Vrije Universiteit Brussel

Faculteit Wetenschappen
Departement Informatica
en Toegepaste Informatica

Content Migration and Layout for the MindXpres Presentation Tool

Proefschrift ingediend met het oog op het behalen
van de graad van Master in de Toegepaste Informatica

Joris Vandermeersch

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

Augustus 2015

Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science
and Applied Computer Science

Content Migration and Layout for the MindXpres Presentation Tool

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

Joris Vandermeersch

Promotor: Prof. Dr. Beat Signer
Advisor: Reinout Roels

August 2015

SAMENVATTING

Klassieke diavoorstellingen worden wereldwijd gebruikt om kennis over te brengen en te delen. Microsoft PowerPoint is het populairste en meest bekende programma in dit gebied. Spijtig genoeg heeft deze software weinig evolutie gekend sinds ze het licht zag in de jaren 1980, ondanks een ongelooflijke stortvloed aan vernieuwingen in bijna elk ander aspect van informatie- en communicatietechnologie. Het basisconcept van digitale presentaties is steeds hetzelfde gebleven, gebaseerd op de oorspronkelijke fysieke overhead- en diaprojecties. De beperkingen in grootte zorgen vaak voor een minder dan ideale weergave van informatie, waardoor het moeilijk wordt voor de presentator om alles goed te kunnen uitleggen, alsook voor het publiek om alles te begrijpen.

Presentatoren ondervinden vaak, tot zelfs gewoonlijk, problemen met het schikken van hun inhoud binnen dit formaat op een manier die duidelijk, overzichtelijk en esthetisch verantwoord is. Het grootste deel van de tijd nodig om een presentatie te maken wordt verspild aan het zoeken van een goede layout voor de voorziene informatie, vaak dan nog met weinig positief resultaat. Professionele designers werken jarenlang aan templates voor dit soort presentatiesoftware, in een poging om een alles passende oplossing te voorzien voor onderdelen die — weinig verrassend — meestal niet voldoen aan de regels die erop toegepast worden.

MindXpres is een programma dat een paradigmaverschuiving in het bewerken en geven van presentaties met zich meebrengt. Het voorziet een uitbreidbaar platform dat een presentator toelaat zich te focussen op de inhoud van zijn presentatie, terwijl MindXpres de visualisatie voor zijn rekening neemt. Het geheel bestaat bijna volledig uit plug-ins die verschillende soorten informatie verwerken en visualiseren, en maakt het mogelijk om nieuwe plug-ins toe te voegen om nieuwe functionaliteit beschikbaar te maken wanneer nodig. De verbeelding van de plug-in ontwikkelaar is de enige grens aan dit systeem.

In deze thesis stellen we een programma en aanpak voor waarmee bestaande PowerPoint presentaties kunnen omgezet worden in MindXpres presentaties, met de bedoeling om PowerPoint gebruikers te overtuigen om MindXpres te gaan gebruiken door hen de mogelijkheden van MindXpres

te tonen met hun eigen presentatie-inhoud. Als bijkomend — maar niet minder belangrijk — doel proberen we het bestaande layout systeem van MindXpres, gebaseerd op templates, te vervangen door een layout algoritme dat een ideale layout genereert gebaseerd op de onderdelen van een presentatie.

ABSTRACT

Classic slide-based presentations are used worldwide to share and transfer knowledge. Microsoft PowerPoint is the most popular and well-known software package in this area. Unfortunately, little evolution has taken place since its inauguration back in the 1980s, despite an incredible amount of innovations in almost every other area of software and computer technology. The main concept of digital presentations has always remained the same, based on the original physical presentations using overhead or slide projectors. The limitations in size often force a less than optimal display of information, making it difficult for the presenter to explain and for the audience to understand the information that is being presented.

Presenters often, if not usually, struggle to put their content into this format in a way that is clear to understand, aesthetically pleasing and well-structured. Most of the time spent creating a presentation is wasted on finding a proper layout for the content provided, frequently with suboptimal results. Professional designers spend years designing templates for these presentation tools, trying to automatically provide a one-size-fits-all solution for content that unsurprisingly mostly does not conform to the restrictions imposed upon it.

MindXpres is a presentation tool that brings a shift of paradigms in authoring and delivering presentations. It provides an extensible platform that allows a presenter to focus on the content of their presentation, while MindXpres takes care of the visualisation. It consists almost entirely of plug-ins to process and visualise various content types, and allows the addition of new plug-ins to introduce new functionality as needed. The only limit in this system is the plug-in developer's imagination.

In this thesis, we propose a tool and an approach to convert existing PowerPoint presentations into MindXpres presentations, in the hopes of convincing PowerPoint users to switch to MindXpres by showing them the possibilities of MindXpres using their own content. As a second goal, we try to replace the default template-based layout system of MindXpres with a layout engine that generates an ideal layout based on the content of a presentation.

ACKNOWLEDGEMENTS

"Simplicity is a great virtue,
but it requires hard work to achieve it and education to appreciate it.
And to make matters worse: complexity sells better."

— *Edsger W. Dijkstra*

CONTENTS

1. INTRODUCTION

For over 25 years, Microsoft PowerPoint has been the market leader in digital prsentations. Admittedly, it was a revolutionary software package when it was first introduced, and its ease-of-use combined with its supreme graphical capabilities — at least compared to other software in the same era – quickly made it one of the most popular software packages in history. 25 years later, Microsoft PowerPoint can claim over 90% market share in presentation software, and on average 30 million PowerPoint presentations are created every day (Parker, 2001, Drucker et al., 2006, Bajaj, 2013).

In this time, Microsoft PowerPoint has gotten many new features, and certainly improved and grew with every new version, but it never really changed its core approach. It started out mimicking the then-popular and widespread use of dia and overhead projection slides, which was at the time a good way to convince people of its purpose, allowing them to feel comfortable with a familiar format instead of alienating potential customers with a new and potentially confusing interface.

