Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
Departement Computerwetenschappen
Web & Information Systems Engineering Laboratory

MindXpres: An Extensible Content-driven Cross-Media Presentation Tool

Proefschrift ingediend met het oog op het behalen van de titel Master of Science in de Toegepaste Informatica, door:

Reinout Roels

Promotor: Prof. Dr. Beat Signer
Begeleider: Prof. Dr. Beat Signer

AUGUSTUS 2012

Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
Department of Computer Science
Web & Information Systems Engineering Laboratory

MindXpres: An Extensible Content-driven Cross-Media Presentation Tool

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master in Applied Computer Science, by:

Reinout Roels

Promoter: Prof. Dr. Beat Signer
Advisor: Prof. Dr. Beat Signer

AUGUST 2012

Samenvatting

Met een dagelijkse hoeveelheid van ruim 30 miljoen PowerPoint presentaties per dag, kunnen we het belang van presentaties niet negeren in onze moderne samenleving. Van educatie tot in het bedrijfsleven, presentaties zijn overal aanwezig en helpen om kennis door te geven tussen mensen. Digitale presentatie tools (hulpmiddelen) bestaan al sinds de jaren tachtig, maar zelfs na al deze tijd is er bitter weinig veranderd aan de concepten die aan hun kern liggen. Deze tools zijn ontstaan uit analoge toestellen zoals de overhead projector of de dia projector, en tot op de dag van vandaag vormen deze dia's, ook wel gekend als slides, de basis van de zogenaamde slideware tools. Een deel van de problemen eigen aan slideware zouden kunnen vermeden worden door een hervorming van de onderliggende paradigma's. Het sequentiële verloop van slides verplicht de presentator om zijn inhoud te verdelen over meerdere slides, wat het moeilijker maakt voor het doelpubliek om het idee te verwerken. Verder is de navigatie naar niet-aangrenzende slides onhandig en verwarrend. In moderne slideware tools is de opmaak van de inhoud vaak belangrijker dan de inhoud zelf, wat ervoor zorgt dat er vaak meer tijd wordt besteed aan de opmaak dan aan de eigenlijke inhoud. Verder lijkt er een gebrek te zijn als het aankomt op het visualiseren van gespecialiseerde informatie types, zoals bijvoorbeeld broncode of interactieve grafieken.

In deze thesis werpen we een kritische blik op de populaire presentatie tools die tegenwoordig worden beschouwd als de de facto standaard. De populariteit van deze presentatie tools lijkt vooral voort te komen uit feit dat gebruikers niets anders kennen. Sinds het ontstaan van digitale presentatie tools, is een representatie gebaseerd op slides de standaard. Dit zorgt voor een zelf-versterkend effect, omdat mensen die nieuw zijn in de wereld van presentaties opgeleid worden met deze slideware tools. Op zijn beurt zorgt dit er voor dat slideware nog meer gebruikt wordt, waardoor de kans nog groter wordt dat slideware geïntroduceerd zal worden aan nieuwe gebruikers. Deze cyclus toont aan dat oplossingen gebaseerd op slides niet per se de optimale oplossingen zijn, maar eerder dat ze de meest gekende oplossingen zijn.

Door een uitgebreide literatuurstudie en wat kritisch denkwerk identificeren we een aantal problematische aspecten die inherent zijn aan de basis ideologieën waarop moderne presentie tools gebaseerd zijn. Door alternatieve

aanpakken beter te bestuderen zien we dat een klassieke aanpak gebaseerd op slides zeker niet de enige is. Door onze blik te verbreden, en door oude vaste waarden los te laten, kunnen we gebruik maken van state-of-the-art research uit de onderzoeksgebieden van informatiebeheer en informatie visualisatie, en deze toe te passen in de context van presentaties. We herbekijken het gehele presentatie proces, en vinden die opnieuw uit op een manier die steek houdt voor de de 21ste eeuw. Door gebruik te maken van onze bevindingen, komen we tot een nieuwe oplossing, genaamd MindXpres.

MindXpres is een nieuw kijk op presentie tools, en baseert zichzelf op recent onderzoek in de onderzoeksgebied van hypermedia, spatial hypertext en zoomable user interfaces. Door een onderliggende basis te gebruiken van semantisch gelinkte informatie, is onze tool in staat om informatie op te slaan en te verwerken op een manier die dicht ligt bij de denkwijze van de mens. In plaats van informatie te moeten spreiden over disjuncte slides kan informatie nu opgeslagen worden in zijn originele vorm. Een LaTeX -achtige taal die instaat voor de creatie van presentaties plaatst de focus terug op de inhoud, wat de benodigde tijd verlaagt om een presentatie te maken, en wat de kwaliteit verhoogt van de te presenteren informatie. MindXpres visualiseert inhoud automatisch door gebruik te maken van thema's en een uitbreidbaar plug-in systeem. Deze visualisatie laag combineert een zoomable user interface met psychologische concepten die helpen bij ruimtelijke redenering, wat helpt bij het intact houden van het te presenteren idee. Hierdoor wordt het ook makkelijker begrijpbaar voor het publiek. De uitbreidbare MindXpres kern laat gebruikers toe om nieuwe types van informatie toe te voegen, of om de manier waarop data behandeld wordt aan te passen. Verder gebruikt MindXpres de nieuwste HTML5 standaarden om een maximale compatibiliteit te garanderen met de toestellen die eigen zijn aan deze tijd.

Terwijl het met MindXpres perfect mogelijk is om klassieke presenties te creëren die gebaseerd zijn op slides, biedt MindXpres veel meer functionaliteit aan, die verder gaat dan de meest recente slideware tools. MindXpres laat de gebruiker toe zich los te rukken van de lineariteit en de beperkingen die opgelegd worden door slideware tools. Via onze keuzes naar de gebruikte technologieën toe, laten we nieuwe manieren van input toe, samen met een ongezien niveau van interactiviteit. Daarbovenop laat de plug-in architectuur de gebruiker toe om informatie types te visualiseren die niet toegelaten worden door andere presentatie tools. Als een eerste stap in een geheel nieuwe richting, presenteren we ons presentatie framework, dat de basis zal vormen voor heel wat toekomstig werk. Via onze innovatieve MindXpres oplossing verbeteren we in de nabije toekomst de presentatie ervaring voor zowel de presentator als het publiek!

Abstract

With over 30 million PowerPoint presentations produced every day, one cannot deny the importance of presentations in modern society. From education to business, presentations are omnipresent and help to transfer knowledge between people. Digital presentation tools have been present since the 1980s but even after all this time, very little has changed to the core ideas. They evolved from analog devices such as the overhead or slide projector, and the concept of a slide still forms the foundation of so-called slideware solutions. Some of the issues of current slideware could potentially be resolved by a shift in paradigms. The sequential traversal of slides forces presenters to spread their content over multiple slides, making it more difficult to understand an idea. Furthermore, the navigation to non-neighbouring slides is awkward and confusing. In modern slideware tools, the presentation of content is more important than the actual content itself, resulting in more time spent on the visualisation rather than on the content. Last but not least, there seems to be a hiatus regarding the presentation of special content types such as source code or interactive graphs.

In this thesis, we take a critical stance towards the popular presentation tools that have become a de facto standard. The popularity of modern slideware tools seems to be partially caused by the fact that users do not know any better. Ever since the birth of digital presentation tools, the slide-based structure has been a standard resulting in a self-reinforcing cycle. People who are new to presenting are trained in the use of slideware tools, which in turn strengthens these same reasons for others to use these tools. This cycle indicates that even though slideware is the most popular solution, it does not per se imply that it represents the most optimal approach.

Through literature study and critical thinking, we identify a number of inherent flaws regarding the core ideologies that define modern presentation tools. A closer look at alternative approaches reveals that a classical slide-based approach is definitely not the only possibility. By broadening our view and letting go of the fixed values in presentation tools, we were able to apply state of the art research in information management and visualisation, effectively rethink the entire presentation process and create a set of requirements for what we believe to be the ideal presentation tool.

Our MindXpres presentation solution is a fresh look on presentation tools and bases itself on recent research in the areas of hypermedia, spatial hypertext and zoomable user interfaces. By using an information base of semantically linked data, our tool is able to store and process information in a more human accessible way. Instead of having to spread complex ideas over disjunct slides, information can be stored in its original form. A LaTeX-like language for presentation authoring brings the focus back to the content, reducing the time required to create a presentation and increasing the overall quality of the presented information. MindXpres automatically visualises content via themes and an extensible plug-in system. This visualisation layer combines a zoomable user interface with psychological concepts that enforce spatial reasoning, which keeps the original idea intact and easier to process by the audience. The extensible MindXpres core allows users to add new content types or change the way information is handled. Furtermore, the use of the latest HTML5 standards for the visualisation ensures the maximum portability of presentations.

While it is perfectly possible to create classic slide-based presentations with MindXpres, our solution offers features that go beyond state of the art slideware tools. MindXpres provides the means for breaking free from the linearity and the restrictive nature of existing slideware. The choice of technology allows new ways of input and a degree of interactivity that is previously unheard of. On top of that, the plug-in architecture provides the possibility to present content types not supported by existing tools. As a first step in a new direction, we provide a solid presentation framework that will form the centre of focus for future research. Through our innovative MindXpres solution, we are going to enhance the presentation experience for the presenter and audience alike in the near future!

Contents

1 Introduction **2**

2 Limitations of Existing Presentation Tools **6**
2.1 The History of Slideware 6
2.2 Common Slideware Tools 8
2.2.1 Microsoft PowerPoint 8
2.2.2 Apple Keynote 10
2.2.3 OpenOffice Impress 11
2.3 The Linearity of Slideware 12
2.3.1 The Nature of the Human Mind 12
2.3.2 Side Effects of Linear Traversal 14
2.4 Paper Emulation 17
2.5 Lack of Input Methods 20
2.6 Distance Between Audience and Presenter 20
2.7 Non-Content-Driven Slideware 22
2.7.1 Content Creation 22
2.7.2 Managing Content 23
2.8 NASA Use Case 24
2.9 Coping Techniques 26
2.10 What Now? 29

3 Towards Better Presentations **30**
3.1 Hypertext 30
3.1.1 Semantic links 31
3.1.2 Spatial Hypertext 32
3.1.3 Transclusion 36
3.2 Zoomable User Interfaces 37
3.3 Alternative Tools 37
3.3.1 Prezi 38
3.3.2 Beamer 40
3.3.3 Fly 42
3.3.4 CounterPoint 42
3.3.5 SlideRocket 43
3.3.6 SlideShare 44
3.3.7 Google Docs 45

3.3.8 PaperPoint . . . 46
3.4 Overview . . . 48
3.5 Newly Introduced Issues . . . 49
3.5.1 Excessive Animation . . . 50
3.5.2 Map Shock . . . 50
3.5.3 Old Habits Die Hard . . . 50
3.5.4 Accessibility of Online Tools . . . 50
3.5.5 Ownership and Copyright . . . 51
3.5.6 Tool Popularity . . . 51

4 The Ideal Presentation Tool 52
4.1 Hypermedia . . . 52
4.1.1 Transclusion . . . 54
4.2 Zoomable User Interface . . . 55
4.2.1 Multiple Navigation Paths . . . 57
4.2.2 Navigational Links . . . 57
4.3 Focus on Content . . . 58
4.3.1 Semantic Content Creation . . . 58
4.3.2 Themes . . . 59
4.3.3 ZUI Layout . . . 61
4.3.4 Presentation Structure . . . 63
4.3.5 Import of Existing Content . . . 64
4.4 Content Presentation . . . 64
4.4.1 A Component Plug-in Architecture . . . 65
4.4.2 Examples . . . 65
4.5 Usability . . . 68
4.5.1 Search . . . 68
4.5.2 Generic Input Interface . . . 68
4.6 Portability . . . 69

5 The MindXpres Presentation Tool 72
5.1 The RSL Model . . . 72
5.1.1 The Link Metamodel . . . 74
5.1.2 The User Model . . . 75
5.1.3 Layers . . . 76
5.1.4 Structural Links . . . 77
5.1.5 Implementation of the RSL Model . . . 79
5.1.6 Conclusions on the RSL Model . . . 79
5.2 Content Creation in XML . . . 80
5.3 Content Visualisation . . . 81
5.4 MindXpres Architecture . . . 84
5.5 Extensibility Through Plug-ins . . . 87
5.5.1 Components . . . 87
5.5.2 Containers . . . 88

5.5.3 Structures 88
5.6 Conclusion 89

6 Implementation **92**
6.1 Goals 92
6.1.1 XML Authoring Language 92
6.1.2 Compiler 93
6.1.3 Visualisation Layer 93
6.2 XML Authoring Language 93
6.3 Compiler 97
6.3.1 Document Validation 97
6.3.2 Parser 98
6.3.3 Transforming 100
6.3.4 GUI 101
6.4 Visualisation Library 102
6.4.1 Library Architecture 102
6.4.2 Dependency Injector 103
6.4.3 Bootstrap 106
6.4.4 Core 107
6.4.5 The Plug-in Architecture 112
6.4.6 Implemented Plug-ins 113
6.4.7 Themes 116

7 Use Case **118**
7.1 The Scenario 118
7.2 Creating the Presentation 119
7.2.1 Creating Structure 119
7.2.2 Slides 121
7.2.3 Basic Content 124
7.2.4 Source Code 126
7.2.5 Youtube Plug-in 127
7.2.6 The Video Plug-in 129
7.3 Conclusion 130

8 Conclusions and Future Work **132**
8.1 Contributions 132
8.2 Future Work 134
8.2.1 XML Authoring Language 134
8.2.2 Compiler 135
8.2.3 Visualisation Layer 136
8.2.4 General 136

1 Introduction

We are all familiar with common presentation tools such as Microsoft PowerPoint[1], Apple's Keynote[2] or OpenOffice Impress[3] which forms part of the recently split off LibreOffice[4] project. They are used in education, business and pretty much every other situation where we want to orally transfer information to a group of people. At the moment of writing, Microsoft estimates that over 30 million MS PowerPoint presentations are produced every day [40]. PowerPoint has an estimated market share of about 95%, but alternatives such as Keynote and OpenOffice Impress are also actively used. Even though they are offered by companies with different motives, the offered features are more or less the same. Because most of us grew up with these tools and were educated in their usage, we rarely question them. Ever since their introduction in 1987, these presentation tools became a de facto standard. Digital slideware tools evolved from older presentation tools such as overhead projectors and photographic slide projectors and this is still very noticeable in today's presentation tools. Even 25 years after the introduction of digital presentation tools very little has changed and the core idea is still very much the same.

The first question that we ask ourselves is "*what do we use presentation tools for?*". The answer to this question is almost always to help bring across information to an audience during an oral presentation. Therefore a presentation tool should help to bring some structure into a presentation by breaking up

[1] http://office.microsoft.com/en-us/powerpoint/
[2] http://www.apple.com/iwork/keynote/
[3] http://www.openoffice.org/product/impress.html
[4] http://www.libreoffice.org/features/impress/

a complex idea into smaller parts. By doing so, we are also introducing an order and a pace to the presentation. For example, if the presenter shows the audience what topics they are going to talk about, the audience will have a general idea of the advancement during the presentation. One of the main reasons though, is to provide a summary of what is being said, the key points or *power points* if you will. They show the audience what is to be remembered from the long stream of information that is delivered to them. Such a tool can also be used to augment what the presenter is saying, usually by presenting information that is hard to pass on orally, including images, tables or statistics. Last but not least, presentation tools are also used to keep the audience's attention and to keep them motivated. We conclude that presentation tools can facilitate the transfer of knowledge from the presenter to the audience if used correctly. This is however not always the case and many features of existing tools are often abused resulting in an adverse effect.

When we examine these presentation tools more closely, we can identify some issues that can be improved, some of which have been there since the very beginning. The most obvious issue is that the concept of a *slide* forces users to present information in specific ways that are far from optimal. Because of the spatial limitations, content is spread over multiple slides which has serious implications on how content is split up, and thus also on the understandability and orientation. The linear navigation in a sequential set of slides can be problematic for both the presenter and the audience, and does not give a good overview of the content. On top of that, most presentation tools tend to focus on aesthetics rather than of the actual content. These editors are chock-full of graphical features such as animations, fonts, colours, clip art and transitions, but only rarely are there features that are beneficial to the actual content itself.

We believe that the current established standards are wrongfully clinging to their origins. The digital tools were created to facilitate the creation of analog media such as overhead beamer sheets for photographic slides. It made sense at the time because these were time consuming and error-prone activities. Such a tool allowed presenters to lay out their slides digitally before printing them on these analog media. However, time has changed and these analog media are long gone. Modern technology makes it possible to project content in real time from a computer, yet we are still stuck with slides, together with all the associated disadvantages. Similar to the revolution Ted Nelson [35] envisions for documents, we believe that it is time to stop seeing presentation tools as simulators for analog media and instead use computers to their full potential.

In this thesis, we explore the current state of presentation tools and investigate areas where improvements could be made. In Chapter 2, we start by

looking at the tools everyone knows: PowerPoint, Keynote and OpenOffice Impress. Through literature study, we look at their history, the way they work and the implications of their ideologies. This establishes a set of problematic areas which will be used as a basis for the rest of this thesis. However, just because tools like PowerPoint and Keynote are the most popular does not mean that they are the only ones. In Chapter 3, we take a look at alternative approaches, related technologies and some non-mainstream tools that bring new ideas to the stagnant pool of popular slideware tools. This gives us a better view of how it could be, as opposed to how things are now. By expanding our view to information management in general, we can take advantage of recent work in active fields such as hypertext and hypermedia.

Together with valuable conclusions extracted from the alternative tools discussed in Chapter 3, and based on our own input, we can effectively redefine some core paradigms for presentations. By paying attention to how humans process information, and the issues we discovered in Chapter 2, we propose a new approach that we believe to be an improvement over the established slideware tools. In Chapter 5, we detail our MindXpres presentation solution, while setting up a set of basic requirements for our ideal tool. This solution is then implemented and discussed in Chapter 6. In order to demonstrate the benefits that MindXpres claims to have, we follow up on the implementation with a use case. This is done in Chapter 7, where we walk through the entire presentation process, discussing our improvements over existing tools along the way. As this is a Master's thesis, time was limited and our work is restricted by the available time. However, we hope that our solution provides a basis for extensive future work which we discuss together with the final conclusions of this thesis in Chapter 8.

By rethinking the entire presentation process, we aim to provide a better experience for both the presenter and the audience. We are confident that our proposed MindXpres solution improves many of the issues we have found and greatly increases the quality of knowledge transfer. We now present our contributions that will hopefully mark the beginning of an innovative cross-media presentation tool.

2 Limitations of Existing Presentation Tools

In this chapter, we take a closer look at some of the limitations of common presentation tools such as PowerPoint, Keynote and OpenOffice Impress. What these tools all have in common is that they are based on slides and therefore we shall refer to these tools as *slideware* from now on. We will first elaborate on the frequently used slideware tools and their history. Once the common concepts have been established, we can make some critical observations on the average presentation as they are often given at the time of writing.

2.1 The History of Slideware

Before the existence of computers, presentations were often purely oral, sometimes augmented with handouts or information written on a chalkboard. Around 1945, the overhead projector was invented. By using a lens, mirrors and a bright light source, some content on a transparent plastic sheet was projected. In the early days of overhead projectors the content was first created on regular paper by typewriter or by manual writing and drawing. A black-and-white photocopy was then made on a plastic sheet so that the non-white parts of the paper were transferred to the plastic sheet. When shining a light through such a sheet, the transparent background lets the light through, but the darkened areas block the light creating shadows in the shape of the content, effectively projecting the content. Thanks to the lens and mirrors, the content was projected in the desired direction in a focussed manner. An unintended side effect of this method was that anno-

tations could be made on the plastic sheet with a marker (before or during the presentation), which could be cleaned off again afterwards. Overhead transparencies generally had a clear and simple presentation of the content as fancy decoration was simply not practical.

Figure 2-1: Overhead projector

A couple of years later, the *slide projector* became commonly available. This device is also commonly referred to as a *dia projector*. The slide projector could project series of photographic slides, with the benefit of having colour and smaller more portable slides. These improvements over the overhead projector allowed for more creativity regarding the presentation of content as artists could now make use of colours, backgrounds and pictures. These slides could however not be annotated anymore during the presentation and a darkened room was required for optimal viewing.

Figure 2-2: Slide projector

The next step in presentations were the so-called multimedia shows during the 1970s. By using multiple slide projectors (up to dozens) the rapid alternation between slides was used to create the effect of motion. The benefit over a regular film was that it allowed for easier effects such as fading in or out, or combining multiple images. These shows were often synchronised with an audio track giving them the name *multimedia shows*. Examples of

such shows can be found on YouTube[1]. However, as one can imagine, these shows were time consuming to create and very expensive to display. Because of this, they were quite rare and were only given to large audiences with the goal of entertainment.

When computers became more widespread, it became apparent that presentations could be created and modified digitally. At first, these digital documents had to be printed to an analog medium such as transparent sheets or photographic slides for analog projection, but as technological advances were made, computers could be used to communicate with projectors without an intermediate medium. This allowed computers to project digital content in real time. Today, in 2012, anyone can easily create their own digital presentations and projectors are affordable and omni-present pieces of hardware.

2.2 Common Slideware Tools

2.2.1 Microsoft PowerPoint

PowerPoint was born in 1984, but not under that name and not at Microsoft. In that year, the small start-up Forethought, Inc. would recruit a PhD student called Robert Gaskins who would come up with the idea. Based on his product proposal [14], the start-up developed a program called Presenter, which was renamed to PowerPoint after trademark problems. The product was developed for the Apple Macintosh and allowed users to create black and white text and graphics for overhead transparencies. It supported different fonts, easy editing and even the integration for charts and diagrams from external applications such as MacDraw and MacPaint. The main benefit was that the presenter (e.g. business man, CEO or professor) could now create their own transparencies instead of having to work through a typewriter operator (e.g. a secretary) as it was common at the time. As new Mac models were released, the software was updated to support colour and it was now possible to create slides for use in a slide projector. In order to do so, the digital files had to be sent to a professional photo processing lab where the presentation was literally photographed on small pieces of film in order to create photographic slides.

In 1987, Forethought was acquired by Microsoft which continued the development. It was not a surprise that Microsoft ported the application to Windows and in 1990, the first version of Microsoft PowerPoint was released, on the same day as the Windows 3.0 release. Both Mac and Windows versions were maintained and by 1992 PowerPoint 3 was released. This new

[1] http://www.youtube.com/watch?v=4hCYWb3G0EA

version added support for real-time projection from a computer and introduced features such as slide transitions and animation of content, which did not make sense for earlier display media such as transparencies.

As technology advanced, the application was further updated and support was added for audio, video and fancier effects. The most notable change came in 1997, when PowerPoint incorporated the Visual Basic for Applications (VBA) macro language. This allowed users to add custom functionality to both the editor and the actual presentation. Examples of possible uses of VBA are countdown timers, the export or import of specific slides or the implementation of custom effects. At the time of writing, the PowerPoint tool shown in Figure 2-3 is one of the most widely used presentation tools, and versions are maintained for both Windows and Mac.

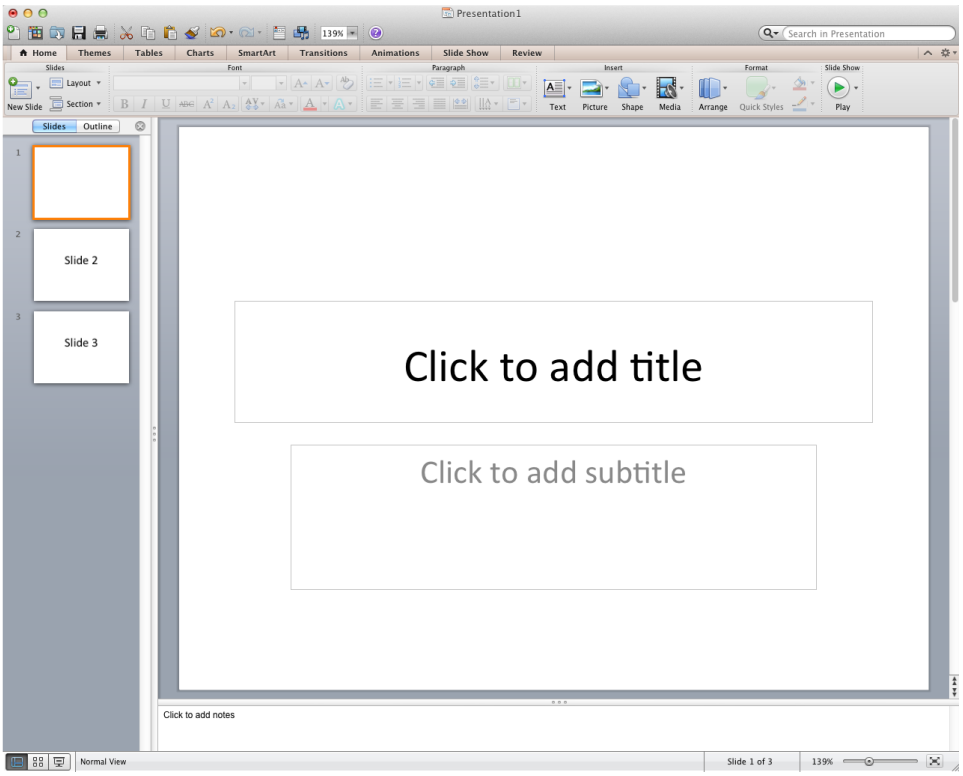


Figure 2-3: Microsoft PowerPoint

It is clear that the application has been designed to be usable by anyone. The tool is and has always been an intuitive graphical slide editor. Via simple operations such as dragging and dropping one can create a simple presentation within minutes. This level of usability helped the application to become popular and to remain a de facto standard for creating presentations.

2.2.2 Apple Keynote

Nowadays Apple's Keynote can be considered as an alternative to PowerPoint but one should mention that it does not have a lengthy history as PowerPoint does. Keynote started as an internal Apple application, mainly used by Steve Jobs for creating the presentations for the Apple keynote events where new announcements are made. In 2003, the first public version was released and it has been maintained and updated up to this day.

When viewing the bigger picture, Keynote is very similar to PowerPoint. In general, it follows the same basic concepts as PowerPoint. It is also an intuitive graphical editor for slides with support for multimedia, charts, animations and effects. One thing that might set it apart from other tools is that it is a common opinion that Keynote helps to maintain a high level of quality considering the graphical design and typography, as commonly found in other Apple products. This is done via a list of predefined good looking themes and fancy effects but of course this is highly subjective and it does not make it a radically different type of product compared to other tools. An small selection of themes is shown in Figure 2-4.

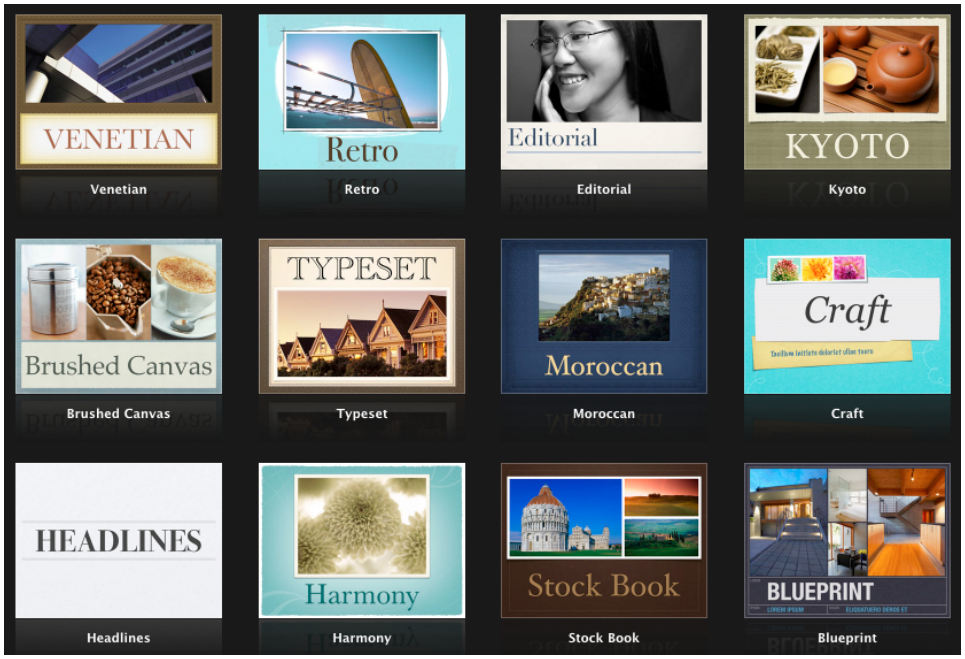


Figure 2-4: Some of the many Keynote themes

Notable is the interoperability of Keynote with other Apple products. Data is easily exchanged with other non-related applications such as iDVD or Garageband. Additionally, Keynote has good integration with mobile Apple devices. For instance, iPhones or iPods can be used as remote controls for presentations and a full version of Keynote is available for the iPad, including support for touch input.

2.2.3 OpenOffice Impress

Impress is a presentation tool forming part of the OpenOffice suite. Like PowerPoint, the application started in another company. Originally, the suite of tools was known as StarOffice, created by the company StarDivision. In 1999, this company was acquired by Sun Microsystems. After the acquisition, Sun Microsystems renamed the project to OpenOffice and made the source code publicly available, effectively building a sponsored open source community around the project. In 2010, Sun Microsystems itself was acquired by the Oracle Corporation and financial support for the project was promptly stopped. The codebase was donated to Apache, where it now lives on as Apache OpenOffice. Because of its free and open nature, this tool suite is particularly popular among users of open operating systems like Linux or BSD. However, the suite is not limited to these open operating systems and is available for any commonly used operating system.

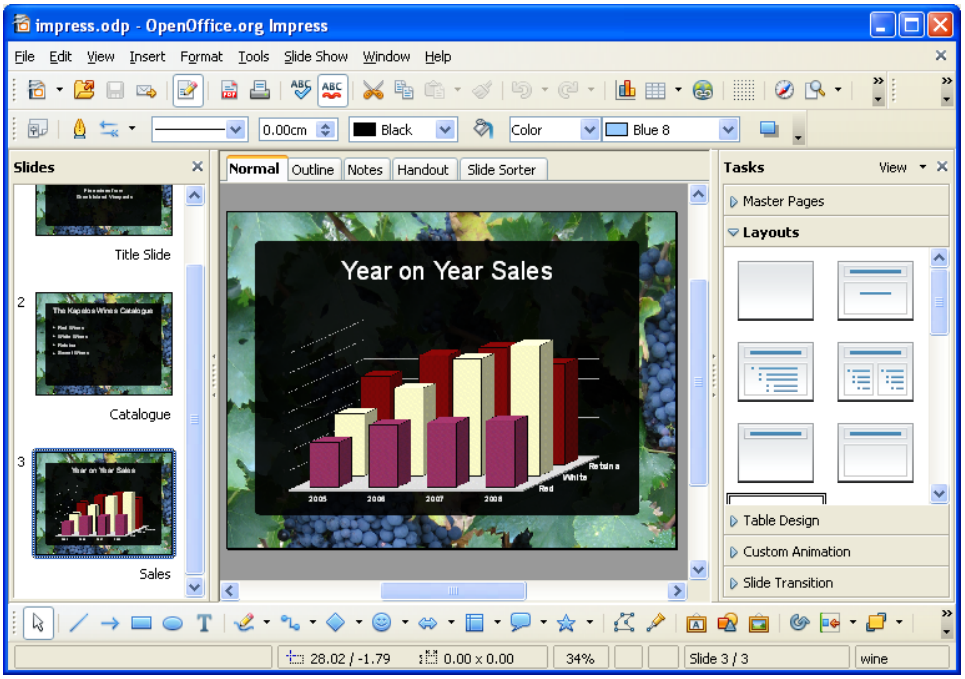


Figure 2-5: OpenOffice Impress

Feature-wise, OpenOffice Impress is comparable to PowerPoint. The tool was never meant to be innovative but instead places itself as an open source alternative to existing tools such as PowerPoint. As shown in Figure 2-5, Impress is a visual slide editor comparable to the other common presentation tools. Just like the other tools, there is support for effects, fonts, multimedia and transitions. Notable is that Impress provides bilateral support for PowerPoint documents making it easier for people to use Impress in environments where PowerPoint is seen as a standard (e.g. school or work).

2.3 The Linearity of Slideware

We are all used to the sequential progression through a predefined slide set offered by existing slideware. However, even this core idea raises some questions. The fact that slideware is limited to this linear format means that content has to be adapted in order to fit in this sequential form. The problem is that a linear representation is often not the best way to present content. Of course this depends on the content itself but in the following sections, we present some issues that arise by forcing users to flatten a complex idea into a one-dimensional sequence of slides.

2.3.1 The Nature of the Human Mind

As Vannevar Bush explained so well in his essay *As We May Think* [5], the human mind works by association. This applies particularly to how humans process and store information. However, computers store and process data very differently. As Bush stated on the retrieval of information by a machine:

> "*Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing. Where data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. It can be in only one place, unless duplicates are used; one has to have the rules as to which path will located it, and the rules are cumbersome.*
>
> *The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain.*"

We can quickly see how this applies to us in reality. Ever since we were children, we have learned by association. For example, when we are first introduced to simple math problems, they come in forms such as "*If person A has 6 pieces of candy and wants to share them equally with his 2 friends, how many pieces does each person receive?*". This works because children can

relate to the experience of having candy that needs to be distributed among friends. As we grow up, this mechanism evolves with us as we keep on learning new things by associating them with experience or mental models that we already know. We facilitate the learning process by making analogies to things we already know. We could state that pretty much any complex idea could be reduced to a simpler analogy that anyone is bound to know. As an example, one could simplify the human vascular system to a postal distribution system where transportation units carry around packages as shown in Figure 2-6. Of course, learning by association is not limited to analogies but could also include related pieces of information that the learner might already know.

