRAPH)*, Feb. 2011.
- [2] B. J. Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. Dobkin, A. Vlachopoulos, C. Dumas, S. Rusinkiewicz, and T. Weyrich, “A system for high-volume acquisition and matching of fresco fragments: Reassembling Thera wall paintings,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 27, Aug. 2008.
- [3] C. Toler-Franklin, B. Brown, T. Weyrich, T. Funkhouser, and S. Rusinkiewicz, “Multi-feature matching of fresco fragments,” in *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, Dec. 2010.
- [4] J. A. Hoxmeier, P. D., and C. D. Manager, “System response time and user satisfaction: An experimental study of browser-based applications,” in *Proceedings of the Association of Information Systems Americas Conference*, pp. 10–13, 2000.
- [5] B. Shneiderman, “Response time and display rate in human performance with computers,” *ACM Comput. Surv.*, vol. 16, pp. 265–285, September 1984.
- [6] J. Nielsen, *Usability Engineering*. San Francisco: Morgan Kaufmann, 1994.
- [7] J. Spolsky, “User interface design for programmers.” <http://www.joelonsoftware.com/uibook/fog0000000249.html>, 2001.
- [8] Nokia, “Qt online reference documentation.” <http://doc.qt.nokia.com/>, 2011.
- [9] EnterpriseDB, “Replication, clustering and connection pooling.” [http://wiki.postgresql.org/wiki/Replication,\\_Clustering,\\_and\\_Connection\\_Pooling](http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling), 2011.
- [10] Oracle, “Mysql cluster.” <http://www.mysql.com/products/cluster/>, 2011.

## Fiche masterproef

*Student:* Nicolas Hillegeer

*Titel:* Een collaboratief platform voor het efficiënt reconstrueren van fresco's

*Engelse titel:* Een collaboratief platform voor het efficiënt reconstrueren van fresco's

*UDC:* 621.3

*Korte inhoud:*

Hier komt een heel bondig abstract van hooguit 500 woorden.  $\text{\LaTeX}$  commando's mogen hier gebruikt worden. Blanco lijnen (of het commando `\par`) zijn wel niet toegelaten!

Thesis voorgedragen tot het behalen van de graad van Master in de  
ingenieurswetenschappen: computerwetenschappen

*Promotor:* Prof. dr. ir. P. Dutré

*Assessoren:* Ir. N/A  
N/A

*Begeleider:* Dr. ir. B.J. Brown