However, this interface is quite restricting, and in recent years different approaches have seen the light of day. The zoomable user interface of Prezi is probably the most well-known, but apart from abandoning the traditional slide format it does little to improve or extend the concept of presenting information to an audience.

This is where MindXpres comes in. Its extensible plug-in system allows anyone with some knowledge of programming to create new functionality to use in presentations. Examples are interactivity with the audiencer through various means, controlling the presentation from another device — or several! — and (re)modelling data while presenting it, based on feedback from the audience.

While this is obviously a big improvement on the traditional presentation model of Microsoft PowerPoint and the likes, it remains hard to convince the general public of its merits. People are generally afraid of change, and it is important to make the transition as smooth as possible. On top of that, people are often worried that the work they did in the past may be lost — or worse, irrelevant — after switching to something new. This alone may be a huge factor in deciding wether or not to start using new software, or to stick

with what they know.

That is where the subject of this thesis comes in. We aim to provide a way for people to convert their existing PowerPoint presentations into MindXpres presentations, allowing them to take their previous work with them in their switch to MindXpres. This way, we lower the treshold for them to make the decision to start using MindXpres as their presentation software of choice. Once all their existing PowerPoint content is available, usable and editable in MindXpres, it should be obvious to anyone why MindXpres is the better option for their presentations.

Another common problem with PowerPoint presentations is the way they look. This is not necessarily the fault of the software; most people just are not trained in graphical design, and as such they know very little about proper layout, color choices, or slide content limits. Everyone has probably encountered slides with full paragraphs of text, too small to read and / or too much to process in the short time the slide is visible — (too) many people have made those slides themselves.

When we say this is not the fault of the software, that is mostly true, as the creators of these slides obviously made a conscious choice to make their content appear like that. It could be said however that Microsoft PowerPoint and other presentation tools are guilty through inaction. We believe it is possible to have software either warn its users against these choices and practices, or — even better — have the software fix these problems automatically.

One of the primary purposes of MindXpres is to provide automated layout, much like LaTeX does, ensuring that the content creator only has to worry about the actual content, while the software takes care of layout. In practice, both LaTeX and MindXpres currently use template-based layouts, where the contents' position is predefined in the template and not related to or based on its size, shape or nature. In the end, everyone who has ever used LaTeX knows that sooner or later you will struggle to get a certain image incorporated in the text correctly, ending up doing the layout yourself anyway, because the predefined template just doesn't work properly for your specific content.

Our goal is to eradicate those situations. Automated layout should dynamically adjust to any content it is given, no matter the size or aspect ratio. This may seem hard, if you consider the limits of slides and the fact that you can only fit so much content on them before they are full. This is where another important aspect of MindXpres comes into play: we are not necessarily bound to the limits of slides. If we don't have to consider the boundaries of traditional slides, we can fit content together in an aesthetically pleasing way much easier, without having to scale anything.

As such, the second part of this thesis focuses on implementing true au-

tomated layout in MindXpres. Again with the goal to convince Microsoft PowerPoint users to switch, showing that their presentations actually could look better in MindXpres, but at the same time we also provide new functionality to other MindXpres users.

We believe this functionality will improve the aesthetic aspect as well as the effectiveness of presentations. If content is not scaled down to fit the articifial confines of a slide, but can instead be shown and studied in detail, this should clearly increase the flow of information towards the audience. Providing an overview of the information in a presentation becomes easier and more effective too: where traditional presentations rely on a boring table of contents, in which the presenter announces the subjects they'll be talking about one by one while the audience forgets the first thing in the list by the time they get to the last, MindXpres allows the presenter to just show all of the presentation's content at once just by zooming out. Here automated layout can help as well: content can be arranged in such a way that an overview effectively hilights the important subjects, different parts or keywords of a presentation.

Last but not least, we hope this functionality improves the experience of creating a presentation. Everyone who ever created a presentation knows, and research has shown (Lok and Feiner, 2001) that often more time is spent on creating and fine-tuning the layout than actually putting in the content. Most presenters however have not had any significant training in creating effective layout, which means this time is often wasted on a layout that ends up not actually benefiting the presentation as a whole. We want to eliminate this problem by taking control over the layout away from the presenter and instead providing them with a programatically generated layout that presents the information provided by the presenter in the best, clearest way possible.

2. SLIDEWARE AND THE IMPORTANCE OF LAYOUT

Computers, software and digital content are everywhere. Everything we use nowadays is somehow related to computers and electronics, and if it isn't, it probably will be soon. This may be a bit of hyperbole, but there's a core of truth in it. If you think about it, more and more things have become and are becoming some kind of computer. Coffee machines used to be simple machines that heated water and let it drip over coffee grounds; now there are coffee machines that are connected to the internet, and can be turned on remotely from your smartphone. That smartphone itself is an incredible evolution as well: just 20 years ago, phones were analog devices, and you could use them to call people and nothing more. Today, our phone does a lot more than that, so much more that calling has actually become a minor feature to most people.

Content is going the same way. Photos used to be on a special film, and could be 'developed' onto special paper through a proces involving a dark room and several chemicals. Movies existed on a projection film, newspapers were actually made of paper and music was available on vinyl disks with grooves that matched the sound waves. All of this content has been digitized since. This means of course that you can see or hear it using a computer, like you would've seen it without a computer before, but on top of that it means the content can be much more dynamic. You can link it to other content, you can make it respond to your actions, you can discuss it with people around the world. Digitized content allows for interactivity, so that the audience is no longer a passive onlooker but an active participant.

It is no surprise, then, that slideshow presentations have evolved from the original dia's or overhead projection slides into a digital form as well. Except, until recently the evolution stopped there. Slideshows did not become interactive, and the audience remained passive onlookers watching a series of images projected on a screen or a wall. The presenter told a story, and the audience listened. Often during or at the end of the presentation there would be a chance to ask questions, but those questions could only be answered vocally by the presenter. If the question needed any visual explanation, the slideshow would not be able to help. We had digital slides, but the difference with the physical slides was neglectable.