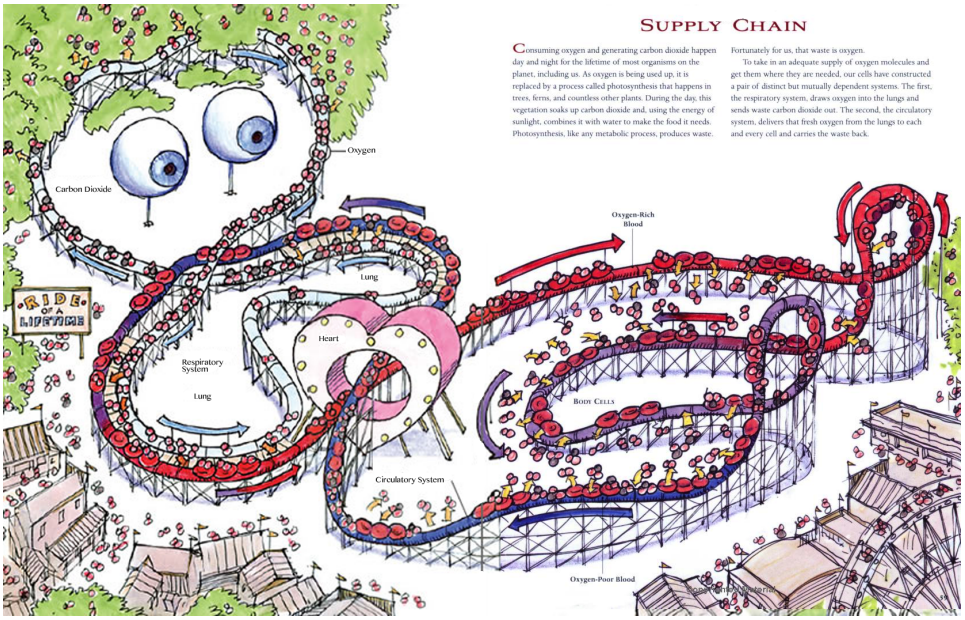


Figure 2-6: The circulatory system, simplified by associating with a simpler concept (Source: *The Way We Work*, by David Macaulay)

However, computers tend to look at data differently. Computers store data in containers such as sequences (e.g. lists of numbers or strings) or hierarchies (e.g. filesystems or the OO paradigm). One could argue that graphs and databases can link information by association, but we must not forget that it is often the humans who make the really meaningful connections in these structures. A computer does not truly understand the semantic meaning of the information and therefore it is very difficult to have it automatically detect associations. Even when we look at recent developments such as ontologies, we see that these definitions and conceptualisations are created by humans and that these are simulations of human associative behaviour by

a computer. It is true that ontologies can be generated semi-automatically but they still require human intervention at some point. Neural networks are a step in the right direction, as these are modelled after simplified neural processing in the human brain. They are trained with information and over time links between "neurons" are strengthened or decayed in order to make connections between given situations. Such neural networks have their uses and show a sort of simulated associative behaviour but it is not feasible to employ them in larger domains. An example of neural network usage would be for Optical Character Recognition (OCR), where the network has to associate a given image (of a character) with the corresponding character, as this is just a matrix of pixels to a computer. A recent paper by Google mentions that they have built a neural network consisting of 16000 processors in order to automatically detect high-level features in images and videos (such as humans or cats) [29]. While this is a major achievement, it also illustrates that even with that much computation power, the human brain still excels at associating and recognising data by a large margin.

Because this is the way humans learn, the concept of association should be a fundamental element in presentations. However, none of the tools allow things like associative linking and it is entirely up to the audience to remember all the information and create the necessary links.

2.3.2 Side Effects of Linear Traversal

Having said that, we clearly see a difference in data processing between humans and computers. However, we also see that slideware enforces the computer's style of thinking on the user. In order to present a complex idea, the user is forced to break it down into a hierarchical structure. Say that we think of the idea in our head as a cyclic graph of associated data, then it is obvious that it becomes a problem if we want to transform this graph to a tree-like structure. Connections have to be removed in order to do so. To make things worse, the user also has to transform this hierarchy of data into a one-dimensional linear traversal of the nodes. More practically explained, if the user wants to show a link or association to a previously shown slide that is not adjacent in the sequence, they need to jump backwards or forwards a number of slides. Practically, this often results in an awkward slide traversal while searching for slides, which might confuse the audience. The enforcement of a linear structure also has a lot of other side-effects next to the linear traversal and in the following, we list the most important ones.

Lack of Overview

Due to the sequential layout, some overview issues are introduced. For instance, the presenter cannot see all past slides or slides to come. While some tools offer a specific presenter view which shows the previous and next slide as shown in Figure 2-7, this is of limited benefit since it does not really provide navigational value. Similarly, it may be difficult for the audience to situate themselves in the presentation (e.g. "*How many more slides on this subchapter?*"). The creator of a presentation is usually responsible for creating indicators for progress and position, such as slide numbers, titles and tables of content.

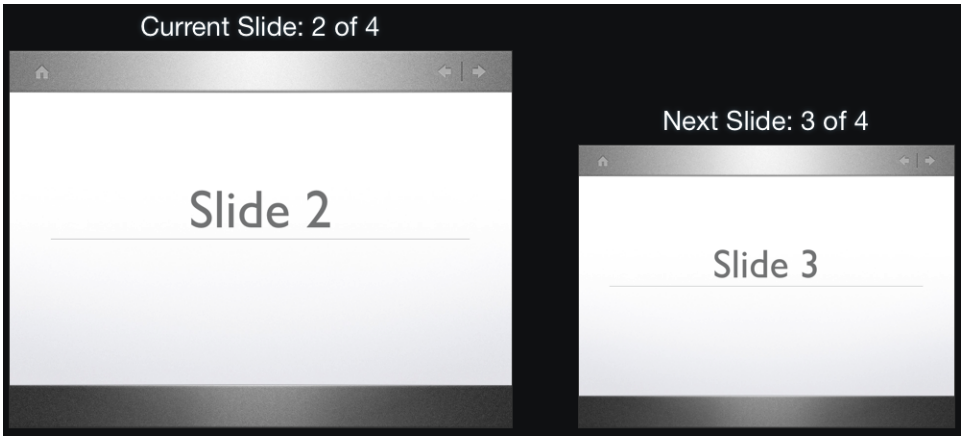


Figure 2-7: Keynote's presenter view

Little Distinction Between Levels of Detail

When we come back to the hierarchical representation of data, we see another problem. While a hierarchy can be presented on a single slide through the means of a bullet list, it is often not possible for most complex ideas to fit on a single slide. This means that we have to spread our hierarchy over multiple slides. By doing so, it is easy to loose detail indicators. When using a bullet list on a single slide, the bullet style and/or font will provide detail indication, but there is only a limited depth we can achieve. This becomes even more problematic when we spread big hierarchies over multiple slides. In short, when showing a specific slide to the audience, they might not know how deep they are in the hierarchy or how much deeper it will get. It might also not be obvious to separate a slide presenting a main idea from a slide presenting a sub-sub-sub-detail. This can be solved by visualising detail indications such as sections/subsection numbering or displaying the current level (e.g. "Presentations → Best Practices → Bullet Lists") on each slide. Nevertheless, this is again the responsibility of the presentation creator.

No Optimal Use of Spatial Reasoning

Humans live in a 3D spatial environment. Because of this, the human mind is hardwired to use spatial reasoning. Concepts such as distance and size have meaning to us even outside of their physical sense. For example, when someone opens their arms very far apart, and tells us he loves us "this much", we know he loves us a lot, even though there is no measurement for love.

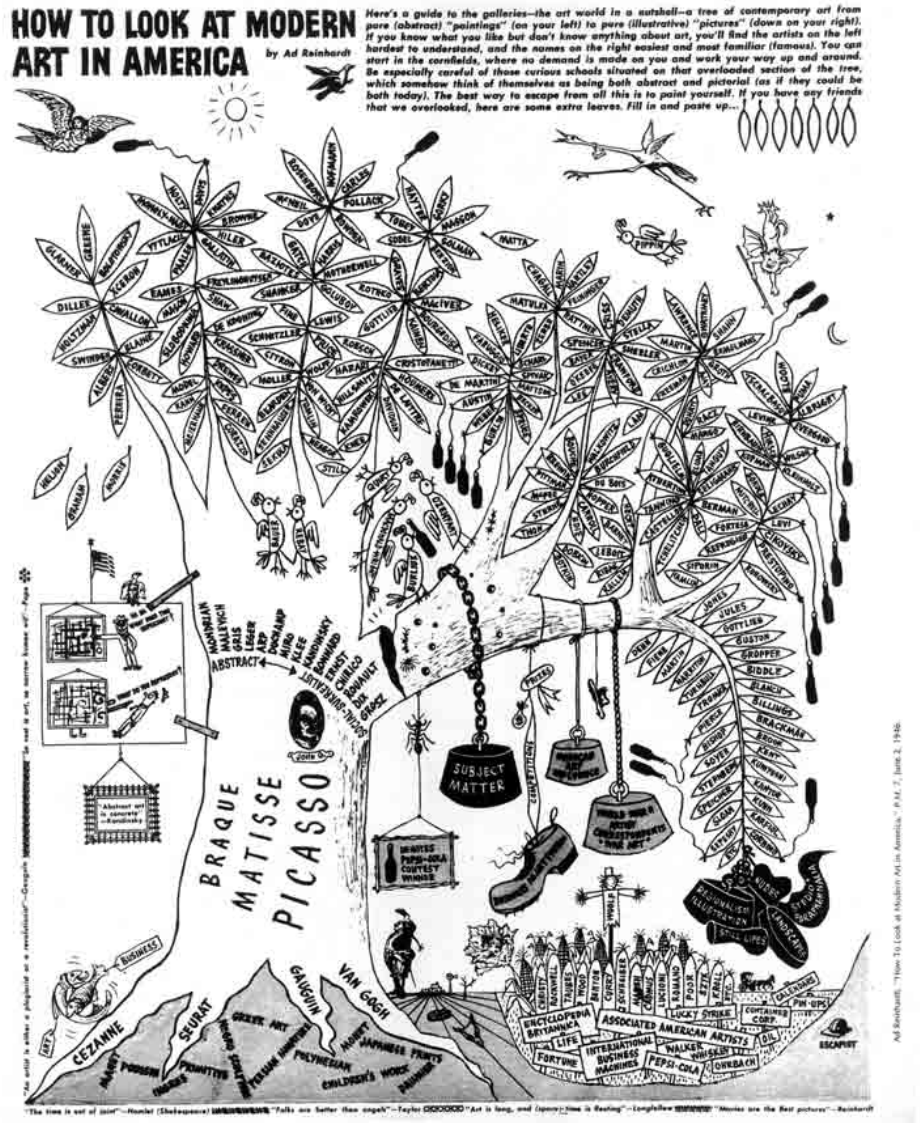


Figure 2-8: Taking spatial reasoning to the other extreme (Source: Ad Reinhardt, How to Look at Modern Art in America, 1946)

Similarly, this concept can be applied to presentations. We automatically realise that smaller things are less important than bigger things and that

things far apart are less likely to be related than things next to each other. While slideware applies this concept to a certain degree (e.g. font size decreases according to bullet list depth) this could be applied even further to make optimal use of our natural spatial reasoning. Figure 2-8 demonstrates an extreme example of such spatial reasoning. However, as explained in the next subsection, the space of a slide is bound and limited, which makes it hard to use it to its full potential.

Limited Working Memory

It is clear that humans have a limited working memory. With working memory we mean the memory we use to hold concepts or information while reasoning or calculating. For instance, when doing a complex calculation without paper or other tools, our working memory will be used to store things like intermediate results and remainders during the calculation process. Of course, the working memory is not limited to numbers but can also be used to store concepts, ideas and memories for use during reasoning. In 1956, an important paper on psychology was released by George A. Miller [33] on the topic of human capacity for processing information. From this paper, Miller's law was born: *the average human can hold 7 plus or minus 2 objects in his working memory.* When we look at current presentations, we can see that this is rarely taken into account. By spreading out content over many slides, the presenter is effectively requiring the audience to remember previous slides often beyond the capacity of an average working memory. One could add *reminders* to the slides, go back a few slides to show a concept again or verbally remind the audience, but in the end, these are coping mechanisms for a more fundamental problem.

As shown, the enforced hierarchical representation and linear traversal introduces some problems. We have highlighted that the damage can be limited manually, but one could state that these are just coping strategies for issues that should not be there. Also note that the manual "fixes" are the presentation creator's responsibility and thus they depend on the creator's presentation skills, which might not always be present.

2.4 Paper Emulation

The well-known presentation tools are all slide-based, but why is that? To find the answer to this question, we need to go back in time a little. As explained in Section 2.1, modern presentation tools evolved from overhead projectors and slide projectors but after so many years, the main concept has changed very little. Initially, presentation software was used to design slides that would eventually be transferred to a physical medium such as a plastic sheet or a photographic slide. As live projection from a computer

became possible, the intermediate medium was left out, but the tools are still very much editors for slides at their core, albeit virtual slides.

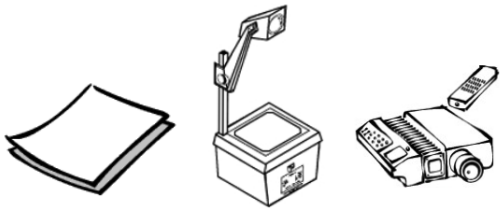


Figure 2-9: The evolution towards slideware

So, why slides? That is a good question. It is clear that the overhead projector was still based on transparent "paper" and it was in fact a way to project paper onto larger surfaces. The slide projector kept this method, only making the "paper" smaller and losing the ability to annotate. Digital presentation tools then kept the concept of these slides, so one could state that they are based on photographic slides, which are based on paper projection, which is based on paper. Therefore we can say that the digital slides that we know are nothing more than paper simulations. It might have seemed the logical thing to do at the time, but now we can ask ourselves "*why do we limit ourselves to the boundaries of paper?*". The idea that modern document formats are merely paper simulations is not new. The phenomenon has been described in detail by hypertext pioneer Ted Nelson [37] and it has been one of his life goals to provide a solution to this problem, for instance through the Xanadu project [35]:

> *Most people don't understand the logic of the concept: "What You See Is What You Get" is based on printing the document out ("get" means "get WHEN YOU PRINT IT OUT"). And that means a metaphysical shift: a document can only consist of what can be printed! This re-froze the computer document into a closed rectangular object which cannot be penetrated by outside markings (curtailing what you could do with paper). No marginal notes, no sticky notes, no crossouts, no insertions, no overlays, no highlighting—PAPER UNDER GLASS.*

This well-known quote by Nelson [37] is true for WYSIWYG[1] editors in general, but it applies particularly well to presentations too. As explained earlier, the presentation tools were initially designed to create slides to be *printed* to physical media. Even though slides can now be projected directly from a computer, the problem still remains: the tools are WYSIWYG

[1] WYSIWYG: What You See Is What You Get

editors that output fixed static content that maps easily to the concept of paper-based media. The big lesson that we can learn from Nelson's work is that computers introduced new ways of storing and working with information. This allows us to step away from information representation that is based on paper or similar physical media. By letting go of these formats, we can avoid some of their limitations and make better use of computers for information management, for instance through hypertext, transclusion or new ways of visualising information (e.g. by utilising the third spatial dimension). The exploration of new document formats is still a very active research field and much research has been done to identify the problems and provide solutions [19, 46, 47].

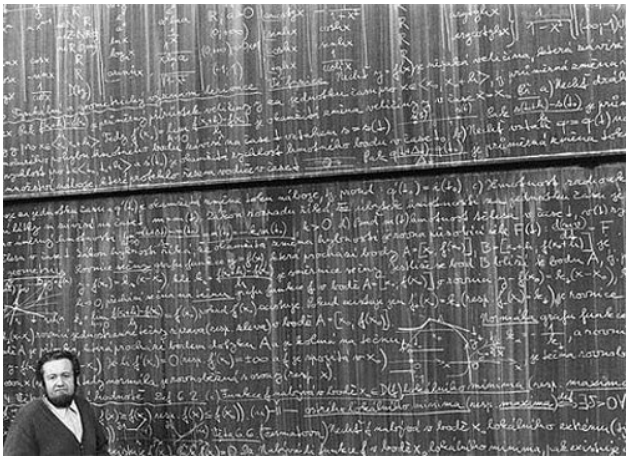


Figure 2-10: The resolution of a chalkboard

The most obvious limitation that digital slides have inherited from the physical media is that the available space is bounded and limited. As digital projectors have a limited resolution, and because the text has to be readable by the audience, there is very little space available for content. The user is forced to abbreviate sentences and to spread larger amounts of content over multiple slides. When we look back at the old days, when professors still used blackboards as their main teaching tools, we can ask ourselves whether digital slides are really that much of an improvement? Surely, they make better use of modern content such images, audio and video, but on the other hand, we gave up the immense resolution of the blackboard, the ability to annotate anything, and the ability to reference any data across the large 2D plane that is offered by a blackboard like the one shown in Figure 2-10.

2.5 Lack of Input Methods

As mentioned in the previous subsection, the ability to annotate anything on the fly was given up in favour of digital slides. Recent PowerPoint versions allow the user to draw on top of the slides with the mouse or with special pens that control the mouse cursor in order to "write", but they always force the presenter to be near the computer. Of course, there are special remote controls for presentations, but these are often used for nothing more than navigation. Digital pens such as the ones made by Anoto[1] allow for better remote interaction with a presentation but the integration with existing presentation tools is still lacking. For example, the innovative PaperPoint [48] presentation tool allows the user to make annotations while away from the computer and solves a lot of the navigational issues.

With new innovations every day, we are heading towards a future with many new input options such as multi-touch, 3D gestures or voice input. Some of these are available as 3rd party extensions of existing tools, but these tools are often limited to the functionality offered by the tool, which is designed with the keyboard or mouse in mind. Current tools could be improved by not designing them with keyboard or mouse input in mind, but by also providing an interface that is independent of the input device making it easier to add new input methods (e.g. multi-touch gestures or voice commands).

2.6 Distance Between Audience and Presenter

The evolution towards digital presentation tools has not been kind to the relationship between the presenter and audience either. During traditional lectures with paper handouts and chalkboards, it was feasible to stop the presenter or teacher to start a discussion. The media (paper and chalkboard) allowed for annotations and corrections during the session. Overhead projectors maintained many of these properties: the room was still well lit, interruptions were possible and annotations could be made on the plastic sheets for everyone to see. As mentioned by PowerPoint creator Robert Gaskins: *Transparencies were not a performance in and of themselves but a focus point* [15]. This changed when the slide projector was introduced. The photographic slides required a darkened room in order to be visible, making discussions more difficult and turning the presentation into a point of focus. The introduction of multimedia makes this even more true as modern content such as audio and video make interruptions totally impossible.

In the previous subsection, we discussed how a presenter is often forced to stay close to their computer in order to navigate through the presentation.

[1] http://www.anoto.com

This already causes a distance between the audience and the presenter. However, recent surveys show that presenters want to be close to the audience, facing the audience and optionally interacting with them [42]. Additionally, when talking about the distance between the audience and the presenter, we do not only mean the physical distance as described above. When sitting in an audience, the presenter is often seen as a sort of leader. Because presentations are frequently given by professors, managers or other superiors, the idea of the presenter leading the whole event is inherently present. The "humble" audience might see the presentation as the presenter showing off this knowledge, and therefore assume that it is okay if they do not understand what the presenter is trying to say. When we see a presentation as an event where knowledge is transferred from the presenter to the audience, this is most definitely not desired.

Several other issues with slideware worsen this phenomenon:

- Because navigation can only happen linearly, it is awkward for an audience member to ask to see a previous slide way back and it is generally "not done". Therefore, a listener might restrain from asking for a better explanation and just assume that the presenter is correct.
- The message that the presenter tries to bring across to the audience is often complex but has been boiled down dramatically to fit on a single slide. If this is done incorrectly, it might easily lead to confusion in the audience.
- Similarly, because a "big idea" has to be spread over multiple slides, an audience member does often not have the full view of the idea that is presented.

In contradiction, the opposite may happen if critical audience members are present. They might question the content and think that the presenter is hiding information or does not know what he is talking about. Of course, this influences the presenter's credibility.

If a presenter is serious about their task of transferring knowledge, they must ensure that the audience is following (e.g. through interaction with the audience). Tools should provide help on this aspect, for instance by providing easier navigation so that the presenter can better answer questions, or to have a mechanism that allows the presenter to navigate to detailed information only when required. We conclude that current tools are geared towards the presenter and not towards the audience.

2.7 Non-Content-Driven Slideware

2.7.1 Content Creation

When working with the classic slideware tools, we see that the applications are not geared towards content. Pretty much every button, menu or sidebar is dedicated to graphical settings such as fonts, colours, themes, animations and images. An example of an application geared towards content would be the LaTeX [28] typesetting system. In LaTeX the user concentrates on writing text whereas layout issues are handled by the application. Note that LaTeX offers a class specifically for creating presentations, called Beamer[1], which is a step in the right direction, which is discussed in Section 3.3.2.

Coming back to slideware tools, we see that the mass of graphical features has a bad influence on the presentation quality if used incorrectly as often seen. We have all attended presentations with images everywhere, text flying around and with weird text effects. Because the features are so prominently offered, inexperienced presenters may think they have to be used in order to create "real" presentations. This results in the general overuse of animations and clipart. Often these animations or images do not contribute to the transfer of the main idea and could be considered as plain noise. The abuse of these items results in a bad signal to noise ratio, making it harder for the audience to follow and learn. In the essay *The Cognitive Style of PowerPoint* by Edward Tufte [53] these items are called *phluff*. Tufte also introduces the idea that phluff causes a vicious spiral of negative influence: a user with little or mediocre content adds tons of phluff to fill in the empty space, which results in bad content, and finally in the need for more phluff. While this is debatable, we cannot ignore that the tools enforce the principle of *"quantity over quality"*, which is again strengthened by educational institutions and major companies. While the occasional use of silly images could be useful to keep the audience's attention and to keep their spirits high, it has been shown that images that are not related to the content have a negative influence on the learning experience [1, 22].

Other than these issues, we also notice that features to present specific information types are lacking or not present. Examples of these information types are long tables with lots of numbers, source code or external resources. Among other things, this makes these tools unsuitable for science or engineering, as illustrated later in Section 2.8. Content could also be made more interactive during the presentation itself. An example could be the interaction with a graph (e.g a supply/demand graph) to see how one function responds to the movement of another. In order to do so, input methods must evolve too in order to avoid similar issues as described in Section 2.5.

[1] https://bitbucket.org/rivanvx/beamer/wiki/Home

2.7.2 Managing Content

Even after creating the content, the problems are not over. Several new problems arise when one needs to modify, change or share presentations (e.g. for a different audience). First of all, as explained in Section 2.3.1, the fundamental computer layers do not work via association. Therefore when content has to be in multiple places at once, it almost always needs to be duplicated. Even though some filesystems support concepts such as symbolic links, there is no such thing on the document level. For instance, if we want to reuse a single slide from another presentation we need to copy it into our new presentation, duplicating the information. When we want the same data to be displayed in multiple places, we have no option but to duplicate it. This counts for both internal (within the same presentation) as well as external (in other presentations) data. Secondly, content is static. While we can of course add things like animations and movies, it is a fact that once the content is created, there is no way to dynamically adjust it to the audience. Let us explain the lack of dynamically adjusted content in an example.

Would it not be great if one could create a single presentation for multiple audiences and we could predefine which parts are designated for which audience? Before giving the presentation we would just specify the target audience and the presentation would automatically adapt. For example, a major company does a presentation at the end of the year, where they keep their employees and stakeholders up to date. It is obvious that the employees will be presented with more HR related topics, and will receive a virtual pat on the back, while stakeholders will probably be more interested in raw statistics and strategies. Even though some of the content is different, there will be a certain degree of overlap. Instead of having to create multiple presentations, these could be combined, and dynamically (optionally automatically) adjusted at run time.

Of course, the parameters for this kind of dynamic adjustment are not limited to the target audience. The presentation could dynamically adjust to other factors such as current time (e.g. other colours when it is dark outside), location (e.g. language) or the time that the presenter has for the presentation.

Lastly, we would like to point out that the current tools have no designated methods for things like user management and content sharing. Typically, one person makes the presentation and then shares it by email or other means. This could be done entirely different. Google Docs gets this right in the sense that presentations can be made in a collaborative environment and there is a form of version control which keeps track of which user changes what. It also facilitates the sharing of created content. What it does not do

however, is to provide means for reusing content. To elaborate, imagine that all the content we create is kept in a repository and that we (and others) could easily reuse certain blocks of data. When used in a company, over time the creation of a presentation could be as easy as puzzling together different pre-existing blocks, freeing more time to be spent on the new content.

2.8 NASA Use Case

All these potential problems may seem a little far fetched. The reader may think that these are irrelevant issues and mistakes that only inexperienced presenters make. This is however not always the case. We hope to illustrate this via a real example with a dramatic result.

Figure 2-11: Video footage from the Columbia lift off

The reader might have heard about the Columbia space shuttle. In 2003 the shuttle burned up during re-entry of the atmosphere. Both the shuttle and its crew were lost in this tragic accident. When the shuttle took off from earth, video footage showed that a piece of debris hit one of the wings during lift off. Even though the debris was guessed to be a piece of insulation foam, the damage could be significant as the impact speed was estimated at around 800 km/h. The shuttle made it to space safely, but it was clear there was potential damage, and there had to be an investigation to decide whether it was safe for the shuttle to make the trip back. A team of Boeing engineers was put to the task and a report was created in the form of 28 PowerPoint slides. The report was evaluated by NASA and the decision was made that there was insufficient evidence that the thermal protection tiles were damaged. We know how the story ends, so we know they were wrong, but let us take a closer look at the slides which caused NASA to think it was safe for the shuttle to return.

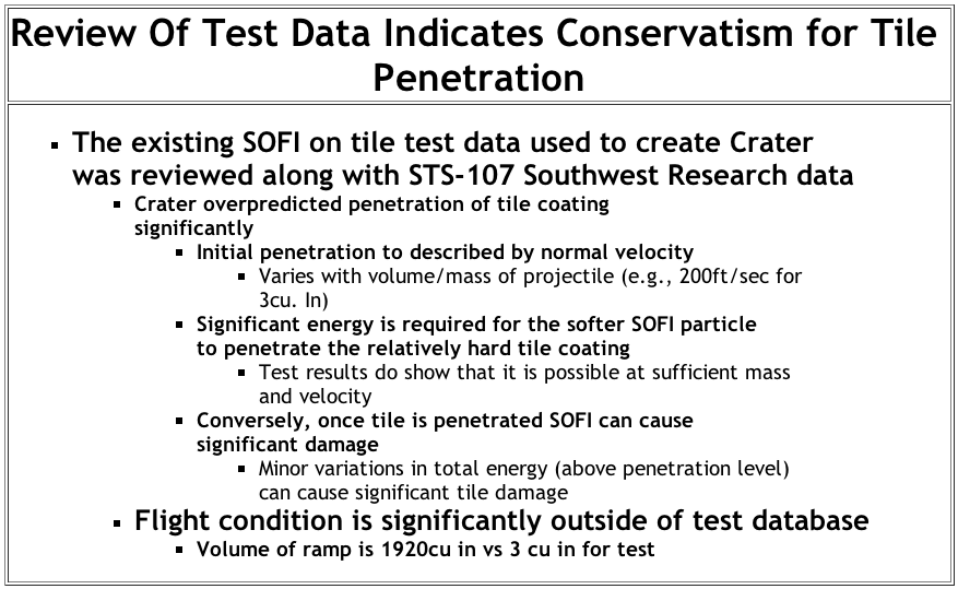


Figure 2-12: One of the key slides from the official Boeing report

Edward Tufte did an in-depth analysis of the key slides and identified a lot of issues [54], many of which we have already discussed earlier. For the full analysis, please refer to Tufte's work, but allow us to present a summary that shows how our previously presented issues surface in the real world.

- These 28 slides were the full report. There was no written text or detailed explanation to accompany the slides. Because of the limited space, sentences are extremely abbreviated and while they may make sense for engineers, this may not have been the case for the higher officials that had to make the final decision. For instance, the lowest bullet on the slide shown in Figure 2-12, which reads "*Volume of ramp*", actually means "*the estimated volume of foam debris that hit the wing*". Obviously this is a much clearer explanation, but there is no space on the slide, so details have to be cut away. This last bullet actually means that the foam debris is estimated to be 640 times larger than the volume they used in their simulation models. This may as well have been the most important line in the entire report but it means nothing for casual readers of this document.

- Throughout the slides, 6 different levels of hierarchy (bullet depths) are used. However, the depth of an item is not per se linked to how important it is. One would expect that higher levels are more important and deep levels are details. In these slides, the messages seem to differ depending on the level. The top levels seem to be rather optimistic, while the lowest levels carry details that seem to present contradicting evidence.

- The vague descriptor "*significant*" is used 5 times on the slide shown in Figure 2-12, but none of them refer to statistical significance, so the reader still has no idea whether it means that there might be a scratch in the paint layer, or if the wing is about to fall off.
- In the sentence "*it is possible*", "*it*" refers to the potential damage to the left wing. However this cannot be extracted from the slide. Again details are left out in order to save some space.

Long story short, the engineers had doubts as evidenced by internal mail from that time, but their concerns were lost in bureaucracy and in the flawed report. When the report traveled up in the company hierarchy, things were misinterpreted, changed and even more meaning was lost. Eventually this caused NASA to decide that it was safe enough for the shuttle to return. One of the worst parts is that a military satellite could have been used to photograph the shuttle for visual confirmation of any potential damage, but even this was deemed unnecessary based on the slides provided by Boeing.

Of course, this does not mean that the slides were the cause of the accident, but we can learn some valuable lessons from this event. PowerPoint, in its current form, is not a tool for science and engineering. It also shows that some content should not be minimised in order to compensate for restrictions that are enforced by slideware.

2.9 Coping Techniques

We have been using digital slides for 25 years now, so we had plenty of time to adapt, experiment and decide on good practices. During these 25 years, we have learned to live with the previously described issues and while it is still possible to create bad presentations, most of us know the pitfalls and hence know how to avoid them. Most of the "tricks" are obvious, such as not putting too much content on a slide, no unnecessary images and animations, or to help the audience to maintain an overview. We will not waste space here explaining these, as there are plenty other (better) resources on how to create good presentations, but we would like to point out some of the latest trends in presentation design.

We take a look at how "modern" slides are supposed to look. First, we look at a a popular blog called `presentationzen.com`. The author describes it as "*a blog on issues related to professional presentation design*" and provides some examples of bad slides and how to improve them. We will show some of these in the *Before* and *After* column respectively:

Source: http://www.slideshare.net/garr/sample-slides-by-garr-reynolds, by Garr Reynolds

The one thing that seems to stand out from the improved slide versions is that they are very minimalistic. Of course with a name like *presentation zen* we expect nothing less, but if we think about it, this trend is fairly dominant in other places, for example in Apple's WWDC[1] presentations and hardware announcements, as shown in Figure 2-13.

[1] https://developer.apple.com/wwdc/

Figure 2-13: Slides from Apple's June 2011 WWDC Event

Another example of this trend towards minimalistic slides are the slides shown in Figure 2-14, from an AMD presentation created to inform developers about OpenCL:

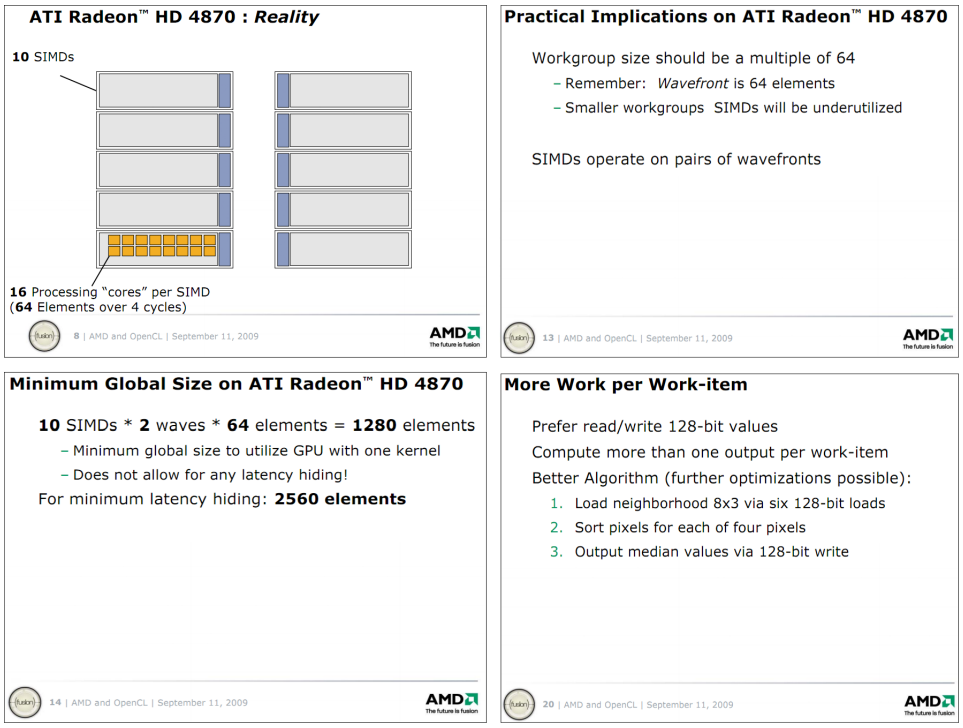


Figure 2-14: Slides from an AMD developer presentation

As demonstrated, we see that in order to cope with the limitations that current tools impose, users seem to have adapted by minimising the content displayed on each slide. This results in information loss which has to be counteracted with a better oral explanation. While this has proven to be a successful coping method, we must ask ourselves: if the tools did not have their current restrictions, would it still be necessary to limit the information presented on slides in such a way? Sometimes it is also not possible to minimise the content. There are plenty of situations where massive amounts of information have to be shown. The first example that comes to mind is in education, more particularly in science and engineering courses. One cannot simply present formulas, proofs or source code orally. There, another problem is that slides often form a major part (or sometimes the only part) of the study material offered by the teacher and therefore they cannot use a coping strategy such as the one illustrated above.

2.10 What Now?

In this chapter we have been quite critical about existing slideware tools, but hopefully the reader will also realise that a lot depends on the creator of the presentation. It is still possible to create good presentations with these tools, but good presentations require skill and a lot of extra attention. In the creation of a new tool, we not only want to fix the presented issues because they might cause bad presentations, but also to lower the skill level and time needed to create good presentations. Tackling these issues could potentially result in a better experience for the audience and presenter alike.

3
Towards Better Presentations

In Chapter 2, we have taken a critical stance towards the established presentation tools and we identified some core issues. We have seen that the established presentation tools are clinging to their roots (analog slides) but that these core paradigms may not be the most optimal in the modern computer era. As we have stated earlier, presentations are all about transferring knowledge to other people. This implies that a presentation tool can be regarded as a means for *managing and visualising information*. In this chapter, we step away from the classical presentation approaches and take a closer look at the state of the art techniques for managing and visualising information in general. We start by introducing some important concepts, followed by a selection of presentation-related tools that use these concepts in one way or another. By looking at presentations from a broader view, we can step away from the paradigms that have been enforced by slideware tools and effectively rethink the different steps involved with presentations.

3.1 Hypertext

When Vannevar Bush wrote his essay on the Memex [5], he first described what would later be named *hypertext*. In his essay, he introduces a device for storing documents together with a way of linking information together. The first actual implementation of such a system was introduced by Douglas Engelbart in 1960, by the name of NLS or oNLine System [11]. In 1965, the term hypertext was officially coined by Ted Nelson [34] and from there on it would form the basis for the World Wide Web as we know it. Ted Nelson described it as "*non-sequential writing*" but there is no exact definition,

and definitions presented by different authors may carry subtle differences. However, the main idea is easy to define. The following extract is a short but concise definition from The Electronic Labyrinth [25].

> "*Hypertext is the presentation of information as a linked network of nodes which readers are free to navigate in a non-linear fashion. It allows for multiple authors, a blurring of the author and reader functions, extended works with diffuse boundaries, and multiple reading paths.* "

This definition shows us that an approach similar to hypertext might be a good base for a presentation tool. The definition above mentions solutions to some of the core problems we defined earlier: non-linear traversal, multiple authors, diffuse boundaries and multiple navigation paths. However, hypertext is not limited to this basic idea and over time extensions were made that could also be very helpful to our cause.

3.1.1 Semantic links

When we take the world wide web as a hypertext application, we see that regular navigational links (hyperlinks) are unidirectional: an HTML anchor link (`<a href="target">`) link goes from page A to page B, but not the other way around. However, hyperlinks are not per definition unidirectional. They can be bidirectional and even multidirectional as a link can have multiple targets. On top of that, we see that these links are not typed. This means that no additional information can be derived regarding the relation between the source and the target. In HTML, an anchor link is interpreted as "*navigate from the current page to the link target*" without further more. This is also not a restriction of hypertext as it allows typed links too. HTML implements some typed links via the `rel` (forward relationship) and `rev` (reverse relationship) attributes. The following line is a common use of such a link:

```
1 <link href="style.css" rel="stylesheet" type="text/css">
```

In this example the `rel` attribute is used to indicate that the target of the link is a stylesheet, which is a hint for the browser on how to interpret it. HTML includes a predefined set of these types, but it is out of the scope of this thesis to describe them all. Now that we have explained typed links, we introduce the topic of this section, *semantic links*. Semantic links are links which describe the character of the relation between the connected components. Let us illustrate this with a simple example:
This graph is quite simple with only five nodes and 5 links, but it holds a lot of information. We can derive that `John Doe` lives in Brussels, he works for the VUB and he has two sons. Links are multidirectional and more importantly,

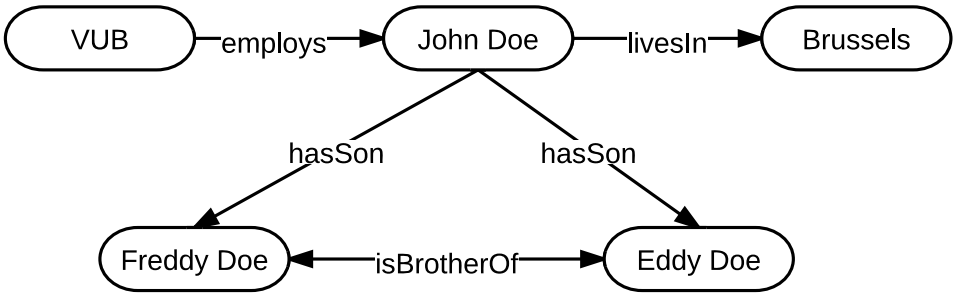


Figure 3-1: Objects connected with semantic links

they carry semantic information describing the relationship between nodes. Essentially one could say that it gives links a meaning. If these semantics are standardised and limited to a certain vocabulary they become machine readable. The benefit of having machine-readable relationships is that it allows easy querying by humans and machines alike. For instance, the query "*Give me all employees of the VUB who have children*" is a trivial lookup as the defined relationships map perfectly on this query in natural language. It would even be possible to have a computer do some reasoning over the data. If enough concepts are defined within the relevant domain a computer could for instance derive the `isBrotherOf` relation automatically. However, here we are crossing the boundary between simple semantic links and ontologies, which are out of the scope of this section.

3.1.2 Spatial Hypertext

When taking a pure hyperspace approach, links between information are *explicit*, whether they are typed or not. When taking the World Wide Web as an example, a hyperlink is a strict relation between two pages. It is explicit because both the source and target are fixed and there is no ambiguity as to what degree they are related, they simply are. The goal of spatial hypertext is to replace explicit links by implicit spatial relationships. By spatially arranging objects and by giving them different colours and shapes, we can represent relationships such hierarchies, groups, ordering and similarity, as shown in the example highlighted in Figure 3-2.
Even though the relationships are not explicitly defined, we can still derive a lot from the spatial arrangement. All elements in the `Characteristics` section have the same size and colour so we can assume this is a list of equal characteristics. In the `Weapons` section we see that there are two types of stakes, wooden and silver ones. The `Movies` section presents a grid of different movies relevant to the topic. Even though we cannot see the individual movie titles, we notice that there is a spacing between the second and third row and assume that the last row is somehow different from the first two.

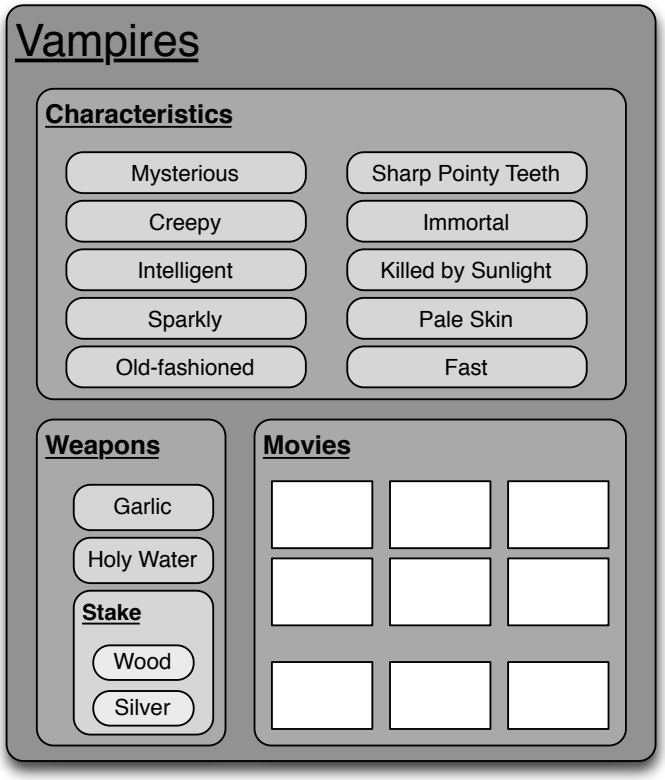


Figure 3-2: Spatial hypertext example

It seems that humans understand spatial relationships intuitively. Even without explaining the conventions used in the arrangement, a first-time viewer can derive almost all relationships as if they were defined explicitly. However, the relationships are not clearly typed. We can see that the third row of movies might be different from the first rows but we do not know why or how. For this reason, implicit linking via spatial hypertext is ideal for fuzzily relating items. The relations and orders are vague but this is not per se a bad thing. When visualising large amounts of explicitly linked information, it can quickly become a mess or a "spaghetti" of links, which is not a desirable situation for humans [10]. For this reason, spatial hypertext is well suited for visualising large amounts of data with n-ary relationships.

We can ask ourselves, why are humans so good at this and what are the basic principles behind this? The answer to this is known and is studied in the field of *Gestalt psychology*. Gestalt is the German word for form or shape and thus Gestalt psychology is the study of how the human mind works regarding shapes and forms. In particular, the studies try to find out how our mind visually attempts to create order and structure out of the seemingly

chaotic and disconnected pieces of information. These studies resulted in a set of laws or principles called the *Gestalt principles*. These principles were first introduced by Max Wertheimer [55] in 1923 and were later refined by Wolfgang Köhler [27] (1929) and Kurt Koffka [26] (1935). Sometimes the principles are named differently between works, or sometimes they are split up in different ways, but we will list the general ideas applicable to hypertext and illustrate them with an example. The images are part of the book *User-Centered Web Site Development* [32], where the Gestalt principles are illustrated in the content of website development.

Similarity Elements with the same shape or colour seem to belong together as illustrated in Figure 3-3.

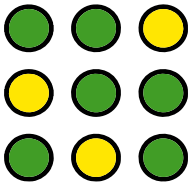


Figure 3-3: The principle of similarity

Continuation Our mind organises elements by the flow of the elements. In the following examples shown in Figure 3-4, we see lines going from A to B and from C to D, but never from A to D or C to B. In a similar fashion we see two rows of circles and not circles arranged in "L" shapes.

Figure 3-4: The principle of continuation

Closure Our mind automatically fills in gaps to form shapes that it recognises. In the example shown in Figure 3-5, there are three cases of images that are technically not circular or round, yet we recognise them as circles.

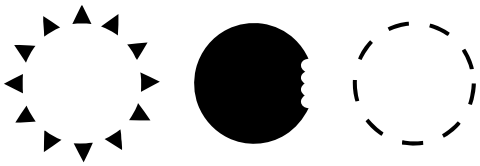


Figure 3-5: The principle of closure

Proximity Elements that are close to each other are interpreted as groups. In the example outlined in Figure 3-6, we see three groups of lines and three groups of circles.

Figure 3-6: The principle of proximity

Figure and Ground Our mind automatically distinguishes a figure from its background based on a number of variables such as contrast, colour, shape or size. Note that this segmentation can change depending on the focus of your eyes. For instance, in the example shown in Figure 3-7, one could see a white square (the figure) on a blue circle (the background) or a blue circle with a hole (the figure) on a white background.

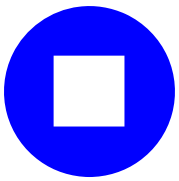


Figure 3-7: The principle of figure and ground

Symmetry We see regions bounded by symmetrical borders as coherent figures. Our experience and expectations cause us to see groups of things. In Figure 3-8, we see two triangles and three groups bounded by brackets.

Figure 3-8: The principle of symmetry

Because we all apply these principles subconsciously, we all understand the relations implied by spatial hypertext intuitively. Together with the fact that spatial hypertext is well suited to visualise n-ary relationships with minimal clutter, it would seem that spatial hypertext can be used successfully as an aid to visualise hypermedia. In fact, we are not the first to come to this conclusion, as plenty of research has been done around spatial hypertext [31, 44], resulting in practical hypermedia applications [19, 24, 41].

3.1.3 Transclusion

When we want content to be in multiple places, we often have to duplicate (copy) it. This is true for almost all common document formats. Consider the situation where you want to reuse a slide from one of your old presentations: you have to copy it into your new presentation. As explained by Vannevar Bush [5], this is because computers often use hierarchies for storing information, from the filesystem level down to the document format. Duplication is a direct consequence of such a representation because a hierarchical structure such as a tree does not allow loops to be formed, which are needed for inclusion by reference. However, in hypertext systems we step away from hierarchical representations and since links are multidirectional, we now have a graph-like structure for storing and relating information.

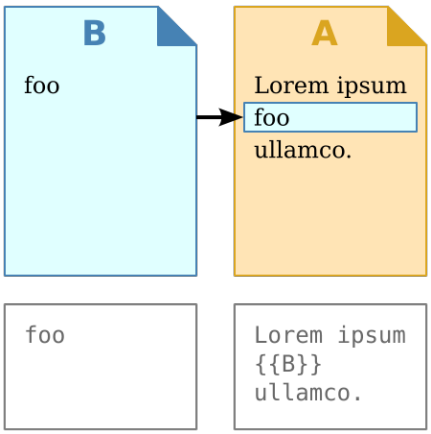


Figure 3-9: A visualisation of transclusion
Source: http://en.wikipedia.org/wiki/Transclusion

In one of his many works on hypertext [36], Ted Nelson describes the concept of *transclusion* or *inclusion by reference* in the context of hypertext which is visualised in Figure 3-9. What this means is that hypertext allows us to refer to other content (even external content) for inclusion at viewtime. An example of this is HTML's inline links such as this image tag:

```
1 <img src="http://www.example.com/picture.jpg" />
```

This line includes an image in the HTML document but the image does not need to be in the document or even on the same server. When the page is viewed, the image is retrieved from the external source and displayed as if it were part of the document. In HTML, transclusion is limited to a few formats but in a hypertext system it could be possible to include any type of data that is supported by the system. Note that transclusion is not limited to entire documents or files. By using so-called *selectors* to specify

parts of a document, it is very well possible to refer to specific pieces of information within documents. Examples of a selector include a region in an image, text that matches a specific regular expression, line numbers or, for instance, a time interval in a video. Some documents have trivial selectors (e.g. plain text or images) while other formats are more difficult to access (e.g. the proprietary PowerPoint format). Tools such as regular expressions or the XPointer [9] language can function as selectors in more complicated document formats.

3.2 Zoomable User Interfaces

In short, a *zoomable user interface*, sometimes also called a zooming user interface or ZUI, is a graphical interface where the user is changes the scale of the view in order to see more or less details of the presented information. One of the earlier attempts of using a ZUI for a computer interface is the Pad [41] project which triggered later projects such as Pad++ [3] and Piccolo [2].However, there is no need to go into much detail as almost everyone has used ZUIs before.
Examples of popular applications with a ZUI are for instance Google Maps[1], Google Earth[2] or Prezi. The benefit of a ZUI is that it can be used to display and navigate large amounts of information in a limited space. This makes ZUIs exceptionally well suited for mobile devices or regular computer screens where space is limited. Another reason why ZUIs are good for large amounts of data is that the visualised information can be adapted to the level of zoom. At a high level of zoom, the information can be drawn very simplistic, but details can be added as the user is zooming in. This makes a ZUI less cluttered at higher levels and it is also beneficial for giving overviews. The user can then decide for themselves where they want to see more information. Google Maps is a good example of a ZUI because it applies all these principles: at the higher levels one can see only the names of the countries/cities but as we zoom in closer, we are given the names of the big streets, then all the streets and eventually small landmarks such as popular tourist attractions and shops as shown in Figure 3-10.

3.3 Alternative Tools

In the previous chapter, we talked about the common presentation tools that are sometimes seen a standard in the domain of presentations. We have discussed their main shortcomings but we are of course not the first to notice some of these limitations. Next to the common tools, there are other tools which differentiate themselves by offering solutions for some of

[1] https://maps.google.com/
[2] http://www.google.com/earth/index.html

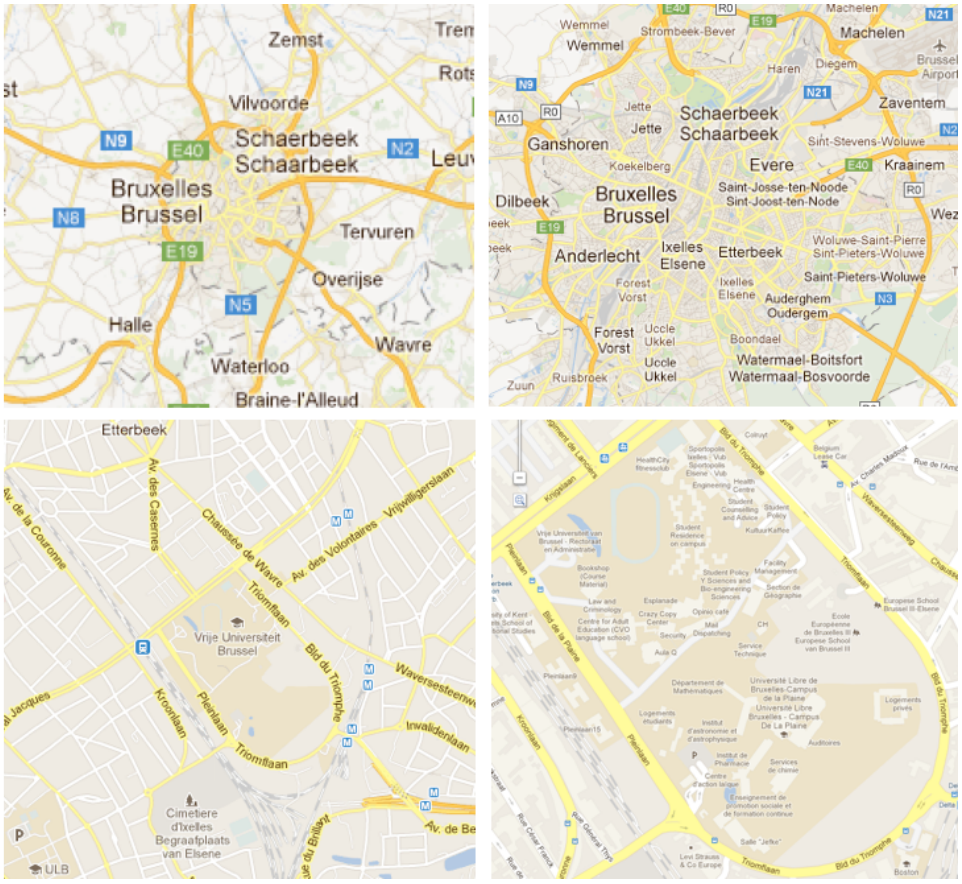


Figure 3-10: Google Maps ZUI

the limitations in existing slideware. In this section, we take a closer look at these "alternative" tools and discuss the features that make them different from the regular tools. However, some of these radically different approaches introduce new problems and we include a small review for each tool. At the end of this chapter, we recapitulate on new issues that these tools might introduce.

3.3.1 Prezi

Prezi[1] is one of the rare radically different tools which has become somewhat known in an environment that is dominated by the classical slideware tools. The thing that makes it so different is that it steps away from the slide paradigm. Instead of restricting the available space, Prezi gives the user an *infinite canvas* to work with as illustrated in Figure 3-11. The users lay out their content on the canvas and can then define a path between the content

[1] http://prezi.com

in order to define an order in the presentation of the information. On top of panning and rotating the canvas, Prezi allows content to be scaled to arbitrary sizes. By zooming in and out, Prezi can present information at any depth. This allows users to employ *spatial reasoning* in their layout. For instance, important concepts can be represented as large objects while details of lesser importance can be very small. This makes Prezi a very good example of a zooming user interface (ZUI). On top of fixed navigation paths, Prezi also allows *free navigation*, making it efficient for users to navigate to any part of the presentation. Prezi is a web-based tool that runs in any modern browser, while the underlying Adobe Flash platform provides a smooth experience. Next to online viewing, created content is also downloadable as a package for offline viewing. This package contains the Flash files together with an executable that will run the presentation.

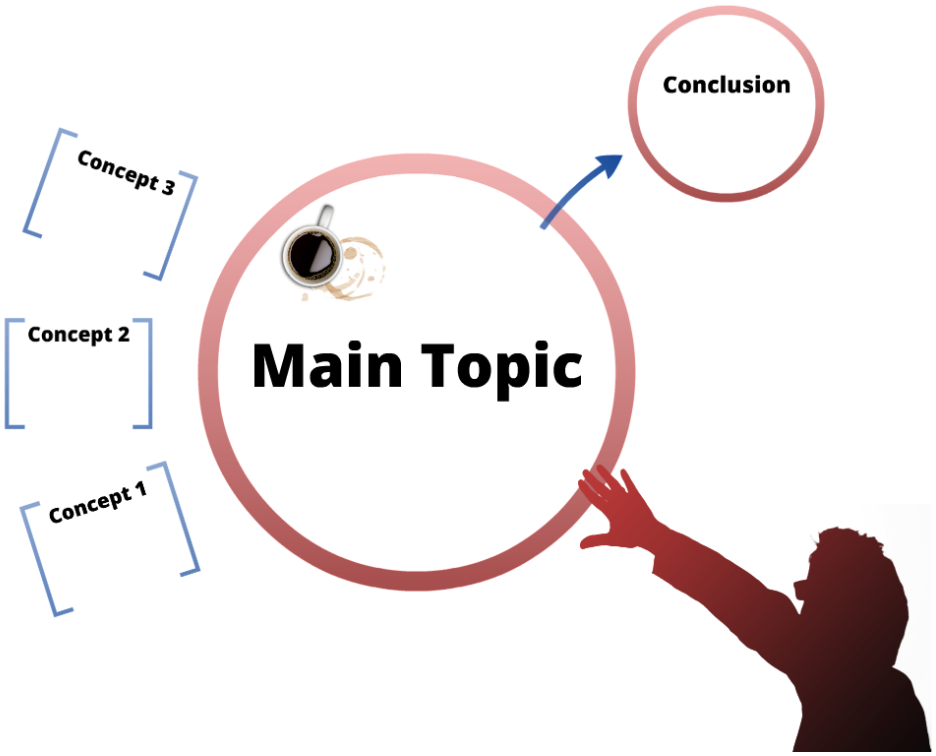


Figure 3-11: A typical Prezi presentation

This different approach to visualising information is very beneficial for some types of information because it allows presenters to make use of spatial reasoning. On top of that it makes navigation very easy and efficient. There is no support for internal or external links, but the free navigation makes up for it for the most part. Prezi supports a lot of content types including images, YouTube videos and diagrams. A notable feature is the ability to

import slides from PowerPoint. However, this type of presentation comes at a certain cost. For the creator of such a presentation, *most time is spent on the layout and visualisation* of the data and not on the content itself. Users have to lay out everything manually; dragging, dropping and resizing content until they are pleased with the result. Creating presentations in the Prezi format is time consuming and the tool does not always make it easy to achieve the desired results.

3.3.2 Beamer

Beamer is an extension for LaTeX, which is provided as a document class for usage within the regular LaTeX toolset. The main ideology of Beamer is the same as for the LaTeX language: place the *focus on content, not on presentation*. This is achieved by providing presentation specific structures on top of the regular document structures. For instance, LaTeX provides document structures such as chapters and sections whereas Beamer adds structures such as *frames* which can be seen as a slide.

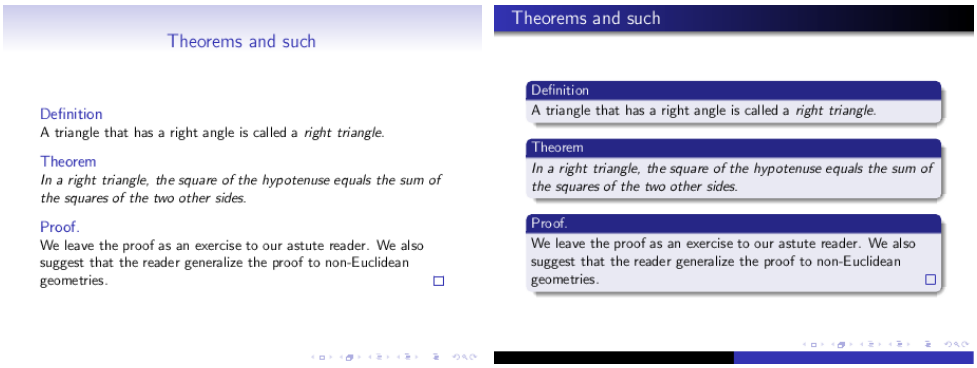
```
1  \usetheme{Singapore}
2  \begin{frame}{Theorems and Such}
3
4  \begin{definition}
5    A triangle that has a right angle is called
6    a \emph{right triangle}.
7  \end{definition}
8
9  \begin{theorem}
10   In a right triangle, the square of the hypotenuse
11   equals the sum of the squares of the two other sides.
12 \end{theorem}
13
14 \begin{proof}
15   We leave the proof as an exercise to our astute reader.
16   We also suggest that the reader generalises the proof to
17   non-Euclidean geometries.
18 \end{proof}
19
20 \end{frame}
```

Listing 3.1: Sample Beamer code

Due to the fact that Beamer is built on top of LaTeX, one can use all the regular LaTeXfeatures within these frames. This allows users to quickly add structures like bullet lists (`itemize` structures), images or mathematical formulas. The tool then outputs the presentation as a PDF document using themes to guide the visualisation of the content. Listing 3.1 illustrates

this combination of regular LaTeXfeatures and functionality offered by the Beamer extension, by making use of the theorem structure within a Beamer slide.

Note that the only place where we deal with the presentation visualisation is on the first line where we specify a theme. The slide is then styled according to the theme without any further user intervention. Figure 3-12 shows the output generated for the code in Listing 3.1 for two different themes.



(a) "Singapore" Theme (b) "Warsaw" Theme

Figure 3-12

This small example shows us that the power of Beamer lies in the clear separation of content and presentation. One can easily create and reuse customised themes and the visualisation of slides can be easily changed by editing the arguments of a single command in the LaTeX code.

In addition to that, Beamer offers two other notable features. As it is based on a LaTeX, there is support for hyperlinks allowing users to create text links and buttons in order to jump between slides. Additionally, Beamer introduces the concept of *overlays* which allows users to display content incrementally on a slide (e.g. in order to pose a question on a slide and have the answer revealed at the next step in the presentation without the need to create a new slide).

The downside is that LaTeX it is not trivial to use for users who never had to write any code before. The slides have to be created through code as shown earlier in Listing 3.1, and there exists no WYSIWYG editor as it would contradict the core principles of LaTeX. The average person is used to user-friendly WYSIWYG editors such as Microsoft Word or PowerPoint, and therefore the very thought of having to manually write code is going to scare away the majority of potential users.

3.3.3 Fly

FLY [30] is another tool that tries to step away from spatial restrictions such as slides. Similar to Prezi, content is laid out on a canvas and navigation is done through panning and zooming as shown in Figure 3-13. As opposed to Prezi, it is possible to define multiple paths through the same document. Free navigation is also possible at any time making it easy for presenters to answer questions. A major difference to Prezi is that Fly was not as commercially oriented and remained a research project in an academic setting while Prezi was meant to be a commercial product from the start. On top of that, Prezi (then called ZuiPrezi) was actually first with the idea of using a ZUI in a presentation tool. For these reasons, Prezi has had more time and money invested into it which resulted in a more developed and more user-friendly product than Fly.

Figure 3-13: The FLY tool

DragonFly [8] is an extension of Fly, which allows users to use a Fly presentation as a navigation control for a video. For instance, a student can navigate to a particular topic in a Fly presentation in order to access the relevant part in a lecture recording.

3.3.4 CounterPoint

CounterPoint [16] is another tool in the category of zooming presentation tools. We included it in the list, because it is somewhat different from Prezi

and Fly. CounterPoint is a plug-in for PowerPoint that allows the user to *spatially arrange PowerPoint slides* on a canvas, as outlined in Figure 3-14, and again makes use of panning and zooming. Like the other tools, one can then define a navigation path or use free navigation. You could say that this tool makes a compromise between the use of slides and free spatial layouts.

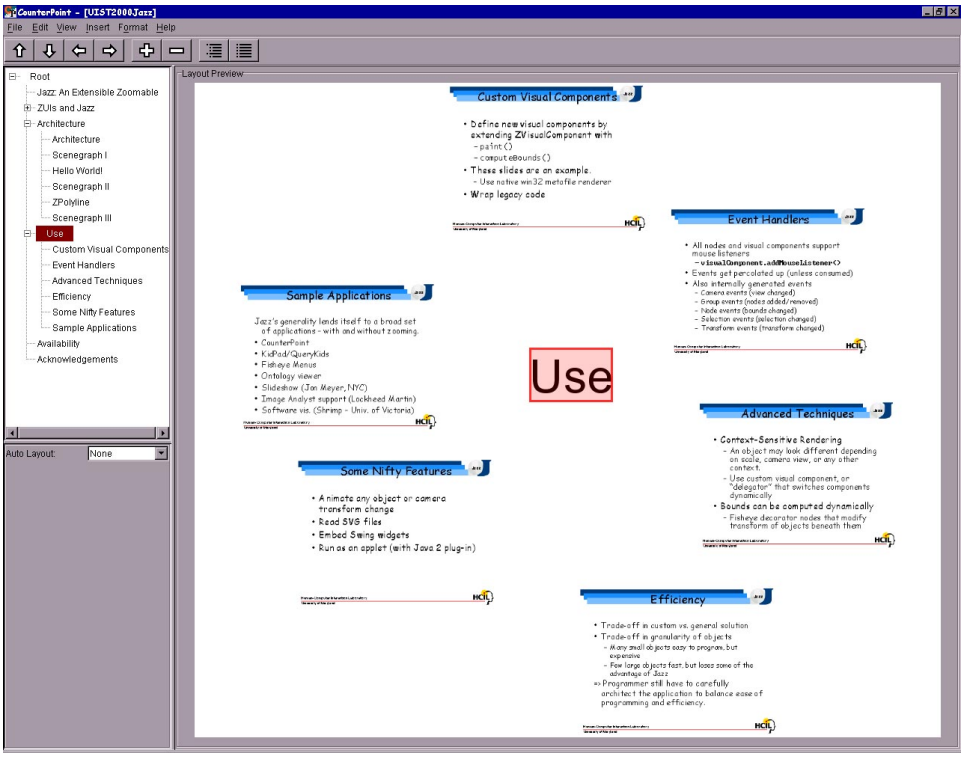


Figure 3-14: The CounterPoint tool

3.3.5 SlideRocket

SlideRocket[1] is a commercial product with a *focus on dynamic multimedia-rich presentations*. The resulting presentations are remarkably close to the multimedia shows discussed in Section 2.1 and can almost be seen as a sort of movie. The presentations have a lot of visual effects to dynamically fade in or move elements around and SlideRocket presentations are often accompanied by an audio track and/or audio effects. For examples of these multimedia-rich presentations please refer to the showcase section on the SlideRocket website.

The tool itself is an online tool and special care has been taken to also make the tool accessible via mobile devices. A notable feature is that there is

[1] http://www.sliderocket.com

support for *collaboration* meaning that one can work together with other people to create presentations or share and reuse slides from a common repository. This includes an automatic versioning system and a form of transclusion, or content inheritance as they call it. SlideRocket presentations allow a particularly broad range of content types ranging from the usual images and videos to live information feeds such as online spreadsheets or stock feeds. The tool also offers the possibility to import slides from popular presentation formats such as PowerPoint or Google Docs presentations.

3.3.6 SlideShare

SlideShare[1] is not so much a presentation authoring tool but rather a platform for *sharing and distributing presentations*. To put it crudely, SlideShare can be seen as the "YouTube" of presentations. The site allows visitors to share, view and search presentations of all kinds.

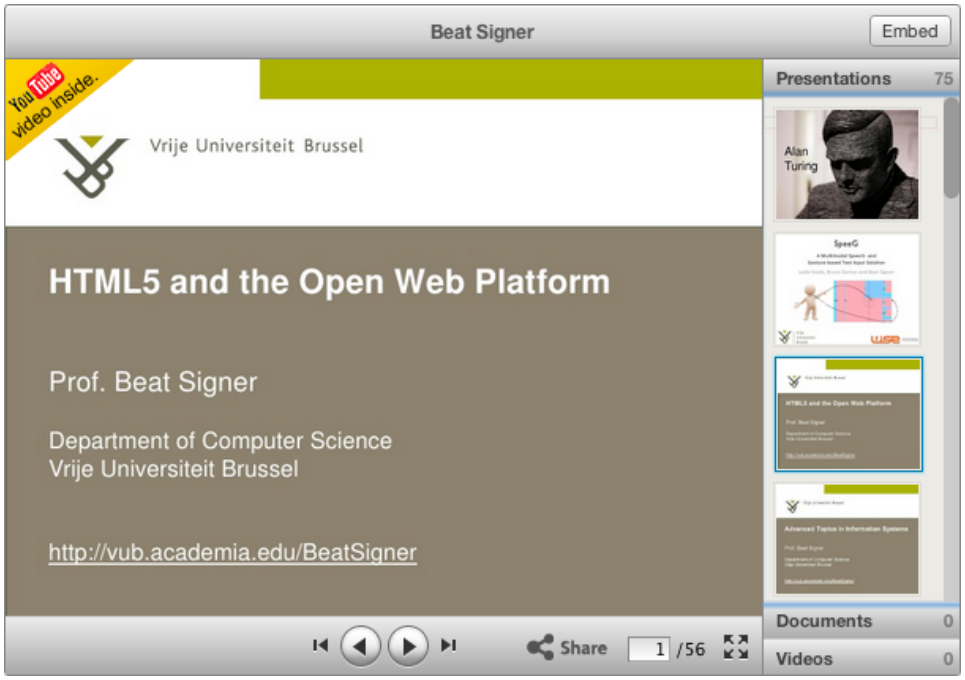


Figure 3-15: The SlideShare embedable document player

Presentations are offered through their HTML5-based "player" which is shown in Figure 3-15, making it easy for people to navigate through presentations regardless of their original format. This player can even be embedded in external web pages similar to YouTube videos. Registered users have a profile that lists their contributions and SlideShare offers some social features

[1] http://www.slideshare.net

such as subscribing (following) to other users in order to be notified of new publications. Users can upload their presentations in almost any format (e.g. PowerPoint, OpenOffice or PDF), which makes it very easy to place content on SlideShare. Using SlideShare has the benefit that it makes content easier to find for others, and sharing a presentation becomes as simple as handing out the corresponding SlideShare link.