In our eyes, the culprit for this is Microsoft's PowerPoint. This software package took the world by storm, making it possible for everyone with a computer to make digital slideshows, which was impressive at the time. However, Microsoft PowerPoint never really evolved beyond that. It did add features that fit within the slideware concept, but never went beyond that comfort zone. Since it was — and still is! — the dominant player in the world of slideware with over 90% market share, this apathy towards change firmly rooted slideware in the concepts of the past. Luckily, a few years ago some people realized this and decided to take matters into their own hands. They stepped away from the classic slide format, allowing for any kind of layout, combined with zoomable interfaces and other methods of displaying data. One such alternative is MindXpres, created in the WISE lab at the VUB.

MindXpres is based on a plug-in architecture. Plug-ins can do anything from arranging data in a certain way to letting the audience control the slideshow. Virtually anything is possible if you only implement it, and implementing it is fairly simple if you know a bit about web development as the whole thing is written in HTML5. Other software packages have plug-ins too of course, but they have a limited set of functionality available to them, they're not as easy to implement, and most importantly: they're bound by the same slide format used since overhead projections.

However, even with the new alternatives, Microsoft PowerPoint remains the most-used slideshow software. People keep using it because it's familiar, they've used it hundreds of times before and as such all their existing work is viewable only through PowerPoint. Switching to a new software package is hard. This thesis aims to make the transition easier, by providing a way to convert existing PowerPoint presentations into MindXpres. On top of that, we try to find a way to immediately release the transferred content from the confines of classic slides, by instead automatically figuring out the best possible layout for the content we extracted from the original PowerPoint file.

2.1 Terminology

The words *slideshow* and *presentation* are often used interchangeably throughout this report, although they do not quite cover the same meaning. By *slideshow* we mean a presentation consisting of a set of slides, the kind Microsoft PowerPoint and many other presentation software provide us with. *Presentation* then refers to the wider concept of material intended to be viewed and manipulated by people in order to convey information, usually but not necessarily from one or several presenter(s) to an audience.

The term *layout* refers to both the process of determining the position and size of each visual object that is to be displayed in a presentation, and the result of that process.

Slideware is a contraction of the words 'slideshow' and 'software', referring to software packages used to create slideshow presentations.

2.2 Problem statement

According to several sources (Parker, 2001, Drucker et al., 2006, Bajaj, 2013), over 30 million PowerPoint presentations are being made every day. That is an enormous amount. Creating a PowerPoint presentation is easy; creating a good PowerPoint presentation, however, is not. Slides have a fixed size, and you can only fit so much information on one slide before the effectiveness of transferring that information to one's audience starts deteriorating. Over the years, many people have created written and unwritten guidelines to creating effective slideshows, specifying how much text and how many images should fit on one slide. Over those years, many people have failed to follow those guidelines. But whether you choose to follow the guidelines or not, one thing remains true: people who create slideshow presentations spend most of their time not on the *content* of their presentation, but on the *layout* (Lok and Feiner, 2001).

The layout of a presentation can have a significant impact on how well it communicates information to and obtains information from those who interact with it. The vast majority of layouts created today are done "by hand": a human graphic designer or "layout expert" makes most, if not all, of the decisions about the position and size of the objects to be presented. Designers typically spend years learning how to create effective layouts, and may take hours or days to create even a single screen of a presentation. Designing presentations by hand is too expensive and too slow to address situations in which time-critical information must be communicated.

Since layout is such a hard skill to master, we propose to automate this task, letting the presenter focus on the content of the presentation and providing a proper layout fit for the content provided through software.

2.2.1 Real-life slideware problems

It may seem like an overstatement to emphasize the significance of layout and formatting in presentations. One could assume these issues are irrelevant, or that only inexperienced presenters would make these mistakes. The real-life example of the space shuttle Columbia illustrates that this is not always the case. Leading up to the tragic incident in which the shuttle burned up

during re-entry after spending 2 weeks in orbit, Boeing Corporation engineers delivered three reports to NASA totalling 28 PowerPoint slides, to help them assess the damage caused by a piece of debris hitting the wing of the shuttle during launch, and the threat this damage might have posed. As Edward Tufte beautifully describes in his article "PowerPoint Does Rocket Science" (Tufte, 2005), the reports existed only in those slides, and the slides were woefully inadequate for the task at hand. Although Tufte likes to suggest this proves that PowerPoint is an inherently bad tool, what it really proves is that PowerPoint makes it easy to create bad presentations, and a tool that either discourages this manner of presenting information or makes it altogether impossible would be a great improvement.

3. RELATED WORK

This chapter's content is largely based on "MindXpres: An Extensible Content-driven Cross-Media Presentation Platform" (Roels and Signer, 2014).

3.1 Background

The importance of digital presentations in this day and age cannot be understated. Millions of presentations are created every day, supporting the oral transfer of knowledge and playing an important role in educational settings. Their origins as tools for creating physical media such as photographic slides or transparencies for overhead projectors are still reflected in the underlying concepts and principles of slide-based presentation tools. The rectangular boundaries of a slide, and the linear navigation between slides, are still restrictions we face today in digital presentations. Tufte argues that these concepts of slideware have a negative impact on the effectiveness of knowledge transfer (Tufte, 2003). While the presenter is compelled to squeeze complex ideas into a linear sequence of slides, those ideas are rarely sequential by nature, resulting in a loss of relations, overview and details. An initial approach to address these issues might involve creating minimalistic presentations or introducing some structure via a table of contents. Sadly, when complex knowledge or other pieces of rich information need to be presented as is (Farkas, 2006) — as in the domain of learning — this does not work.