3.3.7 Google Docs

Google Docs, which is shown in Figure 3-16, is an online office suite that is offered for free by Google. The suite contains tools for all typical document types such as spreadsheets, diagrams, text and presentations.

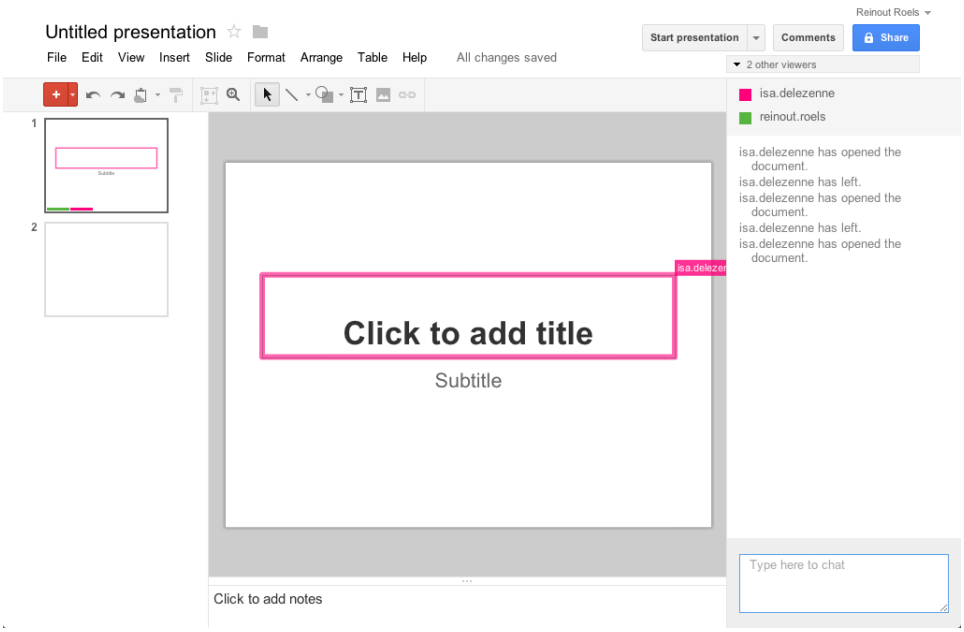


Figure 3-16: A collaboration session in Google Docs

While the tool itself is quite standard when compared to other slideware tools, the presentation tool has two notable features. First of all, the tool is used entirely online. A presentation is created, stored and delivered using any modern internet browser. Technology-wise, all of this is achieved via modern HTML5 so that no additional plug-ins or settings are required which ensures that it will run "out of the box" on the majority of devices. It is also possible to export a presentation for offline usage, with support for popular formats such as PDF, PowerPoint or PNG. The second notable feature is that *collaboration* is an important aspect of the Google Docs suite. It is very easy to share documents between users but the most impressive part is that it is possible to work on the same document with multiple people at

the same time. Changes made in the document are propagated to the other users almost in real time and synchronisation issues are minimised. Every user is given a coloured cursor and users can see who is working on which part of the document. Additional collaboration tools like a chat system and basic version control make collaboration easy and usable for any type of user.

3.3.8 PaperPoint

PaperPoint [48] is a tool which is based on *interactive paper* technology [50]. Before going into the details of PaperPoint, we will first introduce the idea of interactive paper in general. The major goal of interactive paper is to track user actions on physical paper as shown in Figure 3-17. This is achieved via spacial digital pens as, for instance, manufactured by LiveScribe[1] or Anoto. A digital pen outputs ink like any regular pen but in addition, the pen is able to record the strokes together with their exact positions on the paper. This is possible because the specially printed paper contains a background pattern of nearly invisible dots. An optical sensor in the pen is able to detect patterns in these dots which map to specific locations on the paper. This allows the pen to remember everything that is written or drawn, together with the exact location.

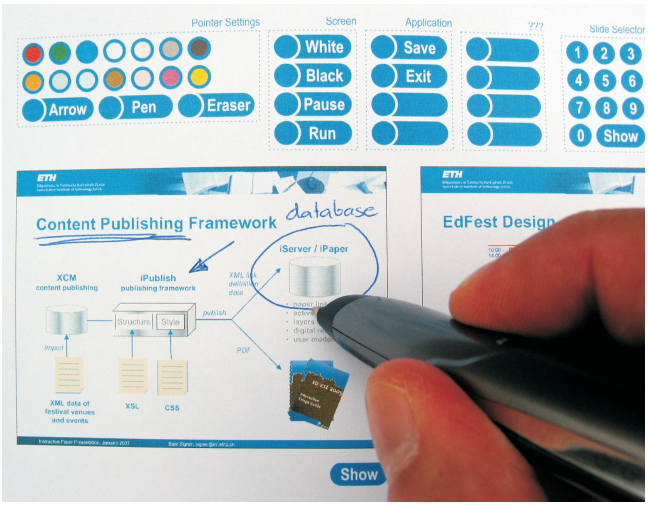


Figure 3-17: Usage of PaperPoint

Some pen models allow direct streaming of the input while others merely act as recorders that need to be manually synchronised with a computer to retrieve the data. Next to the obvious recording possibilities, another great advantage of the streaming models is that a computer can process incoming events in real time. This opens new possibilities, as it is now possible to

[1] http://www.livescribe.com

Figure 3-18: A PaperPoint handout sheet

simulate buttons, sliders and other interactive components on paper. This allows an interactive paper solution to be used as an input mechanism for other applications.

One possible use for interactive paper is to drive a presentation and that is exactly what PaperPoint does. The underlying technology that makes this possible is iServer [49, 51], a cross-media link service. In the iPaper plug-in [39, 45] the iServer platform has been extended with a framework for interactive paper. This framework allows developers to easily create active paper areas (such as buttons) and link them with digital information or services. PaperPoint uses this framework to link interactive paper with a presentation tool. At the moment of writing PaperPoint focusses on controlling PowerPoint presentations as it is the most widely used tool. However, PaperPoint offers a presentation interface and other presentation tools, including our new MindXpres solution presented later in this thesis

could be integrated in the future. Figure 3-17 shows a typical PaperPoint sheet with some thumbnails combined with additional interactive elements such as paper-based buttons which are used for *navigation and annotation.*

The usage of PaperPoint is very intuitive and requires little to no training. One can simply "tap" a button to interact with it and annotation is as simple as writing or drawing on the slide thumbnail. One big advantage of PaperPoint is that it frees the presenter from his computer and allows him to venture as far as the pen's communication module will allow him to. Another advantage is that it helps to break the linearity of PowerPoint presentations. The presenters now have multiple slide thumbnails on a single PaperPoint handout page which allows them to easily find and navigate to a particular slide. While one can still follow the linear sequence of slides defined by PowerPoint via the `Prev` and `next` buttons shown in Figure 3-18, it is now much easier to deviate from this predefined path on demand. Finally, PaperPoint allows real-time annotations in a very intuitive way. The handout sheet contains some paper-based buttons to select different annotations tools (pen, eraser or arrow tool), or to select a colour for the current tool. In Section 2.1, we seen how live annotation was unrightfully removed from presentation tools as they evolved, simply because the newer technology did not allow it. PaperPoint uses modern technology to bring back the long lost feature of live annotation to presentations, which is a major step towards making presentations interactive again.

3.4 Overview

In the last section, we have detailed the most important tools that can be considered to be different from the established slideware tools. Each of these tools has its strengths, but sometimes also comes with other limitations. To recapitulate, we would like to present a short overview of these tools depicting their main strengths. The tools are judged based on the following criteria:

- Spatial Reasoning: Does the tool step away from the slide paradigm and does it utilise spatial reasoning?
- Focus on Content: Is the focus during the creation of a presentation on the content , or on the visualisation?
- Input: Does the tool offer new means of input for steering the presentation?
- Sharing: Are there easy ways to share a presentation other than manually sending files around?

- Collaboration: Is it possible to work together with other people on the same content?
- Portability: Does the presentation tool have strict software or hardware requirements?
- Transclusion: Does the tool allow linking to content by reference?
- ZUI: Can the tool visualise content via a zoomable user interface?

	Spatial Reasoning	Focus on Content	Input	Sharing
Prezi	✓			✓
Beamer		✓		
Fly	✓			
CounterPoint	✓			
SlideRocket				✓
SlideShare				✓
Google Docs				✓
PaperPoint			✓	

	Portability	Transclusion	ZUI	Collaboration
Prezi			✓	
Beamer				
Fly			✓	
CounterPoint			✓	
SlideRocket	✓	✓		✓
SlideShare	✓			
Google Docs	✓			✓
PaperPoint				

As we can see, each of these tools addresses some of the limitations discussed in Chapter 2. However, each of these solutions focuses only on specific aspects so there is no single "perfect tool". When picking a tool, we often have to weigh the benefits against each other and pick the one most relevant to our way of working.

3.5 Newly Introduced Issues

The tools discussed earlier in this chapter deviate from the common slideware tools in one way or another, often improving one or more issues that are present in common slideware tools. However, it should be noted that these new approaches often introduce new pitfalls. In this section, we list some of the new issues that can arise when applying these new ideas to real-life presentations.

3.5.1 Excessive Animation

Zoomable user interfaces like Prezi or Fly step away from slide-based representations and use an infinite canvas to lay out the information. Navigation is done by panning, rotating and zooming in order to bring the desired information to the centre of the screen. This can work well when the transitions are fluent and subtle, which is not always the case. Fast zooming combined with wild panning or rotation have some important negative consequences to consider. First of all, it interrupts the audience's focus on the topic which is certainly not desired. Secondly, these wild animations can make the presentation feel like a roller-coaster ride causing confusion or dizziness in extreme cases. Finally, if the audience is not able to mentally process the origin and target of the animation, this stops being a case of spatial reasoning and the purpose of a zoomable user interface becomes lost.

3.5.2 Map Shock

Canvas-based visualisations such as used by Prezi and Fly can be used to present a lot of information in a single view. When this is taken to the extreme and when presenters use it to bombard the audience with too much information, this can cause a so called *map shock*. This means that the user feels overwhelmed by the amount of information that is being presented and will stop trying to understand what is going on. Of course, this undesired effect This is never a desired effect should be avoided. If unavoidable, incremental displaying of the information could be used to lower the negative impact.

3.5.3 Old Habits Die Hard

Some of these new paradigms are so radically different that it requires a whole new approach when creating presentations. A presenter who is new to the tool has to take the extra effort of adapting to the new paradigm which might involve an entire refactoring of their methods. Some presenters might be willing to do so in order to reap the benefits but others might not be so willing and will stick to the tools that they know.

3.5.4 Accessibility of Online Tools

Some of the tools we discussed are partially or entirely internet-based. It is nice if a tool runs in a regular browser because it does not need installation or configuration but on the other hand this introduces a lot of new questions. If the tool only allows the presentation to be given via their online tool, is this not a vendor lock-in? If I put my time into creating content with this tool, will my effort be lost if I decide to move to another tool? If the company "disappears", what happens to my content? On top of that, online tools must

take into account typical network issues such as speed, disrupted connections and synchronisation problems. If they do not address these issues, this can have a heavy impact on the usability of the tool.

3.5.5 Ownership and Copyright

Once a tool allows easy sharing of content or more advanced concepts such as transclusion, ownership issues will inevitably arise. If presenters find a relevant presentation and decide to embed a slide in their presentation, who owns it? Or when they make small modifications to this slide, who owns it then? Is this presenter even allowed to use this slide in their presentation? These are all problems that can arise in tools where sharing or transclusion is a core aspect. There is no fixed answer to these questions but this should definitely be taken into account in the design of a presentation tool. A tool can, for instance, let the original creator choose to allow transclusion or not, or perhaps only on request. The tool could automatically mention the source if a presenter includes content from another presentation. These examples show that there is indeed no fixed solution to handle these kind of problems but a well-designed tool can certainly help to avoid serious ownership and copyright issues.

3.5.6 Tool Popularity

It is clear that the market for presentation tools is dominated by slideware tools such as PowerPoint or Keynote. This means that people who use alternative tools represent a minority. If these alternative presentations come in an obscure format or require a lot of setting up, it will be difficult for their users to share the presentations with users of other tools. This applies to every step in the life-cycle of a presentation: creation, authoring, delivery and execution. If a tool is not well known and does not provide an easy way of sharing and editing towards users of other tools, it can be very difficult to get new users who are willing to take the effort and give it a try.

4 The Ideal Presentation Tool

In the previous chapters, we have seen that there is still room for improvement in the domain of presentations tools. However, some of the issues are rooted so deeply that it is difficult or unpractical to solve issues by extending existing tools. In the early stages of this thesis, we have decided that it is justified to start designing a new presentation tool from scratch. By building a presentation tool from the ground up, we can rethink and redefine some paradigms that have been tagging along ever since the dawn of digital slideware tools. In Chapter 2, we have identified a list of issues and in Chapter 3 we saw that state of the art tools introduce some nice new concepts. It is our goal to build our tool so that it fixes as much of the issues as possible. By combining the good ideas from existing tools with our own input and insight, we hope to define a solid base for the ideal presentation tool. In this chapter, we discuss how our ideal tool would look like and how it fixes the issues we mention so often. We present our ideal tool as a theoretical design and a set of requirements. Actual implementation-specific choices are discussed in Chapter 5, which builds up to the actual implementation of MindXpres in Chapter 6.

4.1 Hypermedia

Earlier we have seen that hypertext and its related concepts may be a beneficial way of storing and managing information. We believe that a hypertext approach comes with enough significant benefits to step away from old representations and to take a similar approach towards storing and managing information within a presentation tool. Of course modern presentations are

not limited to simple text, so therefore we should look in the direction of *hypermedia*. Hypermedia goes beyond simple text and applies hypertext concepts to other content types such as images, audio and video.

We stop looking at presentations as bounded documents for holding sequential content, but instead expand the definition to *a visualisation over a repository of linked content*. A hypermedia system provides efficient ways of storing, managing and accessing information. Such a system would allow all hypertext concepts to be applied in the context of a presentation tool: transclusion, navigational links, structural links, spatial hypertext and more. Additionally, we must note that a user is not forced to visualise everything in such a repository. This means that instead of having content spread out over multiple documents, one could maintain a single repository of information. A presentation would merely be a specific navigational path through the repository, showing the selected content along the way. Such a repository does not need to be personal either. By collecting and storing content in a shared repository, people can share and reuse content within a group. For instance, a company could maintain a shared repository for a selection of their employees so that creating a presentation becomes a matter of assembling relevant pieces of existing information and contributing new content where needed.

However, the use of hypermedia does not come without pitfalls. It is very important to keep in mind that a hypermedia system needs to satisfy some specific criteria in order to catch on. First of all, there needs to be some notion of *structure*. As a hypermedia repository grows, things can quickly derail into a chaotic graph of tangled links. Certain measures should be taken to keep the content structured and accessible. Next, there is the issue of user management. We mentioned how repositories can be shared within a group, but this introduces some aspects that need special care, namely *ownership*, *access rights* and *security*. Without support for these concepts, the hypermedia system is bound for disaster in an environment with multiple users. Then there is the concept of *context*. In one way or another, the user might want to change certain aspects of the system depending on the context. For example, when a presentation is given for high ranking managers of a company, one might want the content to be displayed differently than in the case of a similar presentation for the store clerks of the same company. In the case of the managers, the company profits are best displayed in a detailed table, per quarter and with their origins, while the yearly results can be presented as a single number for the store clerks. In this example the context is the audience, but of course there are many other possibilities such as location, time, the user or ambient characteristics. Last but not least, hypermedia is a relatively complex concept for the average person. One cannot expect users to have a degree in Computer Science in order to be able to use

the tool. Therefore, special care has to be taken to ensure a certain level of *usability*. Ideally, the entire underlying hypermedia basis is hidden by the actual interface that interacts with the user. Most users don't care about terms like *semantic links* or *transclusion*. These features should be offered in a way that makes sense in the context they are used in, and in a way that is close to a mental model that the average user understands.

Requirements

- The tool should take usability into account regarding the underlying hypermedia system. Ideally, the hypermedia base is shielded by an interface that makes sense for regular users.
- Special mechanisms should be in place for user management. More specifically, ownership, access rights and security are very important in a shared environment.
- It is import to provide easy means of keeping the content structured and managable. If this is not the case, the repository might become a mess of tangled information, limiting the usability of the product.
- Context is a factor that should not be ignored. By providing a mechanism for context-awareness a lot of new features are possible.

4.1.1 Transclusion

In Section 3.1.3, we have already discussed the need for transclusion in documents. Presentations are no different and it would be nice if users could include existing content without actually duplicating it into the presentation (by reference). Therefore, our ideal tool should have support for transclusion for at least some of the base content types such as text and images. We suggest the following types of transclusion:

- Inclusion of content from the local filesystem
- Inclusion of content elsewhere in the same presentation
- Inclusion of content from other MindXpres presentations
- Inclusion of content from other presentation tools (e.g. PowerPoint or OpenOffice Impress)
- Inclusion of content from the internet (via HTTP)

Depending on the content type, user-friendly content selectors should be provided. For instance, regular expressions are a good way of selecting specific text but one cannot expect every user to know regular expressions. In this case, it would make sense to specify an additional selector for text, such as referring to anchors, sections or line numbers.

Additionally, there should be an option to prefetch transcluded content as it is not always feasible to do this during the presentation itself. One cannot always be certain that there will be a network connection during the presentation or that the content will still be available at its original location.

Requirements

- The tool should support true transclusion for at least the basic content types (e.g. text and images).
- Selectors can be complex and precise, but there should be user-friendly alternatives for the casual user.
- It should be possible to prefetch transcluded content to make sure everything will be ready and available at presentation time.

4.2 Zoomable User Interface

In Chapter 3, we have seen that ZUIs can be applied effectively to presentation tools. As shown by iMapping [19], hypertext (and also hypermedia) is efficiently visualised by a ZUI. We have shown that a ZUI solves a lot of problems associated with classical slideware tools. Tools like Prezi and CounterPoint highlight that a ZUI can be an improvement to regular slides if certain conditions are met (see Section 3.5.1). A ZUI is beneficial for multiple reasons:

- A ZUI has an infinite amount of virtual canvas at its disposal. This allows users to visualise things that would normally not fit on a slide to be shown on a limited screen size. Spatial boundaries are non-existant.
- A ZUI allows for easy navigation between presentation components that are not successive in the navigation path. If there is a reason to move to an earlier slide (e.g. based on a question from the audience) it is simply a matter of zooming out a little and clicking the target element.

- A ZUI can make use of spatial reasoning and spatial hypertext. Information can be grouped and structured according to the Gestalt principles (see Section 3.1.2) to make optimal use of our ability for spatial reasoning. Not only will it help users to orientate themselves better, a good layout possitively reinforces the learning process that the audience undergoes.

- A ZUI gives a better overview of the presentation. If the audience is presented with a global overview of the presentation, they get an idea of how long it will take and how the presentation is structured. During the presentation, the audience will have a better orientation as it is spatially apparent where the current view is situated in the big picture. Because humans have a limited working memory, it is important that the audience is frequently reminded how the presented information fits into the grand scheme. In a spatial hypertext approach it becomes quite easy to show how elements relate to each other.

- While multiple paths are possible in a linear sequence of slides, it is much more natural in a ZUI. One could theoretically put all their content in a single ZUI and simply define different navigation paths on the set of content. As free navigation is always possible we still have the additional information at hand if there is a need for it.

- Much like distance in spatial hypertext, transition animation can also help to define the relationship between information. As important elements are usually visualised bigger than details, a zooming animation could for instance indicate that we are literary going into detail.

A ZUI seems to be a good approach for a presentation tool but as we have shown, it is also possible to abuse it. For this reason, we define some strict requirements for our tool, ensuring that it is applied correctly.

Requirements

- Elements can be placed freely on the canvas without slide-like containers. This does however not mean that it is forbidden to group content in slide-like structures. One can still contain content in slides and arange them spatially as done in CounterPoint.

- Transition animations should be meaningful and serene. This type of animation should not be neasea inducing but should instead contribute to the presentation, for instance by carrying information on the relation between elements.

- Not everyone is a fan of ZUIs and this could scare away potential users. It should be possible to disable zooming and/or transition animation to simulate sequential presentations as offered by slideware tools. After all, such a sequential presentation can be seen as a ZUI with a fixed zoom level and without transition animations.
- The ZUI is able to focus on a single element (e.g. picture or text) but it should also be possible to define views that envelop multiple elements. One can see this as a kind of invisible container that can can receive focus like any other element.

4.2.1 Multiple Navigation Paths

The classic slideware tools do not offer a lot of flexibility regarding the navigation path defining the order in which slides are traversed during the presentation. Usually, users are restricted to the order that is implied by the sequence of the slides. In the case of a ZUI, we have much more freedom simply because there is no order. Items are laid out on a canvas in multiple dimensions so in this case a navigational path is the order in which the user decides to visit the information.

Requirements

- Users can define as many navigational paths as they want. These paths can be identified with a name.
- The presenter can leave the predefined path at any time in order to navigate manually.
- After leaving a path, it should be possible to resume it at a later point in time.
- If no path is defined the user gets the choice between a path that follows the order in which elements apear in the presentation document or they can use manual navigation.

4.2.2 Navigational Links

We all know the navigational links from the World Wide Web where we can click on a specific piece of text or image to be taken to another location. A similar concept is applicable to presentations. In order to support quick

navigation to specific content, the tool should support hyperlinks within content, which trigger the navigation to other content when activated. Such a navigational link should be defined by the user, who defines what part of the source content is a hyperlink (e.g. specific words or a picture) and the target content.

Requirements

- The tool should support navigational links (alike to hyperlinks in the World Wide Web) to enable quick navigation to other content.

4.3 Focus on Content

We have seen that a lot of work goes into creating good-looking presentations. More than often, this takes more time than writing the actual content itself. By choosing a ZUI as visualisation approach, we did not exactly aid this cause as Prezi has shown us that laying out everything manually is tedious work. To make the creation process more pleasant and a lot quicker, we propose the following approaches.

4.3.1 Semantic Content Creation

If there is one presentation tool that excels in content-creation it would be Beamer (see Section 3.3.2). By taking a LaTeX-like approach to creating presentations, the tool aids presenters to quickly and efficiently create presentations without having to worry about the actual visualisation. We propose to take this cue with the goal of facilitating the creation process as well as shortening the time required.

```
1  <presentation theme="default">
2    <slide type="titleslide">
3      <title>This is my title</title>
4      <name>Reinout Roels</name>
5    </slide>
6    <slide type="content">
7      <title>This is a content slide</title>
8      <image source="myimage.png" />
9      <bulletlist>
10       <bullet>Bullet 1</bullet>
11       <bullet>Bullet 2</bullet>
12     </bulletlist>
13   </slide>
14 </presentation>
```

Listing 4.1: A presentation in XML

In order to easily describe an entire presentation the tool should provide a semantic interface or a domain-specific language. This proposal does not enforce a specific language, but acts as an example. Listing 4.1 illustrates how easy it is to describe a presentation in XML.

We realise that such an approach is not well suited for casual computer users. One cannot expect PowerPoint users to use our tool if they see that everything has to be "coded" in a scary looking format. However, there is an easy solution to this issue. It is perfectly possible to create a user-friendly GUI that generates the semantic representation automatically (e.g. the XML code in our example). This is similar to the way PowerPoint documents are actually bundled XML files that are generated by the designer. This flexible approach offers multiple methods or creating content, offering a customised level of usability to different types of users. Note that the whole point is to provide a focus on content and therefore our semantic content representation does not contain aspects related to their visualisation.

Requirements

- A semantic interface should be provided to allow users to focus on the content. LaTeX can be seen as an example of such a content-driver approach.
- Optionally, a GUI can be provided as an added layer of usability.
- The language should be well documented and described.
- This language is void of all things visualisation related. The focus is purely on content.

4.3.2 Themes

In an environment that is dominated by tools that focus on visualisation, it is crucial that the proposed tool is able to produce good-looking slides in order to compete. Because of the established standards regarding visual quality, the looks of the resulting presentation can be a deciding factor for some users. The difficulty lies in creating good-looking presentation without taking the focus away from content. This is where themes come into place. A theme is a separate file or bundle that describes the looks of a presentation. It contains descriptions for things like background colours, the fonts used, element sizes and anything else related to visualisation. This has multiple advantages. For one, switching between themes is as simple as changing a single word in the presentation format. Because visualisation information is

separated from the content it is very easy to apply major graphical changes to the entire presentation. Users can also create their own themes which they can reuse for every presentation they create.

Themes not only contain aesthetic details but can also specify how elements should be laid out within their container. In the XML example from the previous section we define a slide with a title and a name to be displayed, but how does the tool know where to display these? In the first line of the example the `default` theme is applied to the entire presentation. In the second line a slide container is defined with the attribute `type="titleslide"`. In the `default` theme the layout would be defined for containers with the type "titleslide". For instance, the definition could state that a "titleslide" has a title that is displayed in the middle of the container and it has the name of the author displayed in the lower right corner. When the presentation is visualised the tool sees that the container has the type "titleslide", the title and name values defined in the authoring language are bound to the regions described in the theme.

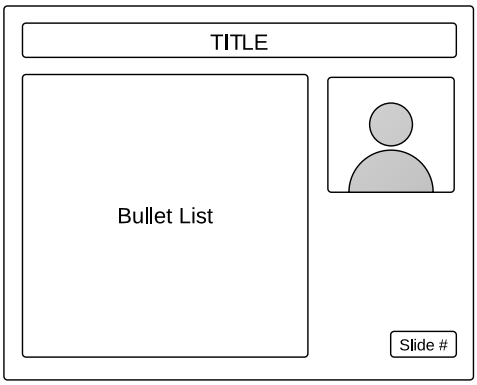


Figure 4-1: Theme defined content regions within a container

Requirements

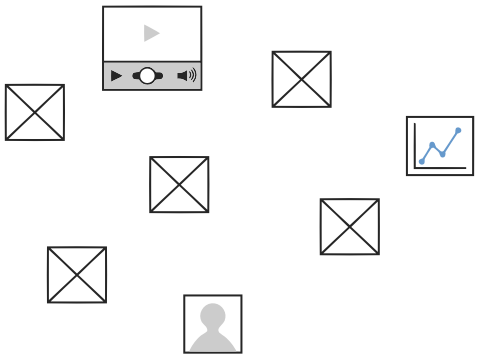
- Visualisation is clearly seperated from content.
- Themes are reusable.
- Users are able to create their own themes.
- A custom theme is able to inherit properties from another theme in order to build further upon that theme.
- Themes can also describe layouts for containers. The theme described what elements the container can have and how they are

laid out. When a container is visualised, the content from the presentation format is bound to the regions defined by the theme.

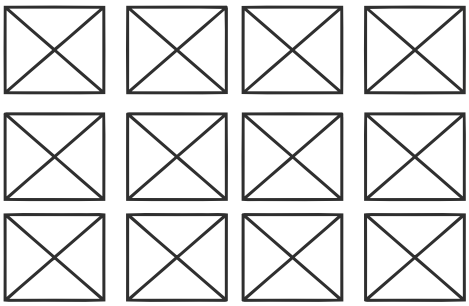
4.3.3 ZUI Layout

Users are allowed to lay out their content freely, but by providing this flexibility, we are shifting the focus from the content to its visualisation. User should not spend most of their time on laying out content on the canvas but should focus on the content and have the tool handle the layout. The tool cannot autonomically generate an ideal layout out of a set of content, but it can support the user by providing some layout templates from which they can choose an initial layout. Allow us to illustrate with some example layouts.

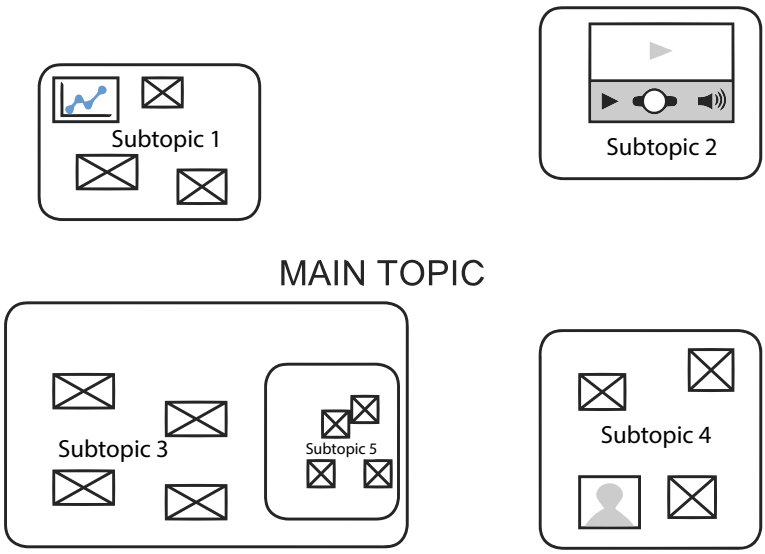
Graph The most basic layout is the graph, where content is laid out in a graph-like structure according to how the information is linked, both semantically and navigationally. The size and shape of the content is taken into account so that enough space is present between the different *nodes* in the graph.



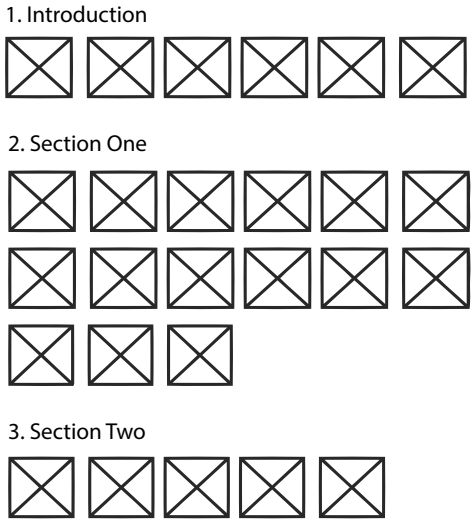
Grid When content has roughly the shame size or shape or are bound by similar containers (e.g. slides), a grid layout may help to provide a clean layout. The content is laid out in rows of equal size. Variants may include grids with one row or one column, in order to create sequential layouts similar to the ones provided by existing tools.



Clustered One might provide a layout that is close to spatial hypertext like we discussed earlier. Elements are grouped according to their relevance, and Gestalt principles are used to create spatial relations between them.



Structured Much like in a book, content is grouped in sections and topics.



Requirements

- The ZUI aspect should not become a burden for the person that creates the presentations. The tool should provide default layouts for content so that the layout of content is handled by the tool, and not by the user.

4.3.4 Presentation Structure

We proposed to take a LaTeX-like approach to presentation authoring, so there should also be support in the language to structure our content. In LaTeX, it is possible to structure a document in chapters, sections and subsections. Next to the obvious visualisation of content within these structures, LaTeX uses the structure for various other features. For instance, the automatic generation of the table of contents or internal referencing are features that are only possible because of this structure.

In a similar way, it is possible to introduce a high-level structure for a presentation. One could group content in *sections* based on the shared topic, or one could divide a presentation in different stages such as introduction, background, problem statement, solution and conclusion. On its own, such structuring might not have an immediately apparent use, but it allows for additional features that will aid the user in the long run. These additonal features include automatically laying out content in a way that makes more sense (e.g. the clustered or structured layout).

Requirements

- The container format should provide methods for structuring the content, similar to the way LaTeX allows users to structure text in chapters and sections.

4.3.5 Import of Existing Content

It can be a major issue to convince a user of another tool to use our ideal tool if they already have a lot of existing content in the format of the old tool. To facilitate the transfer between tools, our ideal tool should provide functionality to import content, automatically converting it to our format. Depending on the content, this could be done fully autonomous or optionally with some user guidance for more complex content.

Requirements

- Provide import functionality for existing presentation formats (e.g. PowerPoint).

4.4 Content Presentation

The common presentation tools support basic content types such as text, images, video and graphs but often the need arises to present domain-specific content that is not supported by the tools. An example of this is to present programming code or source code. In an Integrated Development Environment (IDE) source code is automatically coloured to increase the legibility, but there is no support for this basic functionality in most existing presentation tools. When the need arises to present source code, the user has to display it as regular text and has to colour the code manually or via external applications. Additionally, the spatially bounded slides are particularly bad for displaying source code since a broad overview of the code is required for it to make sense. Instead, teachers are required to spread the code over multiple slides and are forced to skip back and forth between them when the presentation is given. This is just one example of a content type that is difficult to embed in a presentation, but there are many more that can make a presenter's life difficult. We propose the following solution to tackle this problem once and for all.