One of the main issues with traditional slideware presentations is their monolithic nature, especially when content is spread over many self-contained presentation files. "Reusing" previous work involves either switching between files while giving a presentation or duplicating some slides in the new presentation. It should be noted that this issue is not limited to the reuse of single slides: there is an ever increasing wealth of resources available for reuse, spread over a wide spectrum of distribution channels and formats. The possibility to include content by reference or transclusion (Nelson, 1995) may contribute in crossing the boundaries between different types of media and prove beneficial in the context of modern cross-media presentation tools.

The difference in functionality between the authoring of content and its visualisation is striking as well. The primary editors consist of mostly tool-

bars and buttons used for selecting and specifying the way content should be visualised, while support for authoring the content itself is not quite as extensive. Modern slideware has grown to include basic multimedia types such as videos, but most content is still rather static. It is, for example, not possible during a presentation to easily switch from a bar chart to a pie chart data visualisation, or to dynamically change some values in the represented data and immediately see the effect in the graph, which could be beneficial for knowledge transfer (Holzinger et al., 2008). The audience could also be more actively involved in the presentation, through audience response and classroom connectivity systems providing multi-device interfaces allowing to share knowledge and results during, as well as after, a presentation. The evolution of presentations is reminiscent of the Web2.0 movements, where users have switched roles from purely consuming content to contributing as well, content has become more dynamic and interactive, and service-oriented architectures ("The Cloud") have ensured decentralisation of content.

In order to move a step towards the next generation of cross-media presentation tools, it is essential to allow the rapid prototyping and evaluation of new concepts for the representation, visualisation and interaction with content.

Before discussing the requirements for a new generation of presentation tools, we briefly introduce existing slideware solutions. Afterwards, we describe the architecture of MindXpres, its extensible nature and its plug-in mechanism. The HTML5-based implementation of MindXpres is then discussed through demonstration of several use cases and MindXpres plug-ins.

A specific issue with slideware we'd like to focus on in this thesis, is the trouble with layout in presentations. It can be hard to display the content you want in a way that's clear, informative and nice to look at. The vast majority of layouts created today is mostly done by hand: a human graphic designer or "layout expert" makes most, if not all, of the decisions about the position and size of the objects to be presented (Lok and Feiner, 2001). Most software offers some templates, allowing you to drop pictures and text into predefined slots and places on a slide, but then those templates have been defined by someone else too. Computer-generated layout is rare and usually not quite up to the task.

MindXpres is among the software packages offering templates, in that layout is handled by whichever plug-in you choose, but so far no plug-ins have defined dynamic layout algorithms, rather sticking to predefined ways to put text and pictures on slides. But as MindXpres does not constrain us to the limits of slides, this should be seen as an opportunity to offer dynamic layout as well. After all, if we're not limited to a certain area within which our content should fit, it should be much easier to put content next to each

other in a way that makes sense.

3.2 Existing solutions

Since digital slideware was first introduced, their influence, advantages and disadvantages have been studied extensively. There have been studies acknowledging the benefits of slideware as a teaching asset (Holzinger et al., 2008), while others have been less positive. Tufte (2003) heavily criticises slideware for its infatuation with outdated concepts. He discusses the many ramifications of dimensional and structural limitations as well as linear navigation, and points out the discrepancy with how the human mind works. Amongst Tufte's conclusions, and also confirmed by Adams (Adams, 2006), is the suggestion that slide-based presentations are not appropriate for every kind of knowledge transfer and especially not in a scientific context. Recent work shows the importance towards the learning process of integrating content into the bigger picture, both structurally and visually (Gross and Harmon, 2009), which is affected by the navigation and visualisation.

Several approaches have been proposed to offer non-linear navigation. CounterPoint (Good and Bederson, 2002), Fly (Lichtschlag et al., 2009) and Prezi, provide Zoomable User Interfaces (ZUIs) which offer virtually unlimited space. Microsoft has experimented with this concept as well in pptPlex. Other approaches to escape the confines of the slide have been noticed, like MultiPresenter (Lanir et al., 2008) or tiling slideshows (Chen et al., 2006). PaperPoint (Signer and Norrie, 2007a) and Palette (Nelson et al., 1999) additionally facilitate the non-linear navigation of digital presentations consisting of slide selection through augmented paper-based interfaces. Lastly, a category of authoring tools exists which use hypermedia to implement varying paths through a set of slides. NextSlidePlease (Spicer et al., 2012) enables users to define a weighted graph of slides, and tries to suggest navigational paths based on the link weights and the remaining presentation time. Microsoft cultivates this idea in their HyperSlides (Edge et al., 2013) project. Garcia (Garcia, 2004) has additionally explored the potential of Microsoft PowerPoint as an authoring tool for hypermedia-based presentations.

Microsoft PowerPoint was officially released in 1990, with Windows 3.0 (Austin, 2009). It had originally been developed as Presenter, but trademark issues caused a name change early on. It was also originally build for the Macintosh, which may seem surprising nowadays but was actually common practice back then since the Macintosh was widely regarded as a better development environment, more mature, more stable and capable of far better performance and visualisations. Some may argue this still rings true today.

Since then, it has grown to be the world's most popular slide show presentation program, alledgedly having been installed on over 1 billion computers worldwide, and being used on average 350 times *per second* (Parks, 2012). In 2012, it had a market share of 95%, leaving the other 5% to be shared by alternatives such as Apple's Keynote, Prezi, SlideRocket and others. While this number is declining, it may not be going as fast as many people think. As most readers of this thesis have heard before, over 30 million PowerPoint presentations are created every day, for all kinds of purposes, with good and bad results both presentation-wise and goal-wise.

To reuse content in existing presentation tools, that content needs to be duplicated, which results in a multitude of redundant copies that need to be kept consistent with each other: if one copy is changed, all the others must be changed in the same way to prevent inconsistencies and mistakes. While some attempts have been made to solve this problem, there is still a long way to go. When looking for document formats designed to server more general educational purposes, we find formats such as the Learning Material Markup Language (LMML) (Sü and Freitag, 2002), the Connexions Markup Language (CNXML) and the eLesson Markup Language (eLML) (Fisler and Bleisch, 2006). All of these formats share their focus on the reuse of content, but all of them attempt this at a relatively high granularity level. Content can be organised in lessons or modules, and users are encouraged to use these, as a whole, in their teaching. When we investigated the formats more closely, we observed that outgoing links to external content were supported, but transclusion[1] was not. In relation to presentations, Microsoft's Slide Libraries exist as central repositories that store slides to enable slide sharing and reuse within an organisation. The dependency on SharePoint might represent a hurdle for some users, as not everyone has the ability and opportunity to set up such a server. A more significant issue is the fact that slides still need to be searched and manually copied into presentations. Keeping slides in the repository and in other presentations is the responsibility of the authors of those slides and those presentations, as no automatic update system is provided. SlideRocket and SlideShare are both similar tools showing intentions and providing functionality for content reuse. The SliDL (Canós-Cerdá et al., 2010) research framework works much like Microsoft's Slide Libraries, in that it allows for storage and tagging of slides in a database for reuse, but also in that it shares the same shortcomings. The ALOCOM (Verbert et al., 2008) framework aimed at flexible content reuse is built upon a content ontology and a (de)composition framework for legacy documents including PowerPoint documents, Wikipedia pages and SCORM

[1] The inclusion of content via references

content packages. However, ALOCOM may be too rigid for evolving presentation formats, and it currently only supports the authoring phase, although the tool does succeed in decomposing legacy documents as advertised.

Aside from the similarities in the Web's and presentation environments' evolution, some of the problems mentioned in this section can find their solutions in the context of the Web. It should not come as a surprise then, that web technologies are being used more often recently in the realisation of presentation solutions. The Simple Standards-based Slide Show System (S5)[2] is an XHTML-based slideshow file format that enforces the standard slideware model. The W3C's Slidy (Raggett, 2006) initiative offers another presentation format based on the classical slideware model. Both of these formats have some valuable properties. They encourage a clean separation of content and visualisation through the use of CSS themes. The design is resolution independent, and the layout and font size adjust to the available screen real estate. Last but not least, some more recent HTML5-based presentation solutions such as impress.js, deck.js, Shower or reveal.js. Cross-device support is one of the most important advantages to leverage when using a well-known open standard such as HTML. However, as all of these solutions display some restrictions in terms of visualisation, navigation, and cross-media support, they are unfortunately too limited for our needs.

The tools and projects discussed in this section mostly focus on distinguishing novel ideas for presentations. Nevertheless, the different concepts introduced in these tools don't offer interoperability between them. One project may focus on the authoring, another one fixates on novel content types and a third solution supplies radically new navigation mechanisms. Slideware tools may often allow third-party extensions but the API exposed to plug-in developers is usually limited by the software's underlying model. As an illustration, PowerPoint supports interaction from plug-ins with the presentation model, but the model dictates that a presentation consists of a sequence of slides. Many existing web-based presentation formats share this flaw. Because of this, we see a need for an open presentation platform such as MindXpres to support innovation by contributing the necessary modularity and interoperability (Bush and Mott, 2009).

It is perhaps surprising that, to our knowledge, currently no tools exist to calculate dynamic layouts of content in slideware. Existing solutions include template systems, sometimes very fine-grained like LaTeX allowing you to define templates for every single layout choice, usually more coarse like Microsoft PowerPoint or Apple Keynote using Master Slides to define different layouts on a per-slide basis, and always with the option of letting the

[2] http://meyerweb.com/eric/tools/s5/

user customise the layout by hand, literally manually moving the content to the exact place where we want it, unhindered by style guides, good practices or common sense. This has resulted in mindboggling layout choices involving enormous amounts of tiny text crammed onto one slide, or pictures strewn across a slide overwhelming the audience with too much information at once.

3.3 New solutions

Here we propose a set of requirements to establish a wide range of presentation styles and visualisations. This set has been compiled based on a review of the more recent presentation solutions discussed in the previous section.

Non-linear Navigation As we mentioned before, traversing slides in a linear fashion is a remnant of the way early photographic slides worked. Over the years, people have grown used to this form of navigation despite the inconveniences. If the presenter unexpectedly needs to show anything other than the next or previous slide (e.g. to answer a question from the audience), they either need a considerable amount of time to scroll forwards or backwards, or they have to switch to the slide sorter view, to find the desired slide. Also troubling is the lack of any functionality allowing a single slide to be included multiple times throughout the presentation without duplicating the slide in question, meaning if any change has to be made to that slide the same change has to be performed on all copies. This poses the risk of overlooking some copies, introducing inconsistencies and facilitating mistakes. There are several manners in which this lack of flexible navigation might be addressed, including the possibility to define non-linear navigation paths (Spicer et al., 2012, Edge et al., 2013) or zoomable user interfaces (ZUIs) (Good and Bederson, 2002, Lichtschlag et al., 2009, Haller and Abecker, 2010).

Separation of Content and Presentation Similar to the approach of LaTeX and other professional typesetting systems, content should be written in a standardised way with visualisation being handled automatically by the typesetting system. The clear separation between content and presentation makes the presentation tool handle the visualisation, allowing the authors of a presentation to focus on the content. Additionally, this facilitates experimentation with different visualisations. LaTeX does have a document class called Beamer which was designed specifically for presentations, but while we were inspired by its structured and content-driven approach, the content-related functionality and the visualisation possibilities are too limited to be considered as a basis for an extensible presentation tool.

Extensibility Rapidly prototyping innovative navigation and visualisation techniques, but also new content types and presentation formats, should be easy in order for a presentation tool to be successful as an experimental platform for new presentation concepts. It should be possible to add or replace specific components without requiring changes in the core. A presentation tool should provide a modular architecture with loosely coupled components to be truly extensible. Note that this type of extensibility should be offered on the level of content types as well as for the visualisation engine and content structures.