4.4.1 A Component Plug-in Architecture

Most presentation tools offer a fixed selection of components (e.g. text, images and bullet lists) but rarely do they offer the possibility to introduce new ones. We propose to approach this differently and make it easier to extend the available components. Our ideal tool offers an extendible core mechanism where each and every component is implemented as a plug-in. Not a single component is "built in", so even the most basic components are plug-ins. Note that components can also be containers for other components (e.g. a slide) and thus some are nestable. The tool should provide a default selection of components such as as text, images, bullet lists and video but also support the addition of new components, for instance by simply placing the plug-in in a folder. Ideally the plug-in interface is open and documented, meaning that anyone is allowed to create new components. If the tool ever becomes popular we hope that communities will form where component plug-ins are created and exchanged. An additional benefit of having this plug-in architecture is that it would be possible to create component-specific themes that focus on the visualisation of a single plug-in.

Requirements

- The tool should offer a mechanism for plug-in components.
- The interface for these plug-ins should be open and document in order to encourage third party development.

4.4.2 Examples

To demonstrate the possibilities of such an architecture, we present some examples of much-needed components. Note that these are theoretical examples and that they have not all been realised in our implementation.

Source Code

We have already introduced the problems associated with the presentation of source code so we will jump to a possible solution. To save the user a lot of time, the syntax colouring or highlighting should be done automatically. There are a lot of different existing solutions for syntax colouring so it should not be too hard to incorporate once of them in the form of a source code plug-in. Additionally, a source code component could introduce some new concepts for presentations:

- Source code could be presented in a scrollable container, which would facilitate the visualisation of longer source code snippets.
- It could be possible to define a navigational path through the source code, highlighting pieces along the way.
- By parsing the source code, the component could be made to be syntactically aware of its content. For instance, if the presenter were to select a method invocation, the component would scroll to the method definition.

Image Gallery

Modern presentation tools all have support for images, but due to spatial restrictions it is often difficult to display series of images. They have to be spread over multiple slides, making it more difficult to visually group specific images together. This also drastically increases the number of slides. It would make more sense to create a component specifically designed for series of images. The component would provide means of efficient navigation in a group of images, for instance via thumbnails. As a plug-in is free to visualise its content as it wishes, it could go as far as visualisations such as Apple's Cover Flow.

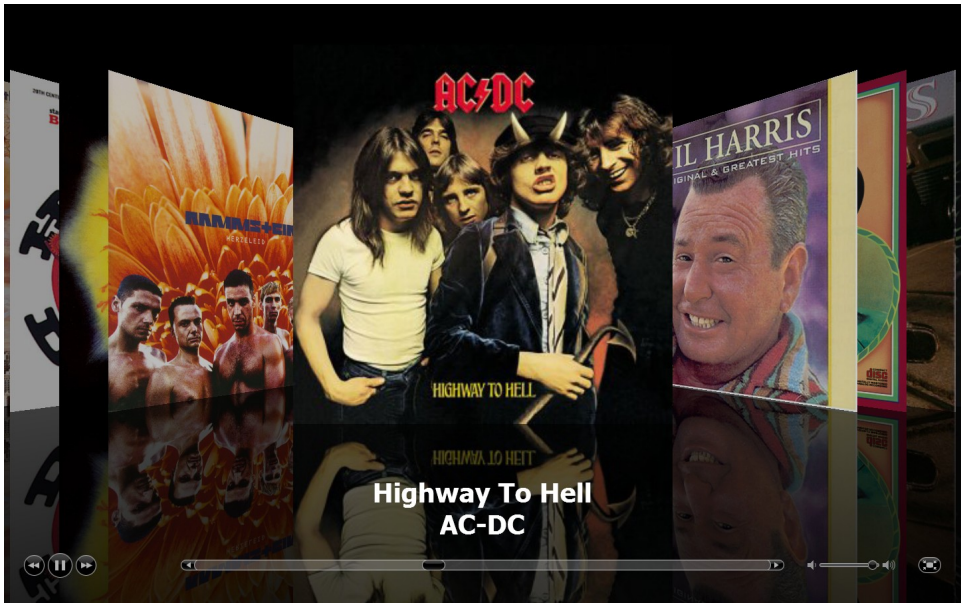


Figure 4-2: Apple's Cover Flow

Improved Video

Including video in a presentation is not new. However, during the presentation these videos are handled in a player that resembles a movie player one might use at home for entertainment. The context of a presentation has not been taken into account and therefore we claim that major improvements are possible regarding video in presentations.

- Bookmarks: Often a video only contains some parts relevant to the presentation. Instead of having to navigate manually, it would be nice if the presenter could predefine bookmarks for quick navigation within the video.
- Timed Events: During a video some interesting concepts could be shown and the presenter might want to discuss these before continuing. In the current video players, the presenter would have to pause the video manually at specific moments in time. A better video component could allow the presenter to predefine specific actions at specific times. For instance *"stop the video at 1:25"*, or *"pause the video for 15 seconds at 0:35"*.
- Annotation: In the current tools it is impossible to annotate a video. However, it would be nice if it were possible to display text at specific moments or highlight certain objects as the video plays. A custom component could allow the presenter to specify annotation events. For instance, *"display a circle at coordinates (100,150) between 0:15 and 0:25"*. This could be combined with the timed events, for instance *"pause at 0:15 for 15 seconds, and draw a circle at coordinates (100,150) while it is paused"*.

Quotes

Much like in text documents, there are reoccurring types of text with identical visualisations. For instance quotes or definitions that are often visualised consistently throughout the document. In such a case, it would make sense to create a component specifically for this type of content. One example is a quote, which might be visualised with an indent, in an italic font, in a special container. By creating the component once (or installing an existing one), the user never has to worry about the visualisation of quotes again, and can instead focus on the content.

Mathematical Formulas

Another niche content type is mathematical formulas. In some existing tools, it is impossible or downright frustrating to enter complex mathematical formulas. Instead of forcing the user to use external tools, it would be better

to create a component to facilitate their rendering. As teachers of mathematics courses often have experience with markup languages such as LaTeX or MathML [6], it would be better if they were able to enter their formulas in their language of choice and that the component handles the rendering.

$$R = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2\right]^{1/2}}$$

Figure 4-3: An example equation

Clocks and Timers

Another component that seems to be missing from most presentations tools are simple clocks and timers. In some situations it would make sense to display the current time or to have a countdown timer (e.g. for strict breaks). With a plug-in architecture it becomes trivial to implement this.

4.5 Usability

4.5.1 Search

Next to free navigation there should be other means of locating specific content during the presentation. Next to the common text-based search, we suggest the ability to tag content with specific taxonomies and keywords. By doing so, a search could be done over both content and tags. This also allows search results to be presented in a structured way, for instance one similar to the structured layout where results are grouped by their taxonomy.

Requirements

- Allow searching through the content during the presentation.
- Allow users to tag content.
- Use tags for enhanced searches during the presentation.

4.5.2 Generic Input Interface

As we have shown in Section 2.5, most tools are centred around keyboard and mouse input. However, research on human-computer interaction is booming

and radically new input mechanisms are still presented in frequent intervals. We are at a point where some devices do not even have a keyboard or a mouse anymore (e.g. tablets and smartphones), yet presentation tools seem to continue the focus on the old input methods. There are many new ways of input such as multi-touch, gestures, speech and digital paper.

Instead of building the tool around the keyboard or mouse, we propose to build the tool with no specific input methods in mind. Instead, the tool should provide an open generic input interface which allows other software or hardware to be used as an input method, on an equal level. Of course, keyboard and mouse input are important input methods and should be supported by default. Instead of centring the tool around them, support for them is created as an extension on the generic interface. Similarly, other input methods are implemented as an extension on the generic interface. Because of this all input methods are treated equally which allows new input methods to be added easily in the future. To facilitate this, one could make use of existing frameworks for multi-modal input such as Mudra [23].

Requirements

- Do not limit the tool to a specific input method, but instead design a generic interface that allows any number or type of input method to be "plugged in".

4.6 Portability

The output of most authoring tools is a presentation in a proprietary format. This means that the presentation can only be given via the tool and thus the number of platforms that can be used for the presentation is restricted to the number of platforms that the tool supports. Sadly, most tools only support a small number of platforms or are limited to particular technologies that are not available everywhere (e.g. Flash). Some tools allow users to export their presentation to static formats such as PDF but then most, if not all, multimedia functionality is lost (e.g. animations, audio, video or transitions). An optimal solution would be to use an open well-known format that is able to represent the functionality of modern presentations. Ideally, this format should be viewable with tools that are available for most of the common platforms.

Requirements

- The tools for viewing the presentation should be available for as many platforms as possible, including mobile devices.
- The container format should be open and documented in order to allow third parties to create viewers for less-used platforms.
- Allow the presentation to be generated in different formats. For example, in some cases the user might still want to have a PDF representation, for instance for printing on paper.

5 The MindXpres Presentation Tool

In Chapter 4, we have detailed what we see as the ideal tool. This was presented in the form of general suggestions and requirements. The goal of this chapter is to turn these requirements in a concrete design. This means that we discuss some of the options for fulfilling the requirements and decide on the implementation-specific details such as tools and technologies. In Chapter 6 this design is then used to create a working prototype of the envisioned tool. In the following sections we present specific technologies and tools that will help us to achieve the requirements imposed in Chapter 4.

5.1 The RSL Model

Hypermedia implies a lot of features. These features include semantic linking, navigational links, structural links, transclusion, annotation and context awareness. In order to realise these features, a solid foundation is needed to manage the hypermedia aspects internally. There needs to be a solid definition of how elements can be linked, how they interact and how higher levels can access the information. Such a definition can be realised through what is called a hypermedia model. Such a model is often defined without regards to implementation related aspects.

There is no shortage in hypermedia models. As early as 1994 such models have been defined, for instance the Dexter Dexter [18] hypermedia model. Over time this model has been extended to support time and context [20], add user models [4] or fix implementation issues [17]. This is only the tip of the iceberg as many domain-specific models have been defined, some extend-

ing others and some as standalone models. We have chosen the Resource-Selector-Link (RSL) model [47] as our basis, which is based on the OM data model [38]. The RSL model is fairly recent (2007) and shortcomings and issues that were make in earlier hypermedia models have been taken into account during the development. The model provides a *general platform for the development of hypermedia systems*, and is a perfect fit for the solutions that we have in mind. The resources, selectors and links can be seen as three base components for a hypermedia representation:

- Resource: A resource is a piece of data that we have in our presentation. This can be text, images or any other content type
- Selector: When we link to a specific resources we do not always wish to refer to the entire resource. Sometimes it is necessary to refer to a specific part of the resource. For instance, a single paragraph of a long text or a small region within an image. A selector is an abstract concept that specifies a selection within a resource. Examples of implementations are for instance a regular expression for a text resource, an XPointer for an XML resource or a time interval for a video resource.
- Link: Linking resources is the most fundamental concept of hypertext. A good model allows links to have multiple sources or targets and provides the possibility for different types of links. Resources can be linked directly, or via a selector.

Allow us to briefly summarise some of the main features that make this model a prime candidate:

- Links are first class objects
- Navigational and structural links
- User Management
- Context Awareness
- Properties
- Selectors
- Extendability

In order to elaborate on these features we have to take a closer look at how the model defines the different components. In the following subsections we go into detail and these features will be explained along the way.

5.1.1 The Link Metamodel

The link metamodel is shown in Figure 5-1. The rectangles represent collections of objects of the type that is shown in the shaded part of the same rectangle. The ovals represent associations between members of collections. Note that the model is a metamodel (a model for models), and thus all the abstract concepts need to be extended in order to get a model for a specific domain (e.g. presentations).

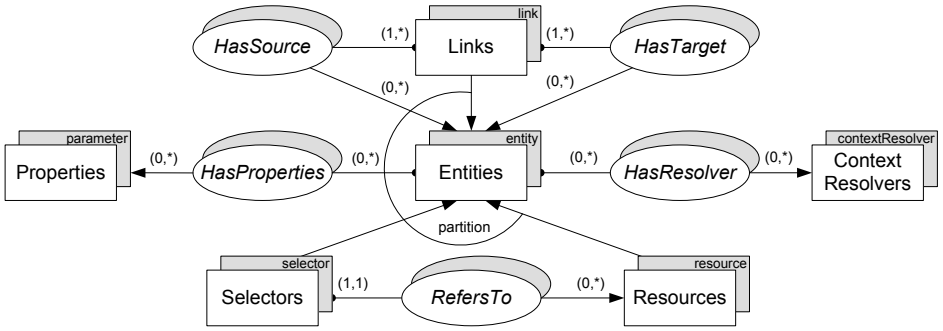


Figure 5-1: Link metamodel (Source: Signer and Norrie [47])

In the centre of the model one can find the `Entity` type, which is an abstract type that is the base for the three main components in the hypermedia system: `Resource`, `Selector` and `Link`.

Resource The `Resource` type represents a specific type of media. One must not forget that this is an abstract type and thus for each concrete content type that the hypermedia system needs to support (e.g. text, images or video) one has to create an extension to this base type. This allows implementations to create plug-in systems for *easy management of supported content types.*

Selector A selector is an abstract concept that allows one to refer to a specific part of a resource. Again, this abstract type needs to be extended for every `Resource` that the hypermedia system needs to support. For instance, a type representing an XPointer expression would be a good selector for the text resource, while a temporal selector would be suited for the video resource type. The cardinality constraints imply that a selector can only refer to one resource, but a resource can be referenced by multiple selectors.

Link A `Link` has its cardinality constraints set up in a way that allows it to have multiple sources and multiple targets. Also note that the sources and targets of a `Link` are of the type `Entity`. This implies that a link can establish relationships between resources, selectors and other links. Referring to other links can be useful for annotating other links with pieces of

information. If a link points to a selector it means that it indirectly points to a part of a resource, where the selector provides the means of extracting the needed part from the resource. Because the cardinality constraints specify that a `Link` has at least one source and target, the model ensures that no dangling links are possible.

On top of these core concepts the metamodel provides two useful concepts that allow additional features.

Properties Properties are a set of parameters that can be associated with a specific `Entity`. These parameters are stored as string key-value pairs and provide a way of allowing additional flexibility for future extensions. For instance, one could allow the customisation of an entity's behaviour via these parameters in order to make it easily adaptable to other domains.

Context Resolver A context resolver is an element that provides context-dependant handling of entities. Such a context resolver may grant or deny access to specific entities based on a context. Such a context could for instance be a time (a specific resource is not allowed to be accessed between specific times) or a resource type (the system may only follow links to a specific resource type, but not others).

5.1.2 The User Model

Another feature that is often lacking in other models is user management. This includes both ownership and access of information. The RSL model provides both of them via the user model as seen in Figure 5-2.

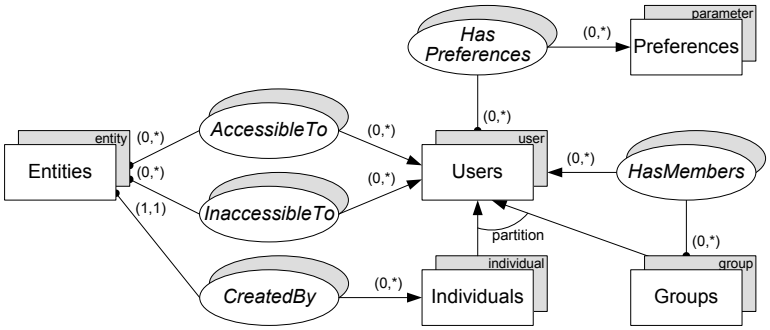


Figure 5-2: User model (Source: Signer and Norrie [47])

The RSL model incorporates user management at the lowest level possible. This is shown by the fact that all the user-related features are handled at the level of the `Entity` that we described earlier. A `User` is either an

`Individual` or a `Group`. The concept of a `Group` is similar to what is offered by many operating systems. Different users have different rights, but instead of having to keep track of these rights for each user individually, groups are created as a classifier for assigning rights. For instance, in our operating system example there might be an administrator group that can do anything, a user group that is not allowed to install new software and a visitor group that is not allowed to install software or modify any settings. When a new user is created, they are simply assigned to one of the existing groups and he automatically inherits the access settings for that group.

When an entity is created, it is owned by one user as defined by the cardinality of the `CreatedBy` relationship. The owner can then assign rights to groups or individuals, allowing sharing and collaboration. There are two relationships for assigning access rights to data: `AccessibleTo` and `InaccessibleTo`. This allows a flexible assignment of access rights as it is possible to set up complex rules. An an example of such a rule could be "*Allow* $group_1$*,* $individual_1$ *and* $individual_2$*, but make it inaccessible to* $individual_3$ *(which is member of* $group_1$*)*".

Together with the `Context Resolvers` we described earlier it is entirely possible to set of a system of access control, securing information where needed. This user model defines the basic foundation for all features related to sharing, authoring and ownership, and is therefore well suited for use in a presentation tool.

5.1.3 Layers

Earlier we mentioned how selectors can be used to access specific parts of a resource. A closer look at the link metamodel shows us that it is allowed to have multiple selectors for the same resource. This means that it is possible that two (or more) selectors resolve to overlapping content. This can be a major issue, as it is not defined how the system should handle these conflicting resolvers. Please allow us to demonstrate this issue with a simple HTML example:

```
1 <a href="target1">
2   This is some text that links to target1, but
3   <a href="target2">
4     this text links to target2!
5   </a>
6 </a>
```

One can now ask the question, if one clicks on "*this text links to target2*", will it resolve to target1 or to target2? It is a perfect example of a situation where overlapping links cause conflicts and unless a resolve mechanism is defined, there may be unexpected behaviour. In HTML, nested links are

illegal and therefore most browsers will handle this example in unexpected ways. We presented this example because even though it is illegal in HTML, it does not need to be in all hypermedia systems. The RSL model provides a solution to this problem via `Layers`, as illustrated in Figure 5-3.

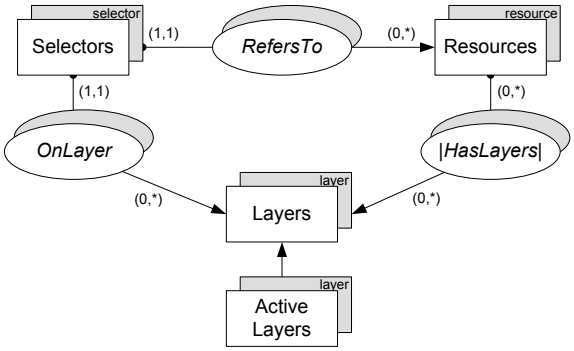


Figure 5-3: Layers (Source: Signer and Norrie [47])

In this model, a selector is associated with a `Layer` and a restriction is in place so that overlapping selectors are not allowed to be associated with the same layer. The vertical bars in the `|HasLayers|` relationship indicate that this is a ordered association, meaning that the layers have a particular order. If a problem with overlapping selectors arises, the selector associated with the top layer is selected and used, resolving all issues.

One could say that this approach is restrictive, because what if one does not want the selector associated with the top layer to be selected, but instead wishes to use another one in some contexts. That is where the `Active Layers` collection comes into play. Layers can be marked as active or inactive via their membership in the `Active Layers` collection. Inactive layers (not associated with the `Active Layers` collection) are simply ignored allowing selectors in lower layers to be selected. On top of that, the layers can be reordered dynamically enabling applications to change the order (and thus selector) depending on the context.

5.1.4 Structural Links

Earlier we have described the link metamodel, but there all links have the same type, namely `Link`. The RSL model extends this type further by proving two subtypes of links, `Navigational Link` and `Structural Link`.

Structural Links are links that tie information together to form structures. For instance, a book has structure, it has chapters, sections and paragraphs. Structural links can for instance be used to represent such a book-like structure, by defining structural relationships over the different resources.

Navigation Links are links that drive the navigation in one way or another. An example of that are the HTML hyperlinks everyone knows. These are visualised in a way that the user can interact with, in order to guide the navigation.

The definition of these two link subtypes can be seen in Figure 5-4.

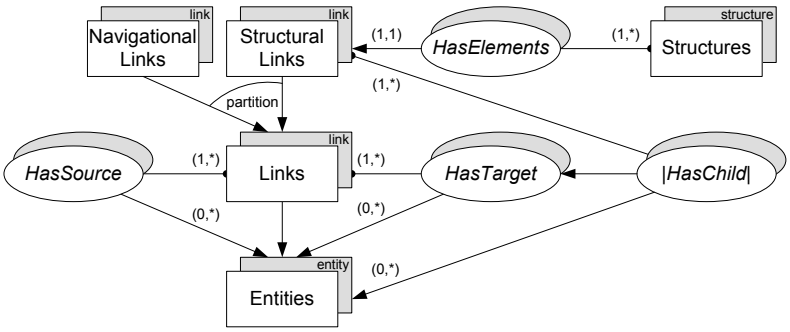


Figure 5-4: Layers (Source: Signer and Norrie [47])

A `Structure` is composed of one or more `Structural Links` and defines a structure over information. A structural link is simply a subtype of `Link` that refers to one or more `Entities` via the `|HasChild|` relationship. The `|HasChild|` relationship is again an ordered relationship. This is needed because a structure implies that there is a certain order and grouping. In the case of a book-like structure, as the one mentioned earlier, we have chapters that need to be ordered in a fixed way that makes sense for the content. It is not only possible to define structures over data, but it is also allowed to define structures over structures. This allows one to build more complex structures that are composed of smaller substructures. For instance, a book has chapters, a chapter has sections and a section has paragraphs. This is a case of nested structures that could not have been achieved with just structures over data, as this would just result is a flat sequence of ordered elements. Note that this concept is very beneficial for transclusion, as it would allow references to be made to structures or substructures instead of just resources. For instance, one could transclude a single paragraph or section from a book into another document. Finally, it is noteworthy to mention that it is also possible to define structures over links. For instance, navigational links can be structured to form navigational paths that will guide a viewer through the information.

5.1.5 Implementation of the RSL Model

The RSL model has been implemented in the form of a hypermedia server called iServer [45]. iServer provides a Java API which allows easy access for applications. In order to support the needed media types a lot of resource plug-ins have been defined with their corresponding selectors. Table 5-1 shows some of the media types that have been implemented, together with the selector that is used for that resource.

Medium	Resource	Selector
paper	document page	shape
web page	XHTML document	XPointer
movie	mpeg file, avi file etc.	time span
movie	mpeg file, avi file etc.	shape
sound	mp3 file, wav file etc.	time span
image	gif gile, jpeg file etc.	shape
database	database workspace	query
physical object	RFID space	RFID tag

Table 5-1: iServer plug-ins

5.1.6 Conclusions on the RSL Model

In the previous sections, we have detailed the RSL model and discussed the major components in detail. The combined model can be seen in Figure 5-5.

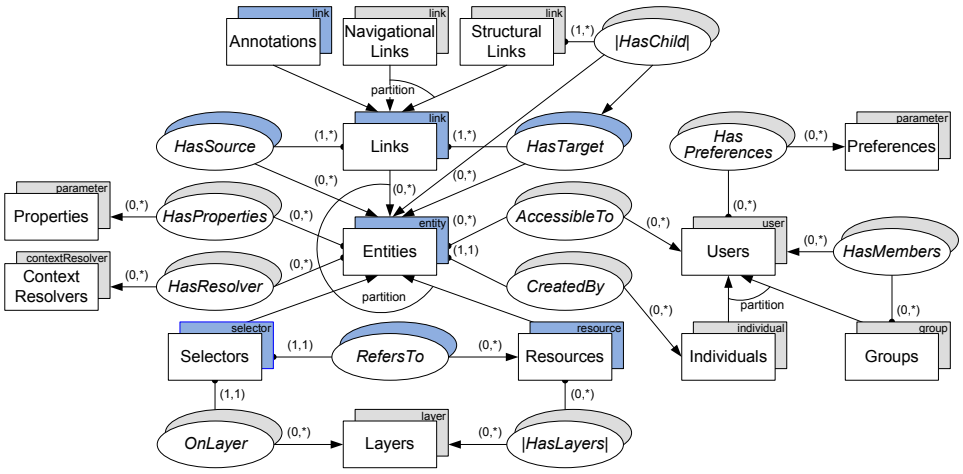


Figure 5-5: Complete RSL model (Source: Signer and Norrie [51])

Overall, we believe that a hypermedia model such as the RSL model is a solid solution for many of the problems associated with slideware. In this

section we have taken a closer look at the RSL model and have discussed a lot of concepts that will provide the necessary features that are needed to tackle some major issues present in modern slideware tools. To end this section on the RSL model, we quickly recapitulate the offered concepts and their implications on our presentation tool.

- The RSL model allows us to step away from hierarchical representations and instead organise information in a graph-like structure. Instead of storing information in structured documents, a hypermedia system can be seen as a respository for resources. A presentation would merely be a navigational path through the resources in one's personal repository. All hypermedia functions can be achieved with the RSL model: transclusion, annotation, navigational links, structural links and more. This opens a lot of doors regarding how information is stored, visualised and managed for presentations.
- Hypermedia does not need to be a chaotic graph of linked resources. The structure functionality allows us to define structure where needed.
- The RSL model provides user management at the core level. This makes features possible such as collaboration, sharing and security.
- The RSL model is easily extendable with new media types and selectors to suit the needs of modern presentations.
- The RSL model was designed with real-life applications in mind, and enough functionality is built in to suit all possible needs. For example, context-awereness is built in at the core level, and special care has been taken to make the model customisable and extendable, ensuring that the model is usefull in the long term.

5.2 Content Creation in XML

As suggested in Section 4.3.1, an ideal way to shift the focus to content is to provide a semantic interface for defining presentations. After giving it some thought we have chosen XML as the language to define presentations. The main reason for this is that XML is so widely known and supported. XML offers us the needed flexibility of defining a custom language through the definition of XML Schemas [13]. In short, an XML Schema is a formal definition of an XML-based language, describing the allowed tags, their allowed relations and their parameters. As an example, XHTML is an application of XML that is defined via an XML Schema[1]. The benefit of this is that it is very easy to verify the validity of a XML document that claims to be a valid

[1] http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd

presentation. First, a check on the XML syntax can be done. If the document adheres to valid XML syntax, the document is well-formed. However, in order to be *valid*, a second check needs to be done that verifies whether the XML structure is valid according to the rules defined in the schema.

XML is easy to work with and regardless of the programming language, XML is generally widely supported. This helps to ensure that it will be possible to build a GUI for presentations that can generate the presentation format. Due to time restrictions, the implementation of a GUI is not part of this thesis but it does show that usability should not be an issue. An XML-based format allows us to fully implement the system with the option of adding a usability later at a later point in time.

5.3 Content Visualisation

In Section 4.3.3 we have already established that a ZUI is an excellent choice for visualising hypermedia. In this section, we will discuss the candidate technologies for building a ZUI, which will form the foundation for presentation visualisation. When picking the right tools, it is also important to keep the other requirements in mind. For instance, the choice of technology can have a major impact on the performance or portability. When we were searching for the right tools for the job, we set up some basic requirements to help us to narrow down the ideal tool(s). We started off with the following requirements:

Interfaceability: As discussed earlier we have a complete separation of the compiler and the visualisation layer. However, ideally we still want to be able to interface with the visualisation layer via other applications and tools, or even between multiple instances of our presentation. An example of why we would want this, is for instance to couple standalone modules for multi-modal input to our presentation. As an example, we could interface a gesture recognition engine such as Midas [43] with the visualisation layer to help steer the navigation.

Cross-platform: In order to make the visualisation layer accessible by as many people as possible, we do not want to restrict it to a single platform. We want to support at least Windows and OSX, and optionally some popular Linux distributions and mobile devices such as smartphones and tablets.

High-level graphics framework: We want the tool to provide some high-level functionality regarding graphics and visualising out of the box. Arguably we could say that pure C with OpenGL meets all other criteria in this list, but one must keep in mind that the project only has a limited time

frame, and thus the focus should be put on implementing our presentation tool and not on creating an OpenGL engine. Therefore it is important to pick high-level frameworks and tools that make it easy for us to focus on what is relevant to the project.

Heavy animation: Since the start of the project we had always envisioned the option for fluent animation in our presentations (much like Prezi). For this reason it is important that the chosen tool is capable of fluid animation such as zooming, rotating and panning. Optionally the tool should have support for managing and creating these animations out of the box, for instance via key frames.

Additionally we favour tools and frameworks with the following optional requirements:

Easy integration of GUI components: Similar to the previously mentioned condition that there should be sufficiently high-level graphical capabilities, it would be nice if our tool of choice also had some support for basic GUI components such as text and images and perhaps even high-level components such as tables, charts, or even a video player for instance.

3D Support: Optionally the ideal tool should have 3D capabilities. While 3D visualisation is not part of the project as it is presented here, it would be nice to have support for this to keep the options open in future work. This request is however more of a bonus and definitely not a deciding factor.

With these requirements in mind we take a look at the candidates. For each candidate we list the negative aspects (marked with "✗") as well as the positive aspects (marked with "✓").

Game Engines

- ✗ Game Engines often require a (relatively) low-level programming language such as C or C++ (e.g. Irrlich[1], Ogre[2] or Panda3D[3])
- ✗ Frameworks for high level languages are often too slow for heavy animation (e.g. SDL[4], Allegro[5] or LÖVE[6])
- ✗ Most game engines do not provide high level animation functionality (zooming, panning or scaling) out of the box

[1] http://irrlicht.sourceforge.net/
[2] http://www.ogre3d.org/
[3] http://www.panda3d.org/
[4] http://www.libsdl.org/
[5] http://alleg.sourceforge.net/
[6] https://love2d.org/

- ✗ Often there is no built in support for GUI elements
- ✓ Very fast regarding 2D and 3D visualisation

Flash / Flex

- ✗ We have no experience with Flash and related technologies
- ✗ Requires a platform dependent runtime library to be installed
- ✗ Flash is not based on open standards
- ✓ Lots of built-in GUI elements
- ✓ Cross-platform (except for Apple's iOS)
- ✓ High level support for animation

JavaFX

- ✗ Requires a platform dependent runtime library to be installed
- ✓ Lots of GUI elements, even web views, video players and grids
- ✓ Cross-platform (except for Apple's iOS)
- ✓ Flash-like animation support
- ✓ It is plain Java, and therefore there is a small learning curve
- ✓ Hardware accelerated
- ✓ Recent new version (2.0)

HTML5

- ✗ While fluent graphics are well underway, pure JavaScript is not sufficient (yet) for things like live rotation and scaling
- ✗ We could use the canvas element with WebGL, but then we loose most of the other benefits HTML offers
- ✓ Lots of built-in GUI elements, easy to visualise almost anything
- ✓ Easy to extend with JavaScript libraries, or even Flash and Java Applets
- ✓ Cross-platform
- ✓ While pure javascript may not be adequate for easy and fast animation, CSS3 transforms can be used to fulfil this need

Ultimately, Flex, JavaFX and HTML5 are almost equal candidates, but in the end we have chosen HTML5 for the following reasons:

- The degree to which a tool can be considered to be *cross platform* is always a bit relative, but when we look at how well HTML does on this aspect one can not deny that when compared to the other candidates, it has almost maximum portability. Every modern desktop operating system has web browsers that support HTML, JavaScript and CSS. When we look at mobile devices we also see that the high-end devices support these standards equally well out of the box. Another benefit is that the user is not required to install anything extra, as we can assume that all desktop computers and high-end mobile devices have a web browser installed by default. This implies that presentations that use an HTML front-end can easily be transported and shared.

- Another important reason to favour HTML5 is that HTML provides us with a lot of visualisation functionality out of the box. All basic components such as text, images, bullet list, audio and video are all present and can easily be fine-tuned with CSS [12].

- HTML5 is still an emerging standard, which means that it will remain valid for a very long time. Even though a lot of browsers already implement parts of the HTML5 standard [21], the official W3C Recommendation status for the standard is expected as late as 2025. This indicates that the standard is here to stay for a long time, and that a HTML5 implementation of our visualisation tool is in fact "future proof".

- Even though HTML already provides a lot of visualisation functionality on itself, but even if this proves to be insufficient we can easily extend HTML with JavaScript libraries, WebGL canvas' , Flash applications and Java Applets. Regarding the other candidates, HTML5 proves to be the most extendible with the added bonus that it also costs the least effort.

5.4 MindXpres Architecture

So far we have defined some loose components, but we need some kind of global structure that will define the implementation of the presentation tool. In this section we define a general architecture for the tool that will suit our needs.

As shown in Figure 5-6, the architecture we use in our solution consists of the following components:

1. XML Container Format
2. Compiler
3. Output Format (+ Visualisation Layer)

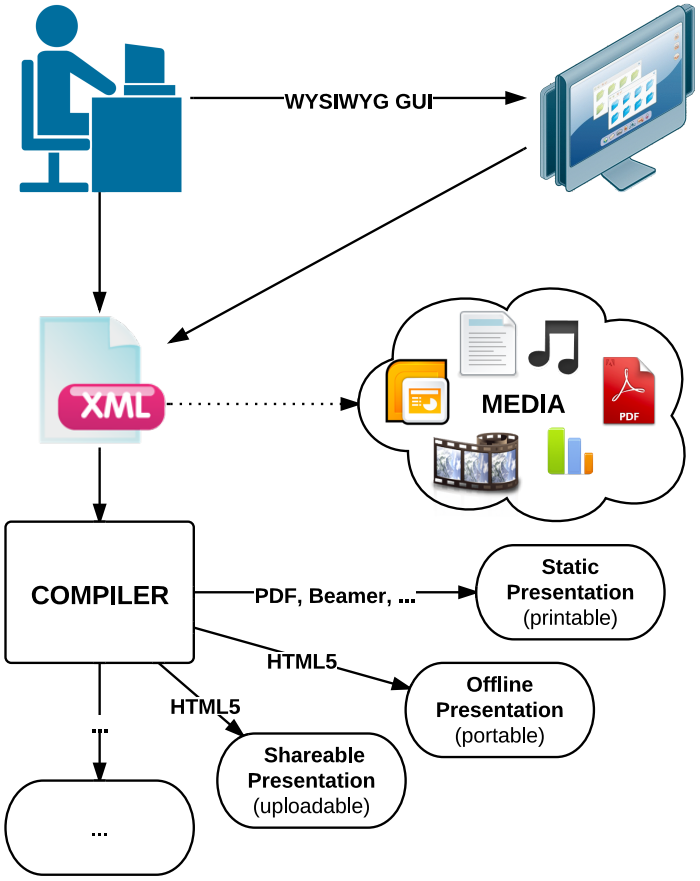


Figure 5-6: MindXpres architecture

Together these components span the entire process of using the presentation tool, from the creation of a presentation to the actual presentation itself. It all begins with the creation of content. The content (and thus the presentation that is represents) is stored in an XML container format. This is a predefined, machine-readable XML vocabulary to define content that will be used in the presentation. This format can be created manually, or optionally, a WYSIWYG Editor could generate the XML for you. Such an editor is however out of the scope of this thesis, and thus this has not been implemented. In the XML container format, XML tags are provided to make use of the provided presentation elements such as titles, images, bullet lists or any other elements provided by the visualisation layer.