Cross-Media Content Reuse We have previously briefly mentioned the lack of content reuse in existing presentation tools. Even though there is a wealth of open education material available, it is rather difficult to use this content in presentations. However, the concept of transclusion does work well for digital documents and parts of the Web (e.g. via the HTML img tag). The seamless integration of external cross-media content as implemented in these environments should also be supported by any modern presentation tool. This includes several different mechanisms that enable including parts of other presentations (e.g. slides), transcluding content from third-party document formats, and including content from open learning repositories.

Connectivity Connectivity for multi-device input and output has become more relevant in relation to presentation tools with the rise of social and mobile technologies. Multi-directional connectivity needs to be supported for several reasons. First, it is a requirement to enable the previously mentioned cross-media transclusion from external resources. Second, multi-directional connectivity is the basis of audience feedback via real-time response or voting systems (Dufresne et al., 1996) as well as other forms of multi-device interfaces.

Interactivity As mentioned earlier, content can be more interactive and the extensibility requirement addresses this issue since the intended architecture should support dynamic or interactive content and visualisations. However, traditional human input devices might not suffice for components offering a high level of interaction. As such, a presentation tool should facilitate the integration of other forms of input like gesture-based interaction using Microsoft's Kinect controller or digital pen interaction (Signer and Norrie, 2010) as implemented by the PaperPoint (Signer and Norrie, 2007a) presentation tool.

Post-Presentation Phase Slide decks often play an important role as study or reference material, even if that was never their original goal. It is a trivial act to share traditional slide decks after a presentation, but this changes when the previously mentioned requirements are taken into account. The nonlinear navigation allows presenters to go through their content in a non-obvious order, and input from the audience might drive parts of a presentation, amongst other possible variables. Special attention should therefore be paid to the post-presentation phase. Playing back a presentation using the original navigational path, annotations and audience input should be made easy, while the content should also be made discoverable and reusable. With the rise and popularity of modern social media, there is a definite possibility to include the social aspect in a post-presentation phase through a content discussion mechanism.

3.4 MindXpres Platform

This section presents the global architecture of the MindXpres[3] crossmedia presentation platform as outlined in Figure 3.1, which addresses the requirements presented in the previous section.

Fig. 3.1: MindXpres architecture

3.4.1 Document Format and Authoring Language

A dedicated MindXpres document format is used to store, structure and and reference content. Content is stored, structured and referenced in Each individual MindXpres document contains the presentation's content itself and may also refer to some external content to be included. A new MindXpres document can be constructed by hand similar to how LaTeX is authored, or — in the near future — it may be generated via a graphical authoring tool. Contrary to other presentation formats such as Slidy, S5 or OOXML,

[3] http://mindxpres.com

the MindXpres authoring language abandons unnecessary HTML and XML specifics and focuses on a semantically more meaningful vocabulary. The syntax of the authoring language is almost entirely defined by plug-ins that enable the inclusion and visualisation of various media types and structures. To allow users some freedom in the way they present their information, the core MindXpres presentation engine does little more than providing a runtime environment for plug-ins and lets them define the media types (e.g. video or source code) as well as structures (e.g. slides or graph-based content layouts).

This also becomes apparent in the document format as every plug-in extends the available syntax. Any visual styling including different fonts, colours or backgrounds is achieved by applying specific themes to the underlying content.

3.4.2 Compiler

The compiler generates a self-contained portable MindXpres presentation bundle based on a MindXpres document. Although a MindXpres document could be directly interpreted at visualisation time, we decided to create this intermediary step for a number of reasons. First, the compiler enables the creation of different types of presentations from the same MindXpres document instance. This lets us not only create dynamic and interactive presentations but also more static output formats such as PDF documents for printing. Second, it is unwise to assume that there will always be an Internet connection available when giving a presentation. To overcome this possible issue, the compiler might create an offline version of a presentation with all necessary content pre-downloaded and included in the MindXpres presentation bundle. Last but not least, the compiler might resolve incompatibility issues by, for instance, converting unsupported video formats or including certain HTML5 libraries.

3.4.3 MindXpres Presentation Bundle

The dynamic MindXpres presentation bundle contains the compiled content along with a portable cross-platform presentation runtime engine which enables more interactive and networked presentations. Resembling the original document, the compiled presentation content still consists of integrated content as well as references to external resources, such as online content that will be retrieved when the presentation is visualised. However, it should be noted that the content might have been modified by the compiler and, for example, been converted or extracted from other document formats that the runtime engine cannot process. References to external content may have

been dereferenced by the compiler for offline viewing.

A presentation bundle's core runtime engine consists of the three modules shown in Figure 3.1. The *content engine* is responsible for processing the content and linking it to the corresponding visualisation plug-ins. The *graphics engine* provides all rendering-related functionality. For instance, some presenters may prefer a zoomable user interface because it provides a better overview of their content (Reuss et al., 2008). This graphical functionality is also available to the plug-ins, which can make use of the provided abstractions. The *communication engine* exposes a communication API which can also be used by plug-ins. It implements some basic functionality for fetching external content while also offering the possibility to form networks between multiple MindXpres presentation instances as well as to connect to third-party hardware such as digital pens or clicker systems.

Finally, the presentation bundle also contains a collection of *themes* and *plug-ins* as referenced by the presentation content. Themes may define visual styling on a global as well as on a plug-in level. The content engine encounters different content types and hands them over to the matching plug-in, which in turn uses the graphics engine to visualise the content.