Once the XML Container file has been created, this is given to the compiler, which will turn this XML document into an HTML document. This HTML document is basically an HTML representation of the original XML document, which makes it easier to visualise using JavaScript and related technologies. Essentially this is just a simple transformation from XML to HTML, but it allows us to add compile-time features to aid both the user and the visualisation layer. Examples of compile-time features include:

- Import data from exotic document formats such as Excel, and convert it to something usable by the visualisation layer.
- Create offline versions of a presentation, downloading all necessary material and including it in the presentation bundle
- Some resources might need conversion, e.g. HTML Video tag requires specific video formats (MP4, Ogg or WebM).
- Lighten the load of the visualisation layer, for instance by cropping, scaling and rotating images beforehand. Perhaps even pre-rendering the slides to images. These "light" versions could be useful for low-end mobile devices that can not handle the full visualisation library well.

Additionally, a compiler allows us to output the resulting presentation in different ways. Again, we provide some examples of potential output types:

- Static presentations for printing, for instance by generating a PDF file.
- Offline versions, where resources are pre-downloaded and bundled.
- A dynamic presentation version for the presenter, where are features are unlocked.
- A dynamic presentation for the audience members with limited functionality.
- A version of the presentation that can be uploaded, with added soecial features such as discussing slides, crowdsourcing, tagging or sharing.
- A “light” version of the presentation that is pre-rendered for low-end mobile devices, but that still offers some navigation features such as overviews and links.

Once the output has been created, it is ready to be displayed. In the case of PDF output, there is of course no need for a visualisation layer as any PDF viewer will do the job, but PDF output is just one of the many possible output types. Most of these output types will have an HTML document as output to make full use of dynamic, interactive visualisation and navigation.

In the case of an HTML output document, one must note that there is still a strict separation of content and visualisation. The visualisation layer is a single large JavaScript library that will visualise the content in the HTML output document. The HTML output document contains only the HTML representation of the content, with a single reference to the visualisation library which is entirely responsible for creating a real presentation out of the content-oriented HTML document. Thus, a presentation "package" or "bundle" consists of the main HTML document, possibly local resources (images, data, video, ...) and a copy of the visualisation library. Together these form a full presentation of the HTML output type, and starting the presentation is as simple as opening the HTML document in a browser (which may offer a fullscreen or presentation mode).

5.5 Extensibility Through Plug-ins

Almost all existing presentation tools are fairly limited when it comes to expandability. If some feature is not present it is often difficult, if not impossible to add it yourself. We wish to change that, by seeing the base presentation engine as a plug-in framework. Our presentation tool does not have a core with hardcoded components and aesthetics, but instead it offers a mechanism to plug in any of these in a user-friendly way. Our solution does not hide features internally; instead we chose to implement as much as possible as plug-ins. This allows third parties to replace, modify or add functionality, giving them access on a very low level. Plug-ins are developed in JavaScript, and adhere to a specific interface that allows the plug-in to interact with the presentation content. The tool ships with a pre-selected set of plug-ins, but adding new ones is as simple as placing the plug-in in a specific folder. Not only does this help developers to extend the tool, but it also makes it easier for regular users to download new plug-ins for the functionality that they wish. In our solution we provide three major types of plug-ins, which we discuss in the following subsections.

5.5.1 Components

The most obvious group of plug-ins are the components. By that we mean the the different content containers that provide visualisations and functionality for a specific content type. Examples of such components include bullet lists, images, video or source code. Such a plug-in decides how its relevant content is displayed, and how one can interact with it. Note that every content container is a plug-in in our solution, all the way down to the simplest content types (e.g. text). This allows third parties to modify or replace a basic content plug-in, changing how the content type is handled and visualised. Note that pure aesthetics such as font sizes, backgrounds or colours can be changed via themes, and thus plug-ins focus more on functionality and

implementation details. For instance, a video plug-in can provide video functionality through the HTML5 video component, but perhaps a third party wants to use a custom flash player for videos. This is a good example of where it would make sense to add a new plug-in, or modify the existing video plug-in.

5.5.2 Containers

We have discussed the concept of components, but now we need a way to group and organise components visually. This is where containers come into play. Containers are elements that contain components, and can provide functionality to help the user organise the components visually. An example of such a container is a slide. Each slide contains different content, but there are also some reoccurring elements. For instance, a slide can have a title, a slide number and the author's name. These common elements can be abstracted in a higher level container, taking tedious work out of the users' hands. On top of that, the variable content on a slide often required a specific layout. For instance, the user may want to display an image next to a bullet list. A container may provide functionality to help the user to lay out his content. For instance, by defining presets or allowing easy definitions of layouts. Because of this approach we stopped to see containers as mere bounding "boxes", but instead we decided to implement them as plug-ins that provide higher order functionality. Another example would be to implement a picture gallery as a container. This way, all images within the container could be organised and visualised in a specific way, automatically, e.g. in an interactive gallery. Note that a container can not only contain components, but also other containers (e.g. a slide can contain an image gallery container).

5.5.3 Structures

In Section 4.3.3 we have defined that it should be possible to have the ZUI lay out components on a larger scale. For instance by displaying element in a grid, or structured in sections as seen in Section 4.3.3. This layout functionality is again provided through plug-ins. One might now wonder what the difference is with containers, as these are also mechanisms for laying out content. The main difference is that structure plug-ins have ties to our XML language to define presentations. Similar to how LaTeX provides structure in documents (e.g. chapters, sections and subsections), the structure plug-ins can define structures that can be used from the XML language. This is sometimes necessary for complex visualisations like the structured layout, because the plug-in needs to know how to group the content. Hence, structures are more complex and on a higher level than containers. This is different from containers which only mark the beginning and the end of a set of elements that need to be contained.

5.6 Conclusion

In the previous sections we have explained how a tool can be built that fulfils the requirements defined in Chapter 4. By defining our own content-driven language we make the shift from a presentation tool as a graphical slide editor to a tool to visualise content. By using the concepts defined in the RSL model in our language, we make use of recent hypermedia research in order to provide a solution to many of the issues in existing tools.

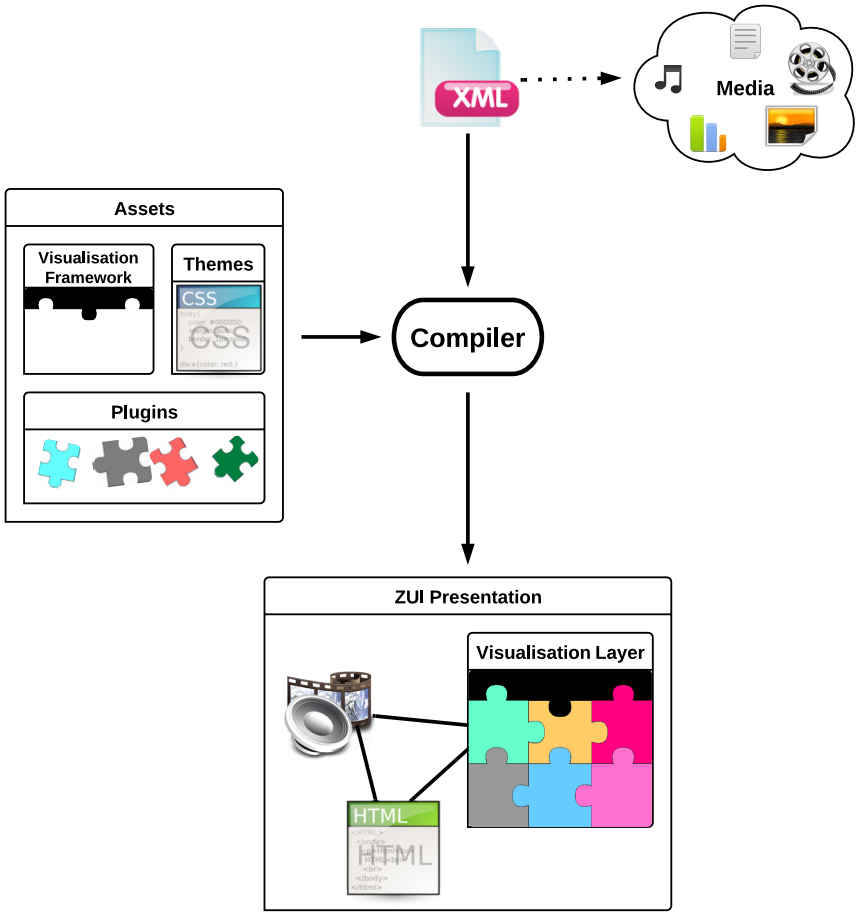


Figure 5-7: The MindXpres flow

This language is then processed by a compiler, transforming it into an HTML document. The resulting HTML document does not contain information regarding the visualisation, but is much like the original XML document, only in valid HTML and a lot less user-oriented. This allows a JavaScript visualisation layer to process the content at runtime, and turn it into a graphical presentation with visualisations that on par with what modern audience expect. The visualisation layer is heavily inspired by ZUIs and

spatial hypertext, and uses these concepts to avoid mistakes made by other slideware tools. This visualisation layer is built with extensibility in mind, and thus it is more of a framework than a library.
By implementing much of the visualisation and component functionality as plug-ins, it becomes nearly trivial to add additional content types or ZUI layouts. At compile-time the compiler bundles the framework with the referenced plug-ins, and together with the HTML content document a full presentation bundle is created. Depending on the users' wishes, the compiler can add the referenced resources in the bundle, for instance for offline usage. Because the result is in a format that is both open and well-supported, the presentation can be viewed on most devices, even portable and less powerful ones such as smartphones. In fact, a modern browser that complies to the HTML5 standards is all that is needed. A general overview of the concepts described in this chapter can be seen in Figure 5-7.
In the next chapter, we discuss the actual implementation of this tool. There we talk about the programming details, documenting how the solution suggested in this chapter is made into a functional implementation.

6 Implementation

In Chapter 4 we have defined a set of requirements for the ideal tool, and in Chapter 5 we have explained our approach to fulfilling those requirements. Now that we have established our approach, we can take this high-level design and turn in into a working implementation. In this chapter we explain how we implemented the design together with some important choices we had to make. In Section 5.4 we have presented the major components of the tool including the XML container format, the compiler and the HTML5 output (HTML format + visualisation library). We start this chapter by defining some goals. These are certain technical features that we wish to achieve with our implementation. Next, we handle the major components in order, describing our implementation and the choices we had to make.

6.1 Goals

We start this chapter by defining some implementation goals. These are general technical features that we deemed important for the prototype. By explicitly listing these goals, we hope to give the reader an idea of what we are building towards in the more technical parts of this chapter.

6.1.1 XML Authoring Language

- Define an XML Schema for the authoring language
- Create a basic presentation in this language

6.1.2 Compiler

- Validate an XML document against the schema
- Translate the XML to valid HTML

6.1.3 Visualisation Layer

6.1.3.1 ZUI

- Build a full-fledged ZUI with zooming, panning and rotating capabilities.
- Allow manual navigation with the mouse by dragging and scrolling.

6.1.3.2 Architecture

- Build a plug-in architecture for components, containers and structures.

6.1.3.3 Plug-ins

- Provide a sample set of basic plug-ins.

6.2 XML Authoring Language

As a starting point, we begin with the XML authoring language. In earlier chapters we established that content creation and definition via a LaTeX -like language has many benefits for users willing to learn the language. On top of that, it is always possible to build a graphical editor that generates this language, so there are no reason to not implement it as an interface to our tool.

In Section 5.2 we have proposed to use XML as a basis for our language, as it provides both the flexibility and restrictiveness that is needed for a machine-readable language that needs to be writeable by humans. In order to make such an XML document easier to compile, we want to make sure that the XML document is well-formed and that the language syntax is respected. Such a correct XML document is commonly called *valid*. In order to validate an XML document, we must first formally define our syntax. This includes the allowed tags, their attributes, the orders and how they can be nested. Such a definition for XML can be made in a Document Type Definition (DTD) or its more advanced successor XML Schema. We have opted for XML Schema as it allows users to define more complex document schemas. The benefit of formally defining a schema is that it can be used to verify an XML document. Any document that passes an XML schema validation is guaranteed to be in the way that you want it to be. For instance,

if the schema defines an optional tag attribute with a default value and the attribute is not present in the XML document, the validator will insert the default value. Because of this, we can make certain assumptions about the document later if it passes a schema validation (e.g. tag `X` always has an attribute `Y`, even if the user did not specify it).

It would be futile to present the final schema here and expect the user to understand it immediately. Instead, we will present some basic examples that demonstrate some concepts that we use heavily in the final implementation. Note that these examples are not per se representative for our final schema solution. In its most simple usage, XML Schema is fairly simple. For instance, the following example defines a document with a root element `presentation`, with two attributes (theme and layout). This element is a `complexType` meaning that it does not contain text, but other elements. Next, we list the elements that `presentation` can contain. By specifying them within a `choice` indicator[1], we say that these elements can occur in any order, any amount of times (including zero).

```
<xs:element name="presentation">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="slide" \>
      <xs:element name="text" \>
      <xs:element name="image" \>
    </xs:choice>
    <xs:attribute name="theme" type="xs:string"
        default="default"/>
    <xs:attribute name="layout" type="xs:string"
        default="structured"/>
  </xs:complexType>
</xs:element>
```

This would allow a simple XML document such as this:

```
<presentation theme="vub">
  <text>foo</text>
  <image/>
</presentation>
```

This document is valid, as `slide` is allowed to occur zero times and the layout attribute is optional. Note that the XML looks different after validation, since the default value for the layout attribute will be injected by the validator. This example is very basic and it is clear that we will need a lot more complexity. We need to be able to nest elements (e.g. components within containers) and some presentation components might provide some subtags for providing additional information or functionality. On top of that,

[1] http://www.w3schools.com/schema/schema_complex_indicators.asp

we need to be able to restrict the nesting since not all elements can contain other elements or only certain types. In the following example, we introduce some new concepts to make this possible.

```
<xs:group name="PresentationComponents">
  <xs:choice>
    <xs:element name="text" />
    <xs:element name="image" />
  </xs:choice>
</xs:group>

<xs:complexType name="Slide">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="title" type="xs:string" maxOccurs="1" />
    <xs:group ref="PresentationComponents" />
  </xs:choice>
</xs:complexType>

<xs:element name="presentation">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="slide" type="Slide" />
      <xs:group ref="PresentationComponents" />
    </xs:choice>
    <xs:attribute name="theme" type="xs:string"
        default="default" />
    <xs:attribute name="layout" type="xs:string"
        default="structured" />
  </xs:complexType>
</xs:element>
```

Listing 6.1: XML Schema type definitions and groups

First of all, we see a `group` element which allows us to collect a group of element definitions for later reuse. This is a form of transclusion where the content of the `group` block is "injected" in places where it is referenced, for example via `<xs:group ref="PresentationComponents" />`. This comes in handy when we want a set of elements to appear in multiple places. Next, we see a standalone type definition called `Slide`. Note that this block does not say that a tag named "Slide" is allowed. It is a type definition, describing the properties of the type `Slide`. This is comparable to a class definition in an object-oriented programming language. The `Slide` type provides two choices, either a `title` element or an element from the `PresentationComponents` group we defined earlier. This means that a slide can contain a title element, a text element or an image element. Finally, we define our root element `presentation`. It can contain two things, either a `slide` element (note that the type is set to `Slide`), or an element from the `PresentationComponents` group (text or image). This allows the following XML document to be defined:

```
<presentation>
  <slide>
    <text>content 1</text>
  </slide>
  <text>foo</text>
  <slide>
    <title>My Slide</title>
    <image />
    <text>content 2</text>
    <text>content 3</text>
  </slide>
</presentation>
```

As a final example, we wish to introduce one more concept. So far we have not seen how we can mix text and elements within an element. The following example defines a `text` element that allows some sub-elements within the text, for instance for formatting the text.

```
<xs:element name="text" >
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="b" type="xs:string" minOccurs="0"
          maxOccurs="unbounded" />
      <xs:element name="i" type="xs:string" minOccurs="0"
          maxOccurs="unbounded" />
      <xs:element name="u" type="xs:string" minOccurs="0"
          maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

This allows us to support constructions such as `<text>foo <b>bar</b> foo <i>bar</i></text>`. As we can see, an XML Schema can quickly become quite complex. One has to take into account orders, cardinality, types and nesting restrictions. We have shown the reader the basic concepts we use to make it manageable: type definitions, grouping and cardinalities. In our final solution, we build our schema file with in a hierarchical way by using these concepts, similar to how we defined a `PresentationComponents` group in Listing 6.1. Under the root node we have three main categories, namely `PresentationComponents`, `PresentationContainers` and `PresentationStructureElements`. Each of these groups contain other elements and may refer to other groups in order to allow nesting of components. Please refer to the source code that accompanied this thesis for the full schema.

There is however, one major problem with this approach. In order to build a schema like this, one needs to know all the allowed tags beforehand, which

is not the case. We have established that all our components, containers and structures are based on plug-ins, hence there is no fixed set of tags that is known beforehand. The solution to this is fairly simple: the compiler can scan the plug-in directories and generate the XML Schema based on the information that the plug-ins provide (e.g. the tags that they offer and their restrictions). However, due to the time restrictions the current implementation does not offer this feature. As a compromise, the implementation comes with a manually written schema that is sufficient for the goals defined in Section 6.1.

6.3 Compiler

Now that we have defined a container format via XML Schema, we can attempt to process and transform XML documents that claim to be a valid presentation. When deciding on the technology to use, we had to take into account some requirements. The compiler might need to do operations on media files, so it was clear that JavaScript would not be sufficient. Therefore, the compiler needed to be a full-fledged desktop application. We opted to use Java for some very simple reasons:

- The Java language is cross-platform, as it is a VM-based language. Java VMs for all major operating systems are provided by Oracle[1].
- The Java community provides us with all the libraries that we might need. This includes XML processing and parsing facilities that make up the major part of the compiler.
- Java is a well-known language and is often used in academic settings. This makes the project more accessible for other people who might wish to contribute.

In the following subsections, we will discuss the various components of the compiler, that will together transform the XML document into a presentation.

6.3.1 Document Validation

After an XML document passes validation, it is ready to be processed and transformed. Thanks to the XML Schema validation, we can assume that the document is valid and that optional attributes have had their default values injected by the validator. Our decision to use Java made this process fairly simple. Modern Java versions offer built-in XML processing via the

[1] http://www.java.com/en/download/

Simple API for XML (SAX). Validation is fairly straightforward, as shown in Listing 6.2.

```
1 Schema schema = loadSchema(schemaPath);
2 Validator validator = schema.newValidator();
3 validator.setErrorHandler(new ValidationErrorHandler());
4 SAXSource source = new SAXSource(
5     XMLReaderFactory.createXMLReader(), new InputSource(new
          FileInputStream(filePath)));
6 validator.validate(source);
```

Listing 6.2: SAX document validation

The moment the document deviates from the XML Schema, an exception is sent to the provided `ErrorHandler` class. The exception contains detailed information on what is wrong, which we use as feedback for the user, making it easier to pinpoint potential problems.

6.3.2 Parser

Once a document has been successfully validated, we can start processing the XML language. There are two main approaches to traversing the XML document.

Document Object Model (DOM): The DOM presents an XML document as a hierarchical tree structure. As an XML document always has a single root element, it makes it easy to traverse the tree with common tree traversal algorithms (e.g. depth-first or breadth-first traversal). The major downfall of this approach is that the entire document is parsed and placed in memory before one can start the traversal. Depending on the size of the document, this can be quite memory consuming.

Simple API for XML (SAX): The built-in SAX API provides means of traversing an XML document that is different from the DOM approach. Instead of loading the entire document into memory, the document is processed *sequentially* and *events* are generated as elements are encountered. This allows the parser to step through the XML without knowledge of previous or future elements.

The choice between the SAX and the DOM approach often depends on the required functionality. If one needs to be able to move, sort or replace elements, a DOM approach makes more sense as we have access to the entire element tree at all times. On the other hand, if it is sufficient to traverse the document sequentially from start to end, without knowledge of past or future elements, SAX is the way to go. Good uses of such a sequential traversal include indexing, conversion to other formats or formatting. In our

context, it makes more sense to use the SAX approach. Each element has to be traversed anyway and it is ideal for converting the encountered tags into another format (namely HTML in our case). As the order of the document has already been determined by the XML Schema, there are no cases where we would want to make changes to the location or order of elements, which again makes the SAX approach the better choice.

As the document is traversed by the SAX parser, we are notified of its progress via the event that it generates. There are different types of events that help us to determine what to do with the information:

- Document Start: Invoked when the parser starts parsing the document.
- Document End: Invoked when the parser reaches the end of the document.
- Element Start: Invoked when an element is encountered.
- Element End: Invoked when an elements is closed.
- Ignorable Whitespace: Invoked when ignorable whitespace is encountered. For instance, the space in `<a></a> <a></a>` is ignorable, as it not part of the XML language, nor is it content.
- Characters: Invoked when content is encountered. For instance, in `<a>foo</a>`, "foo" is content that will trigger this event.

When such an event is thrown, it comes with all necessary information, e.g. the tag name or its attributes. Because the document is guaranteed to be traversed sequentially, we can write away the transformed result for each event and end up with a valid document. This principle is illustrated in the simple examples shown in Table 6-1.

Input	Event	Output
<text>	Element Start (text)	
foo	Characters	foo
</text>	Element End (text)	

Table 6-1: How to use sequential SAX parsing

By catching the rights events, we are now able to traverse the document and handle each event on its own. By using the event information, we can start outputting valid HTML for each event that is received. In the next subsection we will give the exact details of how we transform the data.

6.3.3 Transforming

While traversing the XML document, we wish to transform the elements to valid HTML as we encounter them. Note that it is important that the HTML is valid. This implies that we cannot use custom tags or attributes that are not in the HTML5 standard [21]. One could make use of existing tags for some data types (e.g. the `image` tag for images), but this would complicate things as not all content types have matching HTML tags.

To solve this issue, we opted to transform all our custom XML tags into the same HTML tag, namely the `div` tag. In order to avoid information to be lost (e.g. the tag name) we somehow have to keep it stored in the `div` tag. However, we must keep in mind that we are not allowed to use custom tag attributes, as this would invalidate the HTML. Therefore, we should either abuse existing attributes for this purpose or look for another solution. Encoding information like the tag name and its attribute inside a single existing HTML attributes would be possible, but it is not the cleanest solution. Luckily, the HTML5 standard provides a mechanism to define custom non-visible tag attributes. It is also very easy to use: *all attributes that start with "data-" are considered valid.* For example, `<div data-foo="bar" />` will pass validation. In addition, these custom attributes do not influence the visualisation in any way and are in fact designed for storing information.

Because the XML document that we want to transform has been validated, we are allowed to make certain assumptions about it. For instance, it has specific attributes that are guaranteed to be there (either specified by the user, or a default value injected by the validator), and it only has attributes that are allowed. These assumptions make the transformation fairly straightforward as a basic compilation does not require real reasoning. An custom XML element from the XML document can be transformed to valid HTML simply by converting the tag name and its attributes to `data-*` style attributes for the `div` tag. Allow us to demonstrate this with a simple example:

```
1 <presentation theme="vub" author="Reinout Roels">
```

This custom XML tag can easily be transformed into valid HTML without reasoning. We simply create a `div` tag with the attribute `data-type` to hold the original tag name and prefix the existing attributes with `data-`:

```
1 <div data-type="presentation" data-theme="vub"
2     data-author="Reinout Roels">
```

Implementation-wise, we just need to handle the SAX parser events in the following way to get a valid HTML document:

- Document Start: We copy the content of a template file onto the output stream. This file contains the opening HTML tag, together with a basic header that invokes the visualisation layer (see later).
- Document End: Again, we copy the contents of a template file onto the output stream, which contains some elements that finalise the HTML document.
- Element Start: We convert the element to a `div` tag, encoding the tag name and the attributes as custom HTML5 attributes.
- Element End: We close the `div` tag
- Ignorable Whitespace: This event is irrelevant is can be ignored.
- Characters: When content is encountered, it is simply copied to the output stream without modification.

As we can see, the compilation process is fairly simple. Originally, we thought of using XSL transformations [7] to achieve the same result. An XSL transformation is basically a mechanism that allows the user to apply transformation rules to content that matches specific selectors (XPath [52] expressions). However, XSL transformations give us little control over the reasoning behind the transformation process. While the current SAX-based parser does very little with the XML data, the sequential traversal could allow the parser to do more than simple conversion. It became clear that a SAX approach would be better for future work. As an example, the compiler could for instance intercept unsupported video resources and automatically convert them to a format that is supported by the video plug-in. With this in mind, we decided to do it right from the beginning and take a real parsing approach instead of XSL transformations.

6.3.4 GUI

In addition to command-line interaction, we have provided a simple Swing GUI in order to make it more accessible for users that prefer a graphical interface. As shown in Figure 6-1, the GUI itself is fairly simple. Two text-boxes are provided to enter the source file (the XML document), and the output directory where the result will be placed. Buttons marked with "..." can be used to select the file or folder via the system file chooser. On top of that, you can also drag the source file onto the GUI to set its path. Once the user presses the compile button, the larger text area is used for feedback, marking successful steps in green and issues in red. As Swing is a built-in part of the Java VM, this GUI works on all platforms that run Java.

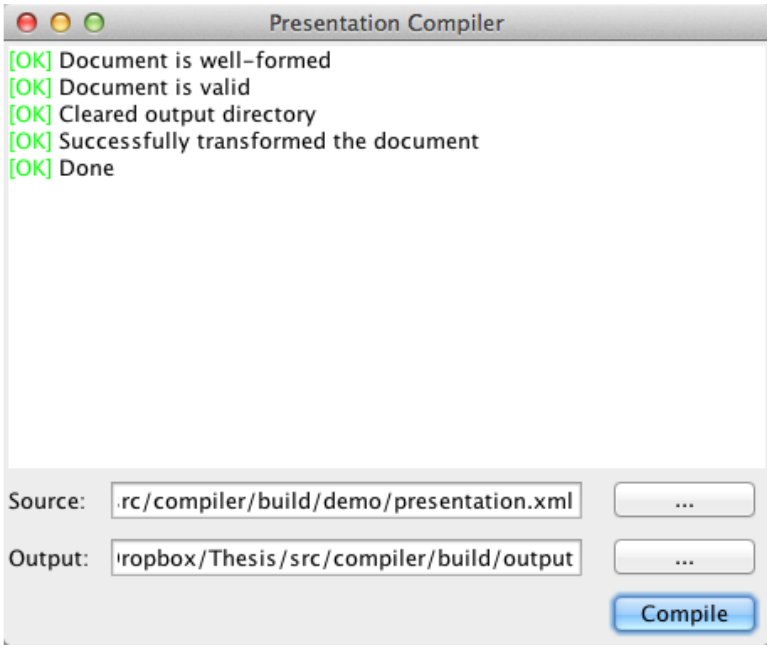


Figure 6-1: The compiler GUI

6.4 Visualisation Library

In the previous section we have detailed the transformation from our custom XML format to valid HTML. It is clear that the resulting HTML document is in no way a presentation. It is merely an HTML version of the content-only XML document. In order to turn this raw data into an attractive and meaningful presentation, we need to apply the aesthetical aspects. This includes spatially laying out the content, applying colours, backgrounds, fonts and visualising data in interactive components. All of this is done at *runtime*, when the HTML document is opened. This is the job of the visualisation library, which is written in JavaScript. By making use of the latest HTML5 features, this JavaScript library is able to transform the plain content into a fully-fledged ZUI with all the bells and whistles we discussed earlier. In the next subsections we provide more details on how this library works.

6.4.1 Library Architecture

For a JavaScript library, its tasks are on quite a big scale. On its own, the HTML document only contains content, meaning that the library is responsible for the layout, the ZUI, the user interaction and the plug-in mechanism, or in other words, the presentation minus the content. In order to keep everything manageable, it is important to have a solid architecture that allows easy manageability from a software engineering standpoint.

The first hurdle comes with the fact that such a library consists of multiple files. On top of that, there may be many plug-ins that come with their own JavaScript files. Having to include all these files in the HTML header is both unnecessary and not very elegant. Ideally, we should have a single reference to the visualisation library and let the library handle the rest from there on. Figure 6-2 shows the architecture we used for our implementation of the visualisation framework.

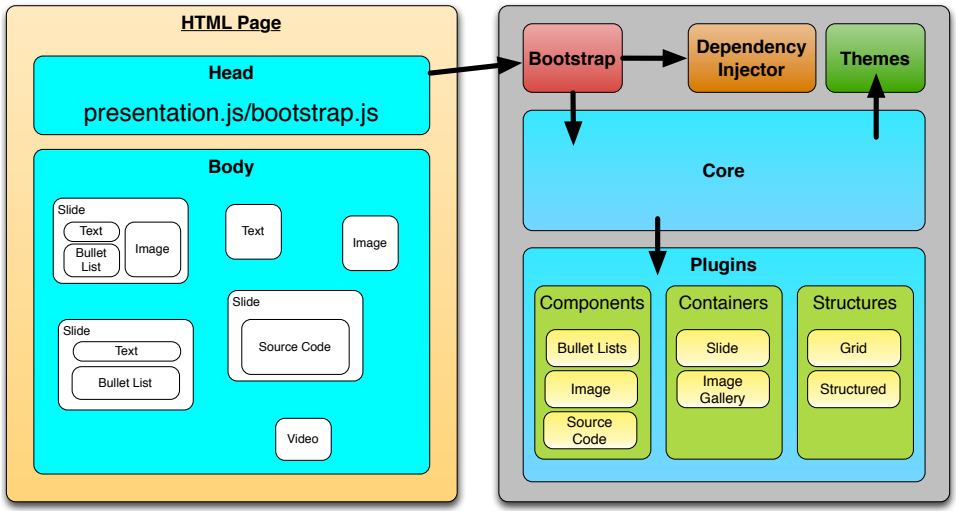


Figure 6-2: Library architecture

We simply include one JavaScript file (the bootstrap process) and the rest happens automatically from there, driven by the HTML content. In the next sections we discuss the different smaller elements that compose the visualisation library.

6.4.2 Dependency Injector

In the previous section we mentioned that we needed an elegant solution for managing JavaScript and CSS dependencies. Without a decent solution, our code gets a tangled mess of hard coded dependencies which would quickly become unmanageable. Ideally, an *include* mechanism like the one in C or C++ could be used to include dependencies where they are needed. Sadly, JavaScript does not have such a built-in mechanism. Yet, we needed a mechanism for loading dependencies on demand for multiple reasons. The most important reason is that it would be a waste of memory and performance to load dependencies that are never used. One could argue that the effect is negligible on modern computers, but we must also keep in mind that we target weaker mobile devices such as smartphones and tablets. Furthermore,

it would simply be wrong from a software engineering standpoint to not to aim for the most efficient solution.

There have been previous attempts for JavaScript dependency injection libraries, for instance using.js[1] or RequireJS[2]. However, all of them lack pieces functionality that we deemed critical. Some of our criteria include the need for support for both JavaScript files and CSS files, or being able to reference dependencies via aliases. Another important requirement was that it should be possible to be notified when all asynchronous inclusions are finished, even if they have not been initiated in the same batch. Ultimately, we deemed this aspect important and doable enough to build it ourselves, in order to make sure it meets our requirements.

Our dependency injector, which we will reference with DI from now on, is used as a static library. Using a dependency consists of two steps.

First we must register the dependency with the DI by calling the register function with an alias, the path to the resource and optionally the resource type (CSS, JS) if this cannot be derived automatically from the file extension.

```
1 DI.register("jquery",
2     "presentation.js/libs/jQuery/jquery-1.7.1.min.js");
```

Next, we can include our resource by its alias, optionally providing a callback function that is to be called after the resource has been fully loaded. If we do not use the optional callback function, no guarantees are made that the library will be loaded by the time you first call it, as this is handled asynchronously by the browser.

```
1 DI.include("jquery", function(){
2   // do stuff
3 });
```

By using simple aliases, the user does not need to remember or look up the full path of the resource every time they need it. As resources are included on demand, it is possible (and likely) that such an include statement for the same alias occurs at different locations in the project. For this reason the library keeps track of all registered resources and marks them if they are already loaded. Because of this subsequent include calls with the same alias will still result in only one injection. To keep the code clean, all resources can be pre-registered in a single file which is included after loading the dependency injector.

[1] http://www.jondavis.net/codeprojects/using.js/test.html
[2] http://requirejs.org/

The injection itself is done by simply injecting tags into the DOM-tree, more specifically into the head section of the HTML document:

For CSS files:

```
<link rel="stylesheet" type="text/css" href="mycss.css">
```

For JavaScript files:

```
<script type="text/javascript" src="myscript.js">
```

The following function injects a tag of type `type`, gives it the attributes defined in the associative array "attributes" and hooks the callback function to the node `onload` event.