3.4.4 Plug-in Types

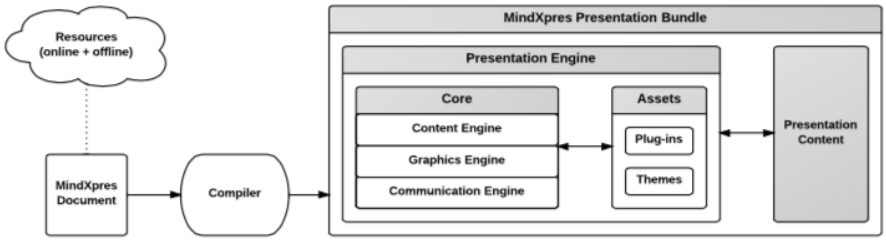


Fig. 3.2: Structure plug-in examples

In order to attain the required flexibility, all non-core modules have been implemented as plug-ins. Even the most basic content types including text, images and bullet lists are handled through plug-ins. We distinguish between three major categories of plug-ins:

- *Components* are the smallest elements of a presentation. The component plug-ins handle the visualisation for specific content types such as text, images, bullet lists, graphs or videos. The content engine invokes the corresponding plug-ins in order to visualise the content.

- *Containers* are used to group and organise components in a specific way. One example of such a container is a slide, where each slide contains different content but also some recurring elements. For instance, every slide of a presentation can contain certain elements such as a title, a slide number and the author's name, all of which can be abstracted at a higher level. Another example is an image container that visualises its content as a horizontally scrollable list of images. It's important to observe that MindXpres does not restrict us to the slide format and content can be laid out in many alternative ways.

- *Structures* are high-level structures and layouts for components and containers. They may scatter content in a graph-like structure or they may clearly group it in sections like in a book. These are radically different ways of visualising and navigating content but the plug-in abstraction allows the user to easily switch between different presentation styles like the ones shown in Figure 3.2. Structures differ from containers in that they do not restrict media types of their child elements in any way while they may influence the default navigational path through the content.

3.4.5 Implementation

HTML5 and its related web technologies were chosen as the backbone for the MindXpres presentation platform. Alternatives such as JavaFX, Flash or game engines were investigated as well, but HTML5 appeared to be the best option. The widely accepted HTML5 standard makes MindXpres presentations highly portable, as any device with a recent web browser can display them, including smartphones and tablets. Moreover, HTML5 offers rich visualisation functionality by design and the inclusion of Cascading Style Sheets (CSS) and third-party JavaScript libraries makes it a powerful visualisation platform.

Document Format and Authoring Language

The MindXpres document format that enables the simple expression of a presentation's content, structure and references is based on the eXtensible Markup Language (XML). Listing 3.3 shows a simple example of a presentation defined in our XML-based authoring language. The set of valid tags and their structure, apart from the `presentation` root tag, consists of what is provided by the available plug-ins.

```
<presentation>
  <slide title="Vannevar Bush">
    <bulletlist>
      <item>March 11, 1890 - June 28, 1974</item>
      <item>American Engineer, founder of Raytheon</item>
    </bulletlist>
    <image source="bush.jpg"/>
  </slide>
</presentation>
```

Lst. 3.3: Authoring a simple MindXpres presentation

Compiler

The compiler has been implemented as a Node.js application. Not only does this accomodate the use of the compiler via a web interface or as a web service, but projects such as node-webkit also enable the compiler to be executed as a local offline desktop application. The decision to use server-side JavaScript was influenced by the fact that Node.js has the ability to bridge web and desktop technologies. On one hand, the framework facilitates interaction with other web services and allows us to work with HTML, JSON, XML and JavaScript visualisation libraries during compilation. On the other hand, the framework can carry out tasks for which web technologies are usually not suitable, including video conversion, legacy document format access, file system access or TCP/IP connectivity.

To enable validation of a MindXpres document in the XML format described above, an XML Schema exists which is augmented with additional constraints provided by the plug-ins. After validating the document, it is parsed and any discovered tags might trigger preprocessor actions defined by the plug-ins, such as the extraction of data from referenced legacy document formats (e.g. PowerPoint or Excel) or the conversion of an unsupported video format. Each tag is then converted to HTML5 and all information is encoded in the attributes of a `div` element. The HTML5 standard allows custom attributes that start with a `data-` prefix. Listing 3.4 hilights converted parts of the original XML document we saw in Listing 3.3. Observe that no visualisation-specific information is included in the transformation, which merely results in a valid HTML5 document ready to bundle into a self-contained package together with the presentation engine.

```
<div data-type="presentation">
  <div data-type="slide" data-title="Vannevar Bush">
    <div data-type="bulletlist">
      ...
```

Lst. 3.4: Transformed HTML5 presentation content

Presentation Engine

The presentation engine's main purpose is to create a visually appealing and interactive presentation based on the compiled HTML content. As Figure 3.1 shows, the presentation engine consists of several smaller components which enable plug-ins to implement powerful features with minimal effort. The combination of these components allows for rapid prototyping and evaluation of innovative visualisation ideas. A resulting MindXpres presentation combining various structure, container and component plug-ins is shown in Figure 3.5.

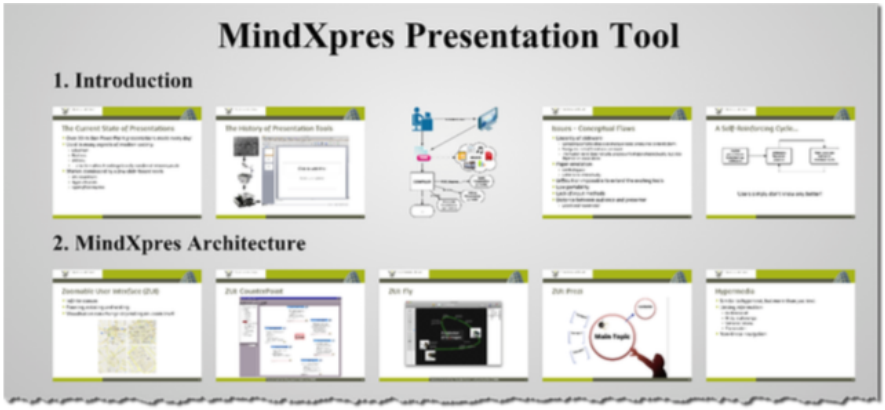


Fig. 3.5: A MindXpres presentation

Content Engine The content engine is the first component that is activated when a presentation is loaded. It uses the well-known jQuery JavaScript library to process the content of the HTML presentation. Whenever a `div` element is discovered, the `data-type` attribute is read and the corresponding plug-ins are triggered in order to visualise the content.