```
function injectHeadElement(type, attributes, callback){
  var el = document.createElement(type);
  for(attr in attributes){
    if (!attributes.hasOwnProperty(attr)) {
      continue;
    }
    el.setAttribute(attr, attributes[attr]);
  }
  document.getElementsByTagName("head")[0].appendChild(el);
  el.onload = callback;
  return true;
};
```

Various helper functions have been created to aid the user, for instance to register and include resources in one call, to include many resources in one batch (array of aliases), or a function that will call a given callback once ALL included resources are fully loaded, even if they were included with separate "include" calls (e.g. "notify me when you're done handling all my include calls").

Some browser have a very aggressive caching policy. Especially during development, this became quite bothersome as emptying the cache for every small change is the only solution for these browsers. In order to bypass these caching issues, the library optionally appends a unique string file path in order to force a reload. For instance, "foo.js" becomes "foo.js?r=123456789". This extra URL parameter has no effect on the loading of the dependency, yet it triggers a reload, which is exactly what we want. In order to avoid potential collisions in the strings, we use the current time in milliseconds as the base for our unique string. For the rare chance that two libraries are loaded during the same millisecond, a small random string is prepended, reducing the odds of a collision to virtually nothing.

6.4.3 Bootstrap

As mentioned previously, the bootstrap file is the only file that needs to be referenced from the HTML content document. Everything else is automatically loaded from there. Observant readers may have noticed a small problem with the dependency injector: if the HTML file only includes the bootstrap file, and other JavaScript resources can only be included with our dependency injector, how is the dependency injector loaded? This is typical "chicken and egg" problem, and this is exactly why the bootstrap file exists. The bootstrap file's main function is to inject the dependency injector, and load the main library file that starts up the entire visualisation process.

As the dependency injector can't be used yet, the bootstrap file includes a small function that includes just the dependency injector, in a manner similar as described in Section 6.4.2. The entire bootstrapping process is as simple as this:

```
1  var presentationLibName = "presentation.js";
2  var pluginDir = presentationLibName + "/plugins/";
3  var libDir = presentationLibName + "/libs/";
4
5  // bootstrap the dependency injection system
6
7  bootstrap(presentationLibName + "/libs/di.js", function(){
8    bootstrap(presentationLibName + "/di_dependencies.js", function
       (){
9      DI.batch_include(["debug", "jquery"], function(){
10       DI.callWhenReady(function(){ // make sure jQuery and Debug
           are fully loaded
11         DI.include("presentation", function(){
12           $(document).ready(function() {
13             Presentation.init();
14           });
15         });
16       });
17     });
18   });
19 });
20
21 function bootstrap(path, callback){
22   var el = document.createElement('script');
23   el.setAttribute("type","text/javascript");
24   el.setAttribute("src", path + (cacheHack ? "?r=" +
       getUniqueString() : ""));
25   document.getElementsByTagName("head")[0].appendChild(el);
26   el.onload = callback;
27   return true;
28 };
```

The `di_dependencies.js` file contains all registration calls for the libraries that we are sure to use later on. This way the main presentation file can assume these libraries have been registered, and include them by their alias when needed. By keeping all these registrations in a single file we also reduce the clutter in the code.

6.4.4 Core

The core is the code that glues everything together, and it is also the real entry point for the visualisation library. The core is responsible for loading themes, plug-ins, handling keyboard input and doing the navigation. When the bootloader first loads the core, the `init` method is called, which is the real entry point for the library. This initialisation method starts by making some important changes to the DOM-tree. To facilitate visualisation, some layers are added above the DIVs which were originally at the root of the document.

1. Presentation Container: By grouping all presentation-related nodes under a single presentation node, we allow other HTML elements, unrelated to the presentation, to live at the document root without interference from the visualisation library. As CSS style rules trickle down the hierarchy, this root presentation node also allows us to apply basic CSS properties to the entire presentation subtree.

2. Perspective Layer: One can think of this layer as a lens or magnifying glass. Scale transformations are applied to this layer, changing the scale of everything underneath. This layer is the only child of the presentation container.

3. Canvas Layer: This layter acts as the parent container for all presentation elements (structures, containers and components). The rotation and panning transformations are applied to this layer. This layer is the only child of the perspective layer.

The perspective layer and the canvas layer could be combined, but we have chosen not to. By separating these two, it makes thinking about complex animations much easier, because it much more intuitive. Most animations consist of sliding the canvas layer for a certain distance, and applying a scale to the perspective layer. With a two-layer abstraction it is easier to reason about the desired effect, e.g. “The canvas needs to slide 500 pixels to the left, while the scale needs to decrease by a factor of 0.7”. Additionally, it also facilitate the temporal aspects of animations, as it is possible to have overlapping intervals of animation on the two layers. For instance, if the panning animation is done over a span of one second, we can delay the zooming animation to when the panning animation is halfway, for smoother animation.

After these layers have been created, all DIVs which used to be at the document root, are now moved to the canvas layer. Then the core iterates over the plug-ins which transform the matching DIVs into something useful (see the next section for a more detailed explanation).

The core provides various functionality and helper functions towards the rest of the library. One noteworthy helper function is the `applyCSS` method. Applying CSS to elements at runtime via JavaScript is not per se difficult. However, many of the new CSS3 features often have browser specific names, for instance Listing 6.3 shows some CSS rules that all do the same, but where each browser requires a different prefix.

```
1 background: -webkit-gradient(radial, 50% 50%, 0, 50% 50%, 500,
      from(rgb(240, 240, 240)), to(rgb(190, 190, 190)));
2 background: -webkit-radial-gradient(rgb(240, 240, 240), rgb(190,
      190, 190));
3 background:    -moz-radial-gradient(rgb(240, 240, 240), rgb(190,
      190, 190));
4 background:      -o-radial-gradient(rgb(240, 240, 240), rgb(190,
      190, 190));
5 background:         radial-gradient(rgb(240, 240, 240), rgb(190,
      190, 190));
```

Listing 6.3: An example of browser-specific CSS rules

This is the same base property (a gradient background), but every major browser has given it a browser-specific prefix, and it will only work in a given browser if you refer to it by its browser-specific name. To facilitate development, the `applyCSS` method automatically converts any CSS property to the browser-specific variant if needed. The method that does this is based upon one we encountered in the impress.js[1] project, which in turn got it somewhere else. It seems that this quite a common piece of code in the JavaScript community, and we were not able to determine the original source. The exact method is shown in Footnote 1.

```
1  var style = document.createElement('dummy').style;
2  var prefixes = 'Webkit Moz O ms Khtml'.split(' ');
3  var memory = {};
4
5  var browserSpecific = function(property) {
6
7    if ( typeof memory[ property ] === "undefined" ) {
8
9    var ucProp  = property.charAt(0).toUpperCase() + property.substr
         (1),
10   props   = (property + ' ' + prefixes.join(ucProp + ' ') + ucProp
         ).split(' ');
```

[1] http://bartaz.github.com/impress.js

```
11
12    memory[ property ] = null;
13    for ( var i in props ) {
14      if ( style[ props[i] ] !== undefined ) {
15        memory[ property ] = props[i];
16          break;
17        }
18      }
19    }
20
21    return memory[ property ];
22  };
```

The `applyCSS` function is one of the methods that is used the most, as a major part of the visualisation relies on CSS transforms. CSS transforms are used for both positioning elements, and animating the view via the perspective and canvas layer. The custom HTML5 attributes are used for storing spatial modifiers. For instance:

```
1  <div id="slide_1" data-x="0" data-y="800" data-rotate="30"></div>
```

These modifiers are either as a result of the compiler translation, or can be set by plug-ins. For instance, the structure and container plug-ins may determine the layout for their children. The core provides a method called `transformByAttributes`, which can be seen in Listing 6.4. This method simply reads an element's attributes and uses them to apply the necessary transforms rules on the element.

```
1  this.transformByAttributes = function(el){
2    var params = getTransformParameters(el);
3    var transformString = "";
4    transformString += makeTranslate(params.translate.x, params.
         translate.y, params.translate.z);
5    transformString += " " + makeRotate(params.rotate.x , params.
         rotate.y, params.rotate.z);
6    transformString += " " + makeScale(params.scale);
7
8    DOMTools.applyCSS(el, "transform", transformString);
9    DOMTools.applyCSS(el, "box-sizing", "border-box");
10   DOMTools.applyCSS(el, "transform-origin", "top left");
11 };
12
13 var getTransformParameters = function(node){
14   var data = node.dataset;
15   var translate = { x: data.x || 0, y: data.y || 0, z: 0};
16   var rotate = { x: 0 , y: 0, z: data.rotate || 0};
17   var scale = data.scale || 1;
18   return {translate: translate, rotate: rotate, scale: scale}
19 };
```

Listing 6.4: Transforming elements by their attributes

One of the other major features that is offered by the core is the animation functionality, mainly used for navigation. In the beginning of this section we mentioned that some layers are inserted between the content and the root node, the most important ones being the perspective layer and the canvas layer. Intuitively one could say that the perspective layer acts as a programmable magnifying glass, and that the canvas layer scrolls the entire canvas to put our desired target under the magnifying glass. Imagine that we want to zoom on a particular subregion of the canvas where all components are visualised. This is done via two operations, we must pan the canvas so that the target subregion is centred, and we must apply a scaling transformation so that the area fits the screen width perfectly.

By separating the scaling and panning transformation onto these two different layers, it allows us to run these two animations in parallel, but with keeping the coordinate system used for panning intact, as it is not influenced by the scale. It also allows us to get creative when applying transformations, as now they are not required to be run at the same time anymore. We could run them sequentially, or have them overlap in any way we wish.

Ultimately our entire ZUI is based around the `focusOnRect` function shown in Line 19. It allows us to focus on a specific sub-rectangle of the canvas layer. The function will then calculate the scale that will make the target element fill the screen. Next, the canvas will be panned so that the rectangle is centred, and the scale transformation is applied to the perspective layer to create a zooming animation.

```
1  var focusOnRect = function(x, y, width, height, animated){
2
3    if(!isDefined(animated)){animated = true;}
4
5    var rect = {x: x, y: y, width: width, height: height};
6    currentRect = rect;
7
8    var viewWidth = parseInt(getComputedParameter(canvasNode,
9        "width"));
10   var viewHeight = parseInt(getComputedParameter(canvasNode,
11       "height"));
12   var maxScaleSize = Math.min(viewWidth / rect.width, viewHeight /
         rect.height);
13
14   var newX = -rect.x + (viewWidth - rect.width ) / 2;
15   var newY = -rect.y + (viewHeight - rect.height ) / 2;
16
17   var zooming = maxScaleSize >= currentScale;
18   currentScale = maxScaleSize;
19
20   if(animated){
```

```
21      DOMTools.applyCSS(perspectiveNode, "transitionDelay",
22          (zooming ? "500ms" : "0ms"));
23      DOMTools.applyCSS(perspectiveNode, "transitionDuration", "1s")
            ;
24    }else{
25      DOMTools.applyCSS(perspectiveNode, "transitionDelay",  "0ms");
26      DOMTools.applyCSS(perspectiveNode, "transitionDuration", "0s")
            ;
27    }
28    DOMTools.applyCSS(perspectiveNode, "transform",  makeScale(
          maxScaleSize));
29    DOMTools.applyCSS(perspectiveNode, "transformOrigin", "50% 50%")
          ;
30
31    if(animated){
32      DOMTools.applyCSS(canvasNode, "transitionDelay",  (zooming ? "
            0ms" : "500ms"));
33      DOMTools.applyCSS(canvasNode, "transitionDuration", "1s");
34    }else{
35      DOMTools.applyCSS(canvasNode, "transitionDelay",  "0ms");
36      DOMTools.applyCSS(canvasNode, "transitionDuration", "0s");
37    }
38    DOMTools.applyCSS(canvasNode, "transform",  makeRotate(0,0,0) +
          " " + makeTranslate(newX, newY, 0));
39
40  };
```

This is the backbone of all navigation-related functionality. To begin with, the presentation is focused on the rectangle that bounds all elements, giving a global overview. Next, individual elements are focused either because they are part of the navigational path, or because the user has chosen to focus on an element. In both cases this is achieved by simply calling `focusOnRect` with the element's bounding rectangle as parameter. Users are also allowed to navigate manually with the mouse, which again relies on the `focusOnRect` method. By dragging the mouse over the canvas, the user can manually pan the view. Because dragging events are produces as they happen, we can calculate the distance between the current mouse position and the last one (the delta value) and simply shift the view rectangle by this amount. Similarly, the user can use the scroll wheel to zoom in or out around the position of the mouse cursor. This is done by calculating the rectangle in the would-be scale, placing it around the mouse cursor and then calling `focusOnRect` with the calculated rectangle as parameter. The calculations are fairly complex, as it one has to take into account the scaling, panning and rotating factors. On top of that, the point under the mouse cursor has to remain the same point in the scaled rectangle, which means one has to shift the canvas a certain amount to make up for the shift cause by the scaling transform. However, these calculations are out of the scope for this thesis, so please refer to the full source code for the method itself. During a drag or scroll operation, many of these events are fired per second, meaning that

this method is called in very short succession when navigating manually. Because the operations need be smooth, the `focusOnRect` method provides the option to disable transition animations, making the transforms apply instantly. Even on mediocre hardware this approach provides a smooth experience, similar to what users are used to in other ZUIs (e.g. Google Maps).

Up to this point we have discussed how we visualise elements, how CSS is used to position them and how animation is done. However, this does not explain how we turn the bland HTML-based content document into a true presentation. Earlier we have discussed our plans for making the visualisation of specific content-types plug-in based. The core is what ties everything together and is thus responsible for loading the plug-ins. The reader might remember that are are three categories of plug-ins: structural plug-ins, container plug-ins and component plug-ins. Every time a plug-in is loaded, it is handed the elements that depend on that plug-in to be visualised. Therefore the order in which the plug-ins are loaded is not to be neglected. In fact, there is only one order that is optimal, namely first the components, then the containers and then the structures. This is because plug-ins can change the dimensions of the elements that they visualise. By expanding the components first, we make sure that the containers receive the final dimensions of their children. Similarly, once the structures are loaded they are provided with the final dimensions of their child elements, which can be both containers or components. In the following section we present the inner workings of the plug-ins.

6.4.5 The Plug-in Architecture

Visualisation of information is a major component of MindXpres, and is mostly driven by plug-ins. The component plug-ins visualise the different content types, ranging from simple bullet lists to interactive 3D visualisations. The container and structure plug-ins focus on grouping and laying out the elements within their bounds. Each plug-in is responsible for transforming its corresponding `div` elements in the HTML document to the visualised version that will be seen by the audience. Such a transformation can be as simple as applying CSS classes to the element (e.g. setting the font size or colour for a text plug-in), but this can also go as far as DOM-tree modifications. For instance the video plug-in modifies each its corresponding `div` elements by injecting a sub-`div`, which in turn contains a HTML5 video tag and another `div` that is used for rendering subtitles. As each plug-in is responsible for transforming the corresponding `divs`, and because hierarchies are always respected this eventually results in a fully transformed DOM tree. Elements are never moved in the tree, only wrapped, and so the HTML document that initially only contained content and structure is augmented with the visualisation code that will make it a real presentation.

Plug-ins are contained into individual folders to keep things clean. Such a plug-in folder includes all JavaScript, CSS, Images and other resources that are relevant to the plug-in. This allows for easy plug-in management, as adding or removing plug-ins is as easy as adding or removing plug-in folders in the plug-in directories. JavaScript does not provide functionality to scan filesystems, so the plug-in mechanism is achieved through naming conventions. Every plug-in must contain some files and object that follow specific naming conventions. By following these conventions, the core knows what to load, and how to access the loaded information. The most important file in a plug-in is is the file called plugin_info.js. This file contains a small object that specifies some plug-in-specific information, such as the tags that it provides for the XML authoring language. In Section 6.3.3 we described that when such a tag is used, it is transformed to a `div`, while storing the original tag name in a custom attribute. This way, when a plug-in is loaded, we first load plugin_info.js. The core then reads the tags it provides, and scans the HTML for all `div` elements that have the attribute `data-type` with the value of the provided tags. If none are found, the core simply moves on to the next plug-in. Otherwise, the names of the other plug-in files are read from the plug-in information object and are injected.

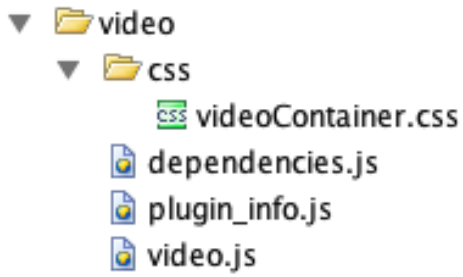


Figure 6-3: The plug-in structure

The plug-in information object also contains the name of the main plug-in object, which is initiated by calling its `init` method. Next, the detected elements are handed to the plug-in via its `process` method. The plug-in is then free to transform the element as it sees fit. This can be by simply applying some CSS, but it can also do more advanced things such as injecting other elements or subscribing elements to DOM events with its own callback functions.

6.4.6 Implemented Plug-ins

In this section we quickly mention the important details for some of the plug-ins we implemented. However, these descriptions will be fairly brief, so

please refer to the full source code for the actual implementation.

Component: Text The text plug-in is fairly trivial. As `div` containers are text contains by default (among other things), we merely need to apply standard CSS rules for font and colour properties. Text tags take attributes that alter the font, the size or the colour. Additionally, the text plug-in provides three helper tags: <b>, <i>and <h>. There make the text within them bold, italic and highlighted respectively. Again, these effects are achieves with simple CSS rules.

Component: Bullet List The bullet list plug-in uses the built-in HTML bullet list functionality (`ul` and `ol`). Via CSS, these elements can be styled, for example to change the bullets. We have opted to use the built-in bullet lists in order to make use of the functionality that is already there, for instance the wrapping and nesting of items.

Component: Timer The timer is a basic but useful plug-in that allows users to display a timer on a slide. Such a timer has many purposes, for instance a count-time timer to indicate a break, a counting timer that indicates the current duration of the presentation, or simply the current time. From an implementation standpoint this plug-in is trivial. The plug-in simply uses the built in JavaScript timer functionality to render the time (in text) to its designated `div` element. For the dynamic timer types (e.g. countdown timer) callback functions are given to the JavaScript timers to update the `div`.

Component: Image Images are a must-have for a presentation tool. Our image plug-in is based upon the HTML `img` tag, and behaves similarly. On top of that, our image tags allow so-called *filters*. For instance, within the image tags one can use the `crop` filter than will cut out the selected part from the original image. For the prototype, this is the only implemented filter, but the possibilities are endless. One could implement filters for colour modifications, transformations (e.g. shearing or flipping) or augmenting the image (e.g. shapes or text).

Component: Video The video plug-in is built upon the HTML5 video player, but augments it in some significant ways. First of all, one can provide subtitles for a video. One simply provides the time in a common format (e.g. “5s” or “4:12”) together with the text to be displayed. Similarly, one can create overlays for the video, displaying specific shapes on top of the video for a specified amount of time. Finally, we also provided timed control over the video playback. For instance, one could ask the video to stop playing at a specific time. Alternatively, one could ask the video to pause for a specific time, while overlaying the video with shapes or text. All this

functionality is achieved by wrapping the underlying HTML video tag in a parent `div`. Next, two sibling `divs` are added under this parent, and are spatially matched with the position of the video. One of these is used for displaying subtitles while the other is used for rendering shapes. By combining built in JavaScript timers with the interface the video player provides, we can sync our events with the playback of the video and provide these features. Finally, by making use of the provided video API we were also able to implement bookmarks within videos.

Component: Source Code Presenting source code is often a major pain. One has to either manually apply syntax highlighting and formatting, or use external tools, both of which are tedious and time-consuming. The source code plug-in works much like code listings in LaTeX, allowing the user to simply paste his code between `code` tags and let MindXpres do the rest. Our plug-in is based upon the open source SyntaxHighlighter library[1], which handles the visualisation of the source code.

Container: Slide To make the tool attractive for potential users, we wanted to make sure not to scare them away with complex semantic graphs and whatnot. To ease the transition to MindXpres, we provide a slide container that behaves similar to what people know from slideware tools. Slides are containers with a title and a number, that provide easy means of laying out the content within. One of these layout features is the ability to partition the available space by means of simple expressions that can be passed as an attribute.

These layout expressions are very intuitive. A column is started with the pipe character (`|`). The number that follows this character is the width expressed as a percentage of the available space. A pipe symbol indicates a new column. For instance `|40|20|40` divides the slide in three columns, 40%, 20% and 40% wide respectively. Similarly, the underscore character divides is used for rows, but the trailing number is now the height of the row, expressed as a percentage of the available space. The asterisk wildcard character (`*`) can be used to take up the available space that is left. If multiple asterisks are present in, the available space is equally divided among the asterisks. To illustrate this, Figure 6-4 shows the visualisation of the expression `_40|40|*|40_*|50|50`.

[1] http://alexgorbatchev.com/SyntaxHighlighter/

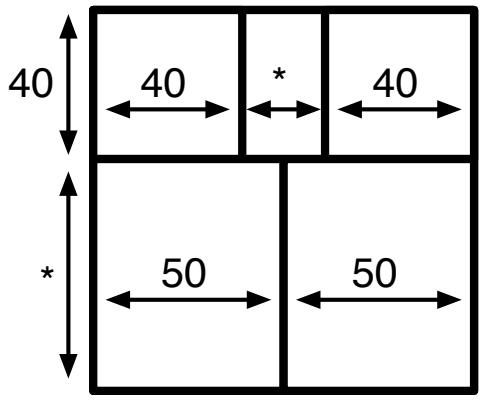


Figure 6-4: The visualisation of a layout expression

Content is assigned to defined cells in the order that the content is encountered.

Structure: Structured We have implemented the structured ZUI layout discussed in Section 4.3.3. This structure provides two tags, namely `structured` and `section`, both of which take a title attribute. First, the div that corresponds to the `structured` tag is encountered, and the title is simply rendered on the canvas. Next, sections are encountered. For every section, its title is rendered together with an automatically counted section number. All the immediate child elements within the `div` are then laid out in a grid underneath the section title.

6.4.7 Themes

In this prototype implementation, themes can be implemented directly in CSS. The core makes sure to load the core and plug-in CSS files first. Then, when themes are loaded they can effectively overwrite already-defined style rules, changing the appearance of both built-in visualisations or plug-in visualisations.

7 Use Case

In Chapter 5 we first detailed MindXpres, our innovative presentation solution. In Chapter 6 we discussed the technical details of our first prototype. During these past few chapters we have claimed to solve many of the issues that are inherent to the slideware tools we all know. The time restrictions implied by this thesis did not allow us to evaluate our solution by means of user trials and such. In this chapter we wish to validate our solution by means of a use case. By walking through a scenario that spans the presentation process, we wish to highlight the major benefits of our solution. As the reader most likely has experience with existing slideware tools, one can compare the process with that of other tools as we walk through the scenario. By doing so, we hope to demonstrate that MindXpres does indeed improve upon existing tools. Additionally, this chapter will give insight on how the tool works from a practical standpoint. By documenting the scenario with small code listings and images, the reader will get a feeling of how MindXpres works, and what it looks like.

7.1 The Scenario

First of all, we need to define the scenario that will be used during the rest of this chapter. This way, we can better illustrate the goals and needs of the user that needs to create a presentation. Instead of using a hypothetical situation, we will use a scenario that the reader can relate to, and that makes sense in the context of this thesis. Therefore, we decided that the goal of this scenario is a presentation on the topic of MindXpres. Moreover, we envision that this presentation on MindXpres will be used during the final defense of

this thesis. Even though this is a fairly important event, we are confident that MindXpres offers the needed functionality, and more. In fact, we hope to illustrate that we can create this presentation with much more ease than if we used other slideware tools (e.g. PowerPoint).

7.2 Creating the Presentation

7.2.1 Creating Structure

Creating a presentation is done via the XML authoring language, so we open our favourite text editor or perhaps even a specialised XML editor. Presentations have a root node called `presentation` and we start with that element in Listing 7.1. The result of these two lines is shown in Figure 7-1.

```
1 <presentation>
2 </presentation>
```

Listing 7.1: An empty presentation

Figure 7-1: An empty presentation

The resulting canvas looks quite empty but this of course expected as we have not defined any content yet. We can now start to build our presentation. As is typical for a project defense, the presentation is often divided in multiple sections. One cannot assume that the audience is familiar with ones work, so often the presentation starts with an introduction to the topic.

Next, the jury is probably interested in the problem statement, that is, what problems does the thesis attempt to tackle. After that, we spend some time on detailing the solution to these problems. Finally, we end with a conclusion. Not only does it summarise our work, it also provides an overview of the concepts we discussed. This overview can be displayed during the time reserved for questions, giving the audience incentive to ask questions.

This type of structure is provided by the structure plug-ins. We must remember that structure plug-ins also guide the visualisation, so it would seem that the structured plug-in provides exactly what we need. It groups content both content-wise and visually, greatly facilitating the work that we would otherwise have to do manually. The structure plug-in provides two tags, `structured` and `section`. Structuring our presentation is easy as shown in Listing 7.2. The result of this structural code is highlighted in Figure 7-2.

```
1 <presentation>
2   <structured>
3     <section title="Introduction"></section>
4     <section title="Problem Statement"></section>
5     <section title="Solution"></section>
6     <section title="Conclusion"></section>
7   </structured>
8 </presentation>
```

Listing 7.2: Applying structure to a presentation

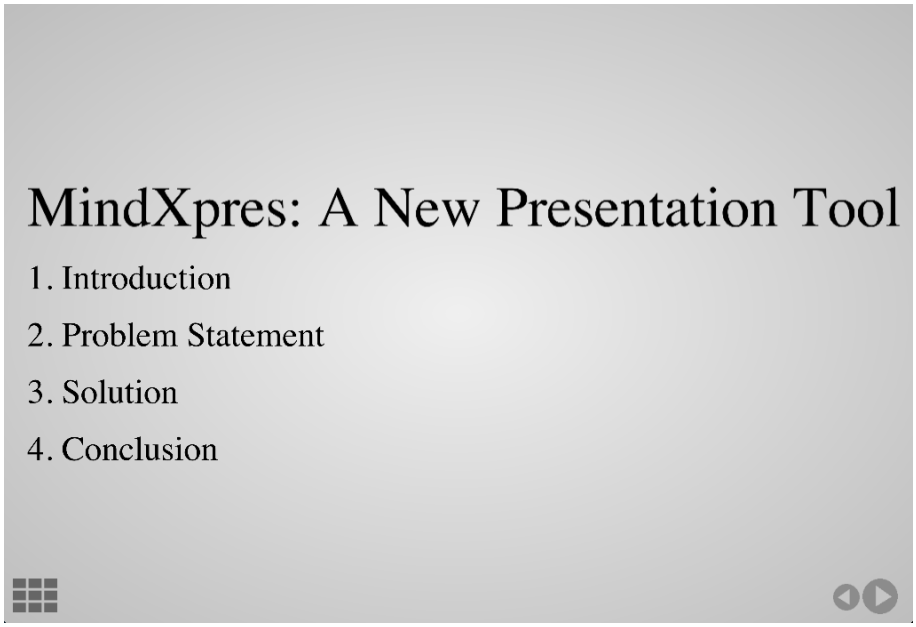


Figure 7-2: A structured presentation

This already looks quite good. We can now start adding real content under the respective sections.

7.2.2 Slides

We have chosen to use the slide container, as it will make the tool seem less radical. By showing some familiar concepts, we want to demonstrate that spatial hypermedia is not as scary as it might sound. We make use of the slide plug-in, which provides the easy to use `slide` tags. We do not know how many slides we will need and just add three slides under each section. The updated code is shown in Listing 7.3. Note that from now on we size the code down to the relevant parts and the three dots (`...`) indicate that irrelevant code is left out. The result of our added slides is shown in Figure 7-3.

```
1  <presentation>
2    <structured>
3      <section title="Introduction">
4        <slide></slide>
5        <slide></slide>
6        <slide></slide>
7      </section>
8      ...
9    </structured>
10 </presentation>
```

Listing 7.3: Adding some slides to the presentation

Figure 7-3: Introducing slides

The slides appear to be plain white rectangles, but this is expected as we have not specified a theme. Therefore, MindXpres falls back on the default settings provided by the slide plug-in. For this fairly important defense, it would be nice if we could give the themes a look that conforms to the VUB guidelines. In order to do so, we can quickly define our own theme. This is done via CSS, simply by overriding the styling rules for a slide. Note that themes are clearly separated from the content, which will allow us to reuse the theme for later presentations.

We start by creating a new folder in the themes folder, which we will call `VUB`. Inside this folder, we create a file called `slide.css`, as we wish to override containers provided by the slide plug-in. In order to style the slides, we will just set the background to the specific image shown in Figure 7-4.

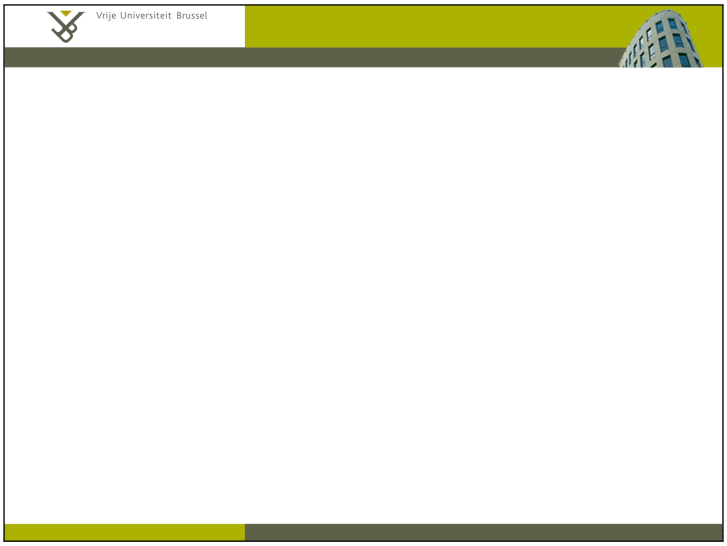


Figure 7-4: The slide background

We create a folder `images` under the `VUB` folder, where we place the background image and add the simple rule shown in Listing 7.4 to the CSS file.

```
1 .slide {
2   background-image: url("images/vub_slide_bg.png");
3   background-size: 100% 100%;
4 }
```

Listing 7.4: A basic slide theme

Next, we modify the `presentation` tag in the XML code to include our new theme: `<presentation theme="VUB">`. After recompiling, the slides are now themed, as shown in Figure 7-5.

Figure 7-5: Applying themes to slides

Note that MindXpres provides all functionality required of a ZUI. If one clicks a slide, the visualisation layer automatically zooms in on it, centring the slide on the screen.

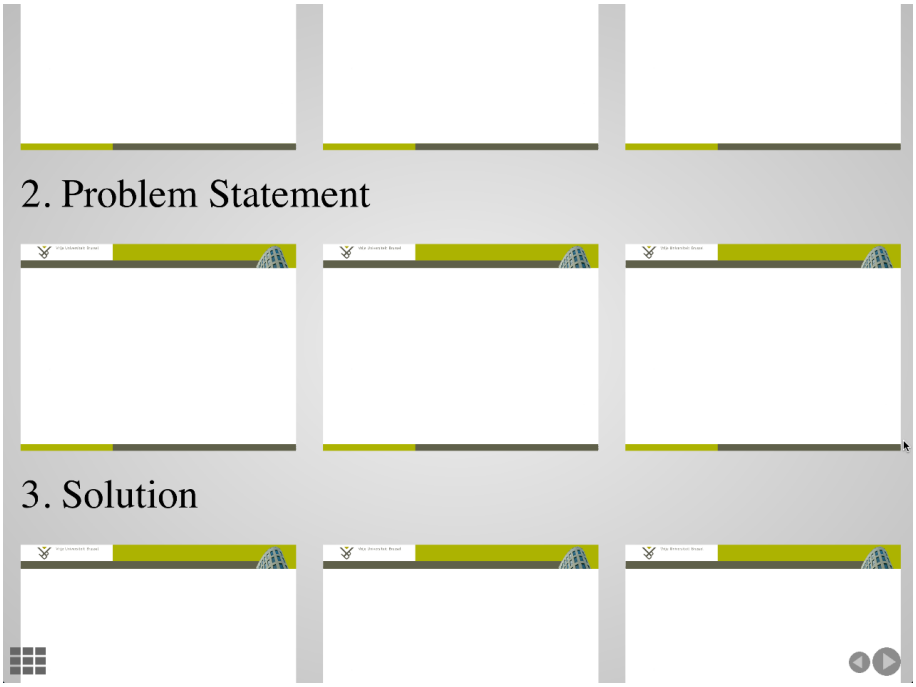


Figure 7-6: Focussing on a single section

Therefore it is best to make sure that a container has the same aspect ratio as a projector's resolution. However, the slide plug-in's default settings make sure that this is the case by default. On top of that, it is also possible to click a section title in order to focus on it. For instance, Figure 7-6 shows the view after the layer focusses on the "Problem Statement" section. This can come in handy to show an overview of the slides that are part of this section. The user can also use the mouse to manually adapt the view. By dragging, the view is panned around, and by using the scroll-wheel the user can adapt the level of zoom. Zooming is done around the mouse cursor, so this can also be used to navigate.

7.2.3 Basic Content

Now that we have a skeleton for our presentation, we can begin filling in the slides. Let us say that we wanted to create a slide for the solution section, where we talk about Vannevar Bush and the memex [5]. First we want to give the slide a title. However, we must keep in mind that our theme has not yet defined any rules for the slide title. We could do without, but then the default settings are used, which would not match our nice background. Listing 7.5 shows the modifications that have to be made to the slide theme.

```
.slide .title {
  font-family: 'Telex', sans-serif;
  color:rgb(95,96,74);
  font-size:46px;
  font-weight:bold;
}
```

Listing 7.5: Styling the slide title

Next, we add some content to the slide. We want to have a bullet list on the left side, with an image on the right side. In order to do so, we use the layout expression `|60|40` to divide the slide in two columns, a larger one for the bullet list and a smaller one for the image. Next, we just insert the bullet list and image tags, which will be applied to the available columns in the order that they are encountered. The result is shown in Figure 7-7.