Graphics Engine The graphics engine accomodates interesting new visualisation and navigation styles. Apart from some basic helper functions, it provides efficient panning, scaling and rotation via CSS3 transformations and supports zoomable user interfaces as well as the more traditional navigation approaches.

Communication Engine The communication engine offers abstractions that enable plug-ins to retrieve external content at runtime. It also provides the architectural foundation to form networks between different MindXpres instances and to integrate third-party hardware (Roels et al., 2014). For the

MindXpres prototype, a small Intel Next Unit of Computing Kit (NUC) was used with high-end WiFi and Bluetooth modules to act as a central access point and provide the underlying network support. MindXpres presentation instances use WebSockets to communicate with other MindXpres instances via the access point. The access point also acts as a container for data adapters that translate input from third-party devices into a generic representation that can be used by the MindXpres instances in the network. In order to transcend simple broadcast-based communication, a routing mechanism was implemented based on the publish-subscribe pattern, allowing plug-ins to subscribe to specific events or publish information. The communication engine supplies the foundation for audience response systems (Roels et al., 2014) or even full classroom communication systems where the creativity of plug-in developers is the only limit.

Plug-ins Plug-ins are implemented as JavaScript bundles consisting of a folder containing JavaScript files and other resources such as CSS files, images or other JavaScript libraries. As a first convention, a plug-in should contain a manifest file with a predefined name. The manifest describes the plug-in using metadata such as the plug-in name and version but also a list of tags it provides and handles to be used in a presentation. The plug-in claims unique ownership for these tags and is solely responsible for their visualisation if the content engine encounters them. As a second convention, a plug-in must implement at least one JavaScript object providing certain methods, one of them being the `init()` method which is called when the plug-in is loaded by the presentation engine. The plug-in may decide to load additional JavaScript or CSS via the provided dependency injection functionality. A second method it needs to implement is the `process()` method which is invoked with a pointer to the corresponding DOM node as a parameter by the content engine when it encounters a corresponding tag. A plug-in is free to modify the DOM tree and may also register callbacks to enable future interaction with the content.

Themes CSS is currently the driving technology behind a basic templating system. These themes offer styling either on a global or on a plug-in level. It has always been the intention to replace this system with a more advanced layout engine that steers away from templates, and part of that is incidentally one of the goals of this thesis. The intention is to provide layout functionality far beyond what can be attained using templates. However, the styling functionality will still be handled by the current system. In the future this system may still be replaced or enhanced to allow more dynamic

styling using JavaScript.

3.4.6 Use Cases

To prove the merits of the architectural and technological choices, we demonstrate the extensibility and feasibility of MindXpres as a rapid prototyping platform through demonstration of a number of content- and navigation-specific plug-ins that have been developed so far. Additional plug-ins for audience-driven functionality such as real-time polls, screen mirroring and navigational takeover can be found in (Roels et al., 2014).

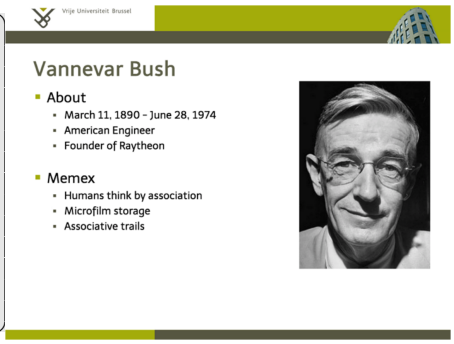
Structured Overview Plug-in

In Section 3.4 we described how structure plug-ins may alter and influence the way presentations are visualised and navigated. In order to illustrate this, a structure plug-in called *structured layout* was implemented, to combine a zoomable user interface with the ability to group content into sections. The resulting visualisation of the *structured layout* plug-in is displayed in Figure 3.5. Whenever a new section is reached while navigating through the presentation, the view is zoomed out to provide an overview of the content within the section and to convey a sense of progress.

Slide Plug-in

Even though one of the main intentions of MindXpres is to discard the concept of slides with all their limitations, it was deemed necessary to include support for this concept as well. As such, a slide-like container plug-in was created. While the benefits and issues of using slides with a fixed size are debatable, this plug-in was implemented as a testament to the framework's versatility. The main purpose of the slide plug-in is to produce a rectangular styleable component container with an optional title and some other information. Containers may also contribute functionality to layout their content. In this instance, the slide plug-in implements a quick and easy layout mechanism that allows the presenter to partition the slide into rows and columns. Slide containers are then assigned to these slots in the order that they are discovered. The use of the slide plug-in together with the resulting visualisation is depicted in Listing 3.6. This demonstrates the use of the image plug-in (a component plug-in) as well, which introduces a simple form of cross-media transclusion. A visualised external image can be cropped and filters (e.g. colour correction) may be applied without duplicating or modifying the original source.

```
<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>...
  </item>
 </bulletlist>
 <image source="http://example.com/bush.jpg">
  <crop bounds="10%, 5%, -10%, -20%" />
 </image>
</slide>
```



Lst. 3.6: Slide plug-in

Enhanced Video Plug-in

When showing a video in an educational setting, we often need more functionality than the average video player can provide (Reuss et al., 2008). MindXpres offers an enhanced video plug-in as demonstrated in Listing 3.7, adding the possibility to overlay a video with text or arbitrary shapes. This overlay functionality can be used as a basic captioning system as well as to highlight