```
<slide layout="|60|40" title="Vannevar Bush">
  <bulletlist>
    <item>About
      <item>March 11, 1890 - June 28, 1974</item>
      <item>American Engineer</item>
      <item>Founder of Raytheon</item>
    </item>
    <item>Memex
      <item>Human memory works by association</item>
      <item>Microfilm storage</item>
```

```
11        <item>Associative trails</item>
12      </item>
13    </bulletlist>
14    <image source="http://example.com/bush.jpg"/>
15  </slide>
```

Listing 7.6: Adding content to a slide

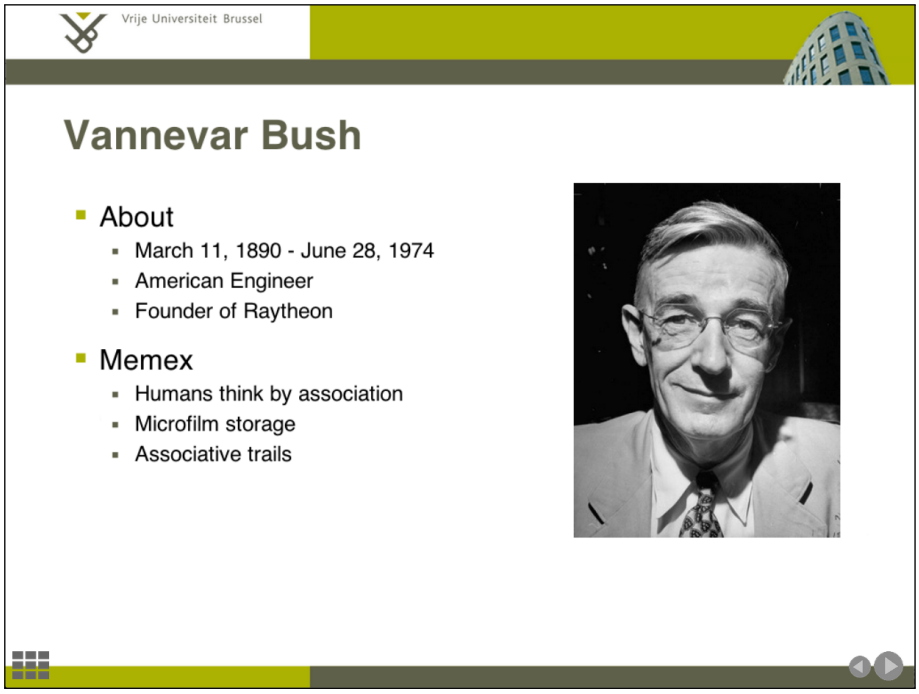


Figure 7-7: A slide with content

The resulting slide looks already quite decent, with little effort from our side. However, Bush's suit seems to make him quite stern. Maybe if we were able to make him look happier, the jury becomes happier which might result in a better grade. We could cut out his face, removing the suit, making him look much friendlier. Luckily, MindXpres allows us to crop the image without actually having to edit the image file. We simply use the `crop` filter that is provided by the image plug-in, as shown in Listing 7.7. We simply tell the plug-in to extract the rectangle that is defined by the points (10%, 5%) and (width - 10%, height - 20%). The result of the `crop` filter is shown in Figure 7-8.

```
1  <image source="http://example.com/bush.jpg">
2    <crop bounds="10%, 5%, -10%, -20%"/>
3  </image>
```

Listing 7.7: Applying a crop filter to the image

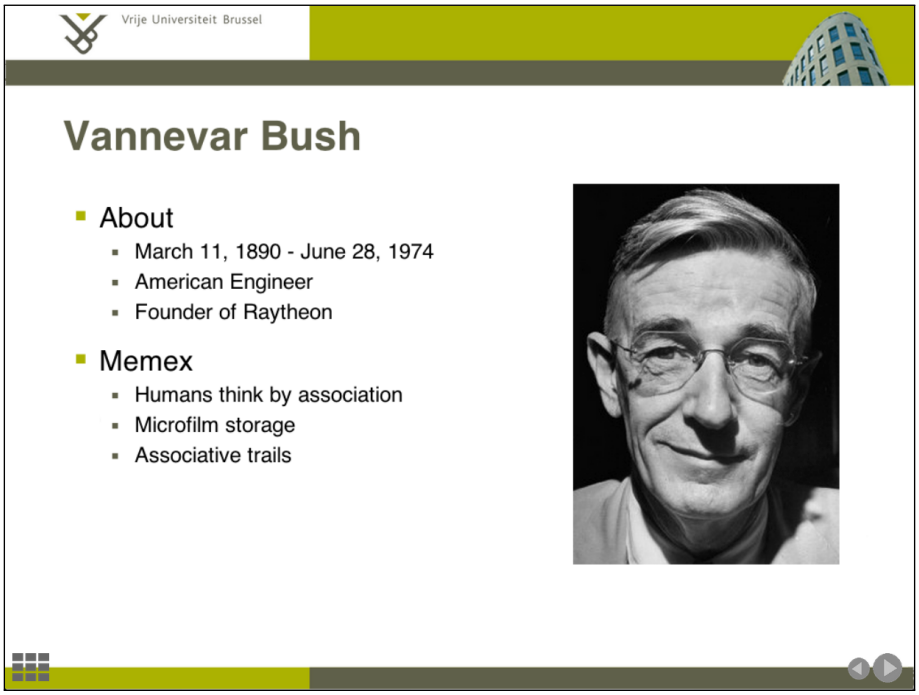


Figure 7-8: The same slide with the crop filter applied

7.2.4 Source Code

Of course, we also want to show some code snippets. Normally we would have to spend a lot of time on formatting and highlighting the code. However, MindXpres makes this a breeze. The exact code, and the result, is shown in Listing 7.8 and Figure 7-9, respectively. This time, we divide the slide in two rows, a smaller one for a bullet list and a larger one for the code.

```
1  <slide layout="_30_70" title="Dependency Injection">
2    <bulletlist>
3      <item>Load dependencies on demand
4        <item>Insert a new node into the DOM tree</item>
5        <item>Call the callback function when the resource has been
             loaded</item>
6      </item>
7    </bulletlist>
8    <code language="javascript">
9      var injectHeadElement = function(type, attributes, callback){
10       var el = document.createElement(type);
11
12       for(attr in attributes){
13         if (!attributes.hasOwnProperty(attr)) {
14           continue;
15         }
16         el.setAttribute(attr, attributes[attr]);
```

```
17        }
18
19        document.getElementsByTagName("head")[0].appendChild(el);
20        el.onload = callback;
21        return true;
22      };
23    </code>
24  </slide>
```

Listing 7.8: Visualising source code

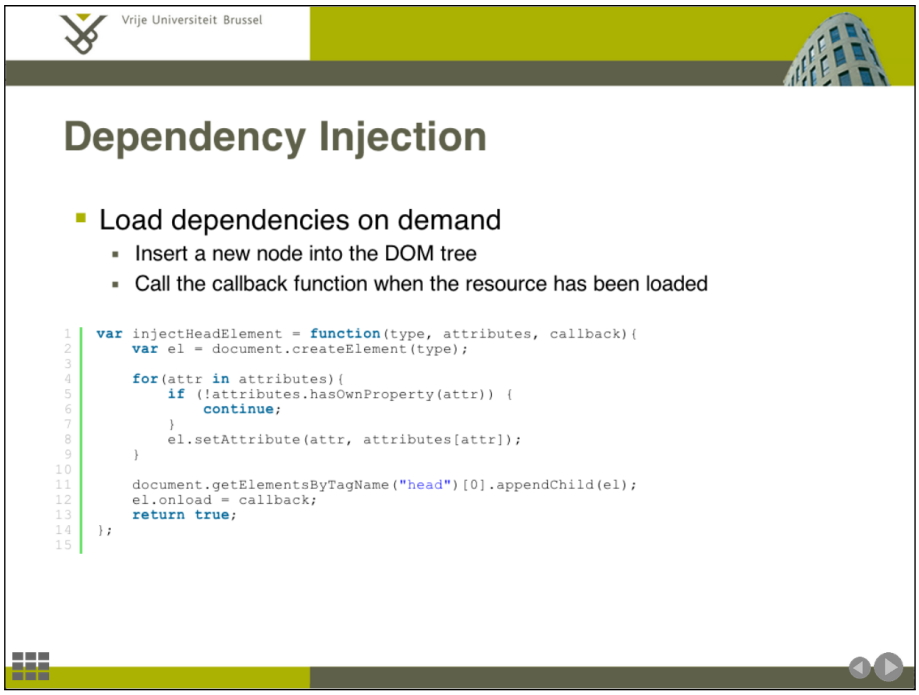


Figure 7-9: Visualising source code

7.2.5 Youtube Plug-in

Now imagine that we wanted to include a Youtube video on a slide. In this scenario, no such plug-ins are available, but that is not per se a problem.Let us demonstrate how easy it is to write a small plug-in.

We start by creating a new folder in the `components` folder, which we will call `youtube`. As defined by convention, we create a `plugin_info.js` file. In this file we need to specify the tags that are offered by the plug-in, as shown in Listing 7.10. In our case, we just want a single tag, namely `youtube`.

```
var youtube_plugin_info = {
  types: ["youtube"],
  path: ""
};
```

Listing 7.9: The plug-in info object

Next, we add another file called `youtube.js`. There, we define the main plug-in object. This object has two methods, `init` and `process`. In our case, we can leave `init` blank, and just use `process` to handle elements that represent youtube containers. Listing 7.10 shows the coded needed for the plug-in. It reads the `source` attribute from the `youtube` tag, extracts the movie identifier from the URL, and then writes the typical HTML code used for embedding youtube videos. Usage of the new plug-in is trivial, as shown in Listing 7.11. The result of this plug-in can be seen in Figure 7-10.

```
var youtube_plugin = new function(){

  this.init = function(){ };

  this.process = function(type, elList){
    elList.each( function(index, el){
      var width = 640;
      var height = 480;
      var source = $(el).data("source");
      var id = source.split("=")[1];

      el.innerHTML = "<iframe width=\"" + width + "\" height=\"" +
           height + "\" src=\"http://www.youtube.com/embed/" + id
          + "\" frameborder=\"0\" allowfullscreen></iframe>";
      $(el).width(width);
      $(el).height(height);
    });
  };
};
```

Listing 7.10: The plug-in info object

```
<slide title="The Memex">
  <youtube source="http://www.youtube.com/watch?v=c539cK58ees" />
</slide>
```

Listing 7.11: Using the new YouTube plug-in

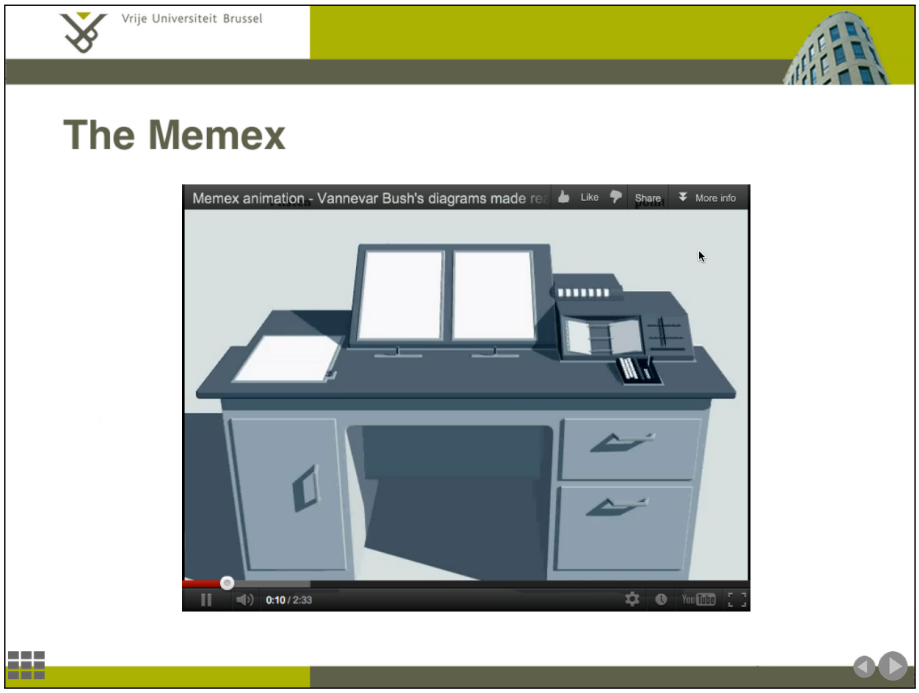


Figure 7-10: Trying out our new YouTube plug-in

7.2.6 The Video Plug-in

Next to the YouTube plug-in, we also want to try out the video plug-in that comes with MindXpres. It is based on the HTML5 video player, but provides some additional features such as subtitles and overlays. More importantly, we can pause a video at a given position, for a chosen amount of time. While the video is paused, subtitles or shapes can be shown to clarify certain things.

We quickly try out the video plug-in with the code shown in Listing 7.12. The result of this code is shown in Figure 7-11, exactly at the 0:35 mark.

```
1 <slide title="The Memex">
2   <video source="resources/memex.mp4">
3     <pause start="0:35" duration="5s">
4       <caption>Notice how each piece of information has a unique
            code that is used for direct access</caption>
5       <rectangle position="52% 44% 20% 10%"/>
6     </pause>
7   </video>
8 </slide>
```

Listing 7.12: Using the video plug-in

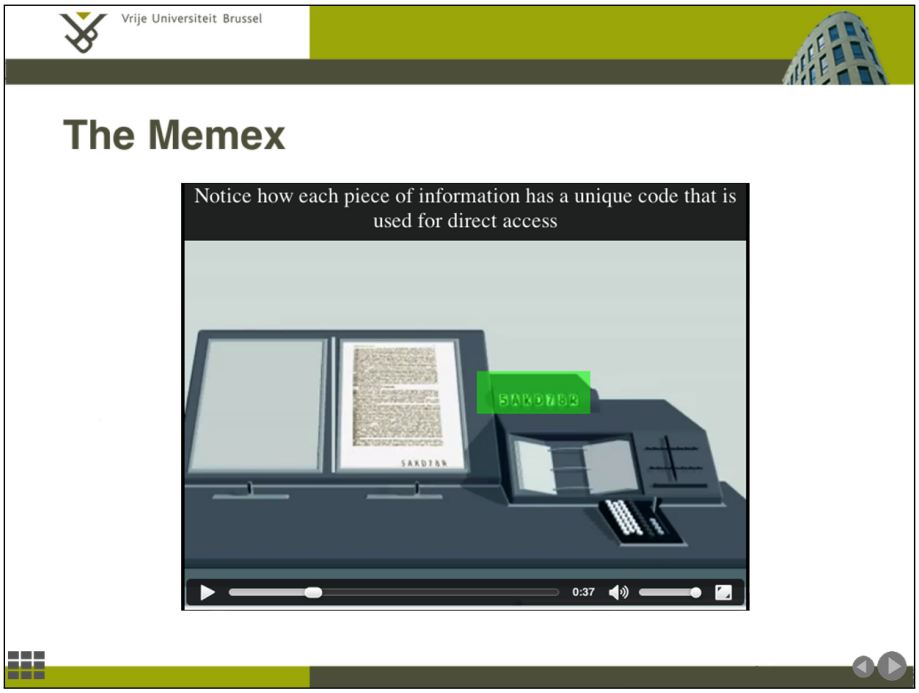


Figure 7-11: Automatic pausing and annotation of a video

7.3 Conclusion

In this chapter, we have demonstrated the usage of MindXpres by means of a simple scenario. As one can see, MindXpres tried to handles as much as possible automatically. The user only needs to supply the bare necessities, which takes a lot of tedious work out of the user's hands, freeing time that can be spent on the content itself. We have also demonstrated the ease of creating a new plug-in. With only 20 lines of basic JavaScript code, we have written a plug-in for an entirely new content type. One must also note that the functionality demonstrated in this use case represents just the beginning. As MindXpres matures, more advanced plug-ins will be made available, making the tool even more impressive.

8 Conclusions and Future Work

In this final chapter, we would like to recapitulate a number of contributions that have been made in this thesis. Additionally, we outline and discuss some future directions for our extensible MindXpres cross-media presentation tool.

8.1 Contributions

The literature study that has been carried out as part of this thesis shows that there is indeed room for improvements in the area of presentation tools. The market of presentation solutions is still dominated by the classic slideware tools that have set the standard since the introduction of digital presentation tools. While there are alternative approaches, they often focus on innovation features for specific niche markets. Therefore, these alternatives solutions normally do not cover the entire presentation process, but rather improve either the authoring, sharing, visualisation or interaction with information. Nevertheless, by analysing and studying these alternative tools, we were able to learn valuable lessons with regards to alternative forms of presentations, and how they relate to the limitations of classic slideware tools. By combining the aspects that we believe to be beneficial for presentation tools with our own ideas for an ideal presentation tool, we came up with an innovative model an architecture for and extensible cross-media presentation software addressing many of the issues of current state of the art slideware. Our MindXpress presentation tool steps away from the classical slide concept and is based on state of the art research from fields such as information management, hypermedia and zoomable user interfaces in order to rethink presentation tools. This investigation of the state of the art in hypermedia

and zoomable user interfaces helped us to avoid some of the pitfalls when designing our MindXpres cross-media presentation tool.

During this thesis, we made the following steps and contributions:

- A literature study has been conducted in order to identify problematic areas in modern presentation tools. Additionally, we have taken a closer look at state of the art as well as alternative presentations solutions in order to use some of the findings in our work.
- Based on the insights gained from the literature study and on the investigation of existing slideware, we broadened our view in order to explore concepts and technologies that might be beneficial in the context of presentation tools, and help to overcome some of the limitations of existing slideware. This research resulted in a list of requirements that we deemed essential for the ideal presentation tool. By making some careful decisions with regards to the use of concepts and technologies, we were able to provide solutions for many of the shortcomings associated with existing slideware tools.
- Based on the requirements for the ideal presentation tool we defined a model and architecture for our proposed solution, which we called MindXpres.
- Finally, we have implemented an extensible functional prototype of MindXpres, which supports many of the innovate features and is going to form the basis for future research on content-driven cross-media presentation solutions.

By implementing MindXpres from scratch, we were able to leave slide-based approaches behind and rethink presentation tools from the ground up. We found that hypermedia is a much better way of storing information, as it much closer to how humans think and learn. By thinking about how the human mind works, we were able to suggest a way of visualising the otherwise hard to display linked information optimally. We combined a zoomable user interface with psychological concepts such as the Gestalt principles, making it easier for viewers to process the information. From the literature study we learned that current slideware tools are often too focussed on aesthetics. Instead of providing content-driven functionality, most existing slideware tools are merely graphical slide editors. Most of the time during the creation of a presentation is spent on the visualisation and not on the actual content. On the other hand, MindXpres places the focus back on content and takes cues from content-driven typesetting systems such as LaTeX. By providing a LaTeX-like language for the authoring of presentations, we let the user handle

the content and MindXpres automatically takes care of the visualisation. Through the use of themes and a plug-in architecture we provide means of visualising information that are above par with what is offered by other presentation tools. The interface to the extensible MindXpres core is open, ensuring that users or developers can easily create plug-ins for new content types they wish to visualise. For the implementation we have chosen HTML5 as the underlying technology, ensuring maximum portability together with a high level of workability. Finally, MindXpres tries to bring interactivity to a whole new level by providing a generic interface for input, paving the way for all sorts of multi-modal input.

8.2 Future Work

Earlier we mentioned that this thesis has to be seen as an initial version of the MindXpres presentation tool. Ever since its conception, it became clear that the limited time available for this thesis would not suffice to bring MindXpres to its full potential. Therefore, we have chosen to spend our time on defining a solid foundation, making sure that future research can build on the extensible core framework. We did not rush the implementation in favour of excessive functionality, but rather ensured that we got the basics right from the beginning. This implies that we spent most time on the core architecture. While we can offer a fair degree of usability, we only had time left to implement a subset of the core extensions that make MindXpres so promising. In this section, we discuss some future work that will bring MindXpres a step closer to its full potential.

8.2.1 XML Authoring Language

The XML authoring language allows the user to focus on content and to spend less time on creating presentations. The vocabulary is almost entirely defined via the plug-ins, and no big changes have to be made regarding the authoring language since it is automatically extended via new plug-ins. However, we have already discussed the usability implications of an interface like this. One cannot expect casual computer users to learn and use a language like this. The very thought alone scares away most of the potential users. Therefore, a usability layer needs to be added in order to get a user base. One way to handle this is to build a graphical editor that generates the language for the user. An intuitive GUI can build a bride between casual computer users and the somewhat advanced functionality that is offered by MindXpres.

In the prototype, we have loosely implemented some concepts from the RSL model. However, initially the plan was to use the currently developed new version of an RSL implementation. Due to some delays in the release of this

new RSL implementation, we decided to come up with our own implementation of some of the RSL concepts in order not to jeopardise the outcome of this thesis. It is clear that the RSL model is not trivial to implement and in the near future we therefore plan to use the new RSL server implementation. This would allow MindXpres to make full use of all the features offered by the RSL model.

Finally, in order to attract users that already have existing assets (e.g. PowerPoint slides), it would make sense to allow the import of other presentation formats. Users could then be offered the option to have their existing work automatically converted to the MindXpres format.

8.2.2 Compiler

As explained in Section 6.2, our prototype compiler validates an XML document with a predefined XML Schema. This is due to the fact that the XML vocabulary is mostly provided by the available plug-ins which means that there is no way of predetermining the allowed tags and attributes. This can be solved by processing the available plug-ins at compile time and by generating the XML Schema based on the installed plug-ins found in the plug-in folders.

At the moment, the compiler is fairly basic. In the prototype, the compiler is mainly responsible for translating the XML authoring language into HTML, but we have foreseen a greater goal for the compiler. First of all, our prototype is only capable of creating one type of presentations. This can be extended to include other types and formats, such as PDF documents for printable presentations. Next, it would be possible to truly process the encountered elements instead of just translating them. For instance, one could build filters that intercept specific elements and modify them in one way or another according to the context. As an example, one could intercept unsupported video formats in order to convert them to a format is supported by the video plug-in. Another example would be to locally gather all the referenced resources and bundle them with the presentation for a portable offline version of the presentation.

Finally, some improvements can be made to the output HTML format. The most important addition would be the use of HTML microdata, which allows the nesting of semantic data within HTML in a computer readable manner. This would not only help with the indexing of presentations by search engines, but it would also allow MindXpres to better query the data via a semantic search interfaces.

8.2.3 Visualisation Layer

While thinking about the architecture of the visualisation layer, we intended to hard code as little as possible and provide most functionality via plug-ins. This was successful for the major part, but there are still some aspects that are built into the core that could be implemented as plug-ins. This is especially true for the HUD (heads-up display) elements such as the overview button and the *next* and *previous* buttons. Ideally, HUD elements should be implemented as plug-ins too, allowing users to configure and create HUDs according to their wishes. For instance, if one wanted to include a mini map of the canvas, this should be implemented as a HUD plug-in.

Because of the lack of time, we were not able to fulfil all the requirements that we have identified in Chapter 4. For instance, there is currently no facility to search a presentation. The aspect of multi-modality has also been slightly neglected, as only mouse and keyboard are supported at the moment. In the near future, additional input mechanisms should be implemented. In order to do so, it could be beneficial to integrate a specialised multi-modal interaction framework such as Mudra [23]. Themes could also be improved. For this prototype users are required to write CSS files to theme the generated HTML. It is clear that this is not optimal, as it means that another language has to be learned on top of the XML authoring language. In the future, one could improve this by creating a more user-friendly way of defining themes.

Finally, it is important to note that the provided plug-ins for a number of content types are fairly basic. The visualisation layer is only restricted by some broad technological boundaries. Anything that works in a browser will also work in MindXpres. This includes interactive elements such as games, 3D visualisations and the integration of physical objects as plug-ins. Even though the MindXpres visualisation layer is written in HTML5, one can still make use of embeddable technologies such as Flash or Java Applets. In other words, creativity is the only big restriction for extending the tool. In future work, we would like to see some innovative plug-ins that truly demonstrate the full capabilities of the MindXpres visualisation layer.

8.2.4 General

In this thesis, the major part of the solution is focused on the authoring and delivery of presentations. However, one can distinguish three major phases in the presentation processes:

- Before the presentation: This includes all the work involved before the presentation is given such as the authoring but also collaboration and sharing.

- During the presentation: This phase occurs when the presentation is given, meaning that is the presenter is "walking" through the presentation with an audience.
- After the presentation: When the presenter is done and the audience is no longer present, this phase comes into effect, where members of the audience can, for example, share their comments.

As we can see, most of the work carried out in this thesis covers the first two stages. In future work, additional research could be done to support the phase after the presentation. For instance, one could upload the presentation and allow forum-like discussions to be made on a per-slide basis. Another example would be the option to create attachments for content (e.g. a slide explaining a snippet of source code could have the full source code attached to the slide for users who view the presentation afterwards). Finally, one could potentially combine physical media such as digital pens with the presentation tool for note taking during presentations, with the option to time synchronise and integrate the annotations with the presentation afterwards.
Even if we were able to identify the many shortcomings of existing slideware solutions, propose innovative features for non-linear cross-media presentations and provide a prototype implementation of a next generation presentation tool, MindXpres has to be seen as the beginning of a broad range of future research on content-driven cross-media presentation tools.

Bibliography

[1] R. A. Bartsch and K. M. Cobern. Effectiveness of PowerPoint Presentations in Lectures. *Computers & Education*, 41(1):77–86, June 2003.

[2] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit Design for Interactive Structured Graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, August 2004.

[3] B. B. Bederson and J. D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of UIST 1994, 7th Annual ACM Symposium on User Interface Software and Technology*, pages 17–26, Marina del Rey, USA, 1994.

[4] P. D. Bra, G.-J. Houben, and H. Wu. AHAM: A Dexter-Based Reference Model for Adaptive Hypermedia. In *Proceedings of Hypertext 1999, 10th ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany, February 1999.

[5] V. Bush. As We May Think. *Atlantic Monthly*, 176(1):101–108, July 1945.

[6] D. Carlisle, P. Ion, and R. Miner. Mathematical Markup Language (MathML) Version 3.0. W3C Recommendation, October 2010.

[7] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999.

[8] C. Corsten. DragonFly: Spatial Navigation for Lecture Videos. In *Extended Abstracts of CHI 2010, 28th International Conference on Human Factors in Computing Systems*, pages 4387–4392, Atlanta, USA, April 2010.

[9] S. DeRose, E. Maler, and J. Ron Daniel. XML Pointer Language (XPointer) Version 1.0. W3C Last Call Working Draft, January 2001.

[10] D. M. Edwards and L. Hardman. Lost in Hyperspace: Cognitive Mapping and Navigation in a Hypertext Environment. pages 90–105, 1999.

[11] D. Englebart. The Augmented Knowledge Workshop. In *Proceedings of HPW 1986, ACM Conference on the History of Personal Workstations*, pages 73–83, Palo Alto, USA, 1986.

[12] E. J. Etemad. Cascading Style Sheets (CSS) Snapshot 2010. W3C Working Group Note, May 2011.

[13] S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation, April 2012.

[14] R. Gaskins. Sample Product Proposal: Presentation Graphics for Overhead Projection, August 1984.

[15] R. Gaskins. PowerPoint at 20: Back to Basics. *Communications of the ACM*, 50(12):15–17, December 2007.

[16] L. Good and B. B. Bederson. Zoomable User Interfaces as a Medium for Slide Show Presentations. *Information Visualization*, 1(1):35–49, 2002.

[17] K. Grønbæk and R. H. Trigg. Design Issues for a Dexter-based Hypermedia System. *Communications of the ACM*, 37(2):40–49, February 1994.

[18] F. G. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2), 1994.

[19] H. Haller and A. Abecker. iMapping: A Zooming User Interface Approach for Personal and Semantic Knowledge Management. In *Hypertext 2010, 21th ACM Conference on Hypertext and Hypermedia*, pages 119–128, Toronto, Canada, June 2010.

[20] L. Hardman, D. C. A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, 37(2):50–62, February 1994.

[21] I. Hickson. HTML5 - A Vocabulary and Associated APIs for HTML and XHTML. W3C Working Draft, 2011 May.

[22] A. Holzinger, M. D. Kickmeier-Rust, and D. Albert. Dynamic Media in Computer Science Education; Content Complexity and Learning Performance: Is Less More? *Educational Technology & Society*, 11(1):279–290, 2008.

[23] L. Hoste, B. Dumas, and B. Signer. Mudra: A Unified Multimodal Interaction Framework. In *Proceedings of ICMI 2011, 13th International Conference on Multimodal Interaction*, Alicante, Spain, November 2011.

[24] F. M. S. III, H. Hsieh, P. Maloor, and J. M. Moore. The Visual Knowledge Builder: A Second Generation Spatial Hypertext. In *Proceedings of Hypertext 2001, 12th ACM Conference on Hypertext and Hypermedia*, pages 113–122, Åarhus, Denmark, 2001.

[25] C. Keep, T. McLaughlin, and R. Parmar. The Electronic Labyrinth, 1993.

[26] M. Koffka. *Principles of Gestalt Psychology*. Harcourt, January 1967.

[27] W. Köhler. *Gestalt Psychology*. Signet, December 1959.

[28] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Professional, July 1994.

[29] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, and A. Y. Ng. Building High-level Features Using Large Scale Unsupervised Learning. In *Proceedings of ICML 2012, 29th International Conference on Machine Learning*, Edinburgh, UK, June 2012.

[30] L. Lichtschlag, T. Karrer, and J. Borchers. Fly: A Tool to Author Planar Presentations. In *Proceedings of CHI 2009, 27th International Conference on Human Factors in Computing*, pages 547–556, Boston, USA, April 2009.

[31] C. C. Marshall and F. M. S. III. Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext. In *Proceedings of Hypertext 1993, 5th ACM Conference on Hypertext*, pages 217–230, Seattle, USA, 1993.

[32] D. D. McCracken and R. J. Wolfe. *User-centered Website Development: A Human-Computer Interaction Approach*. Prentice Hall, May 2003.

[33] G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review*, 63(2):81–97, March 1956.

[34] T. H. Nelson. Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate. In *Proceedings of ACM 1965, 20th ACM National Conference*, pages 84–100, Cleveland, USA, 1965.

[35] T. H. Nelson. *Literary Machines*. Mindful Press, 1982.

[36] T. H. Nelson. The Heart of Connection: Hypermedia Unified by Transclusion. *Communications of the ACM*, 38(8):31–33, August 1995.

[37] T. H. Nelson. *Geeks Bearing Gifts: How the Computer World Got This Way*. Mindful Press, 2009.

[38] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proceedings of ER '93, 12th International Conference on the Entity-Relationship Approach*, pages 390–401, Arlington, USA, December 1993.

[39] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, pages 9–12, Banff, Canada, November 2006.

[40] I. Parker. Absolute PowerPoint: Can a Software Package Edit Our Thoughts? *The New Yorker*, pages 76–87, May 2001.

[41] K. Perlin and D. Fox. Pad: An Alternative Approach to the Computer Interface. In *Proceedings of ACM SIGGRAPH 1993, 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 57–64, Anaheim, USA, 1993.

[42] E. I. Reuss, B. Signer, and M. C. Norrie. PowerPoint Multimedia Presentations in Computer Science Education: What do Users Need? In *Proceedings of USAB 2008, 4th Symposium on Usability & HCI for Education and Work*, pages 281–298, Graz, Austria, November 2008.

[43] C. Scholliers, L. Hoste, B. Signer, and W. D. Meuter. Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of TEI 2011, 5th International Conference on Tangible, Embedded, and Embodied Interaction*, pages 49–56, Funchal, Portugal, January 2011.

[44] F. Shipman, J. M. Moore, P. Maloor, H. Hsieh, and R. Akkapeddi. Semantics Happen: Knowledge Building in Spatial Hypertext. In *Proceedings of Hypertext 2002, 13th ACM Conference on Hypertext and Hypermedia*, pages 25–34, Maryland, USA, 2002.

[45] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. Books on Demand GmbH, May 2008.

[46] B. Signer. What is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse. In *Proceedings of ER 2010, 29th International Conference on Conceptual Modeling*, pages 391–404, Vancouver, Canada, November 2010.

[47] B. Signer and M. C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In *Proceedings of ER 2007, 26th International Conference on Conceptual Modeling*, pages 359–374, Auckland, New Zealand, November 2007.

[48] B. Signer and M. C. Norrie. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proceedings of TEI 2007, First International Conference on Tangible and Embedded Interaction*, pages 57–64, Baton Rouge, USA, February 2007.

[49] B. Signer and M. C. Norrie. A Framework for Developing Pervasive Cross-Media Applications based on Physical Hypermedia and Active Components. In *Proceedings of ICPCA 2008, 3rd International Conference on Pervasive Computing and Applications*, pages 564–569, Alexandria, Egypt, October 2008.

[50] B. Signer and M. C. Norrie. Interactive Paper: Past, Present and Future. In *Proceedings of PaperComp 2010, 1st International Workshop on Paper Computing*, Copenhagen, Denmark, September 2010.

[51] B. Signer and M. C. Norrie. A Model and Architecture for Open Cross-Media Annotation and Link Services. *Information Systems*, 36(6):538–550, May 2011.

[52] J. E. Simpson. *XPath and XPointer*. O'Reilly & Associates, July 2002.

[53] E. R. Tufte. *The Cognitive Style of PowerPoint: Pitching Out Corrupts Within*. Graphics Press, July 2003.

[54] E. R. Tufte. *Beautiful Evidence*. Graphics Press, July 2006.

[55] M. Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychological Research*, 4(1):301–350, 1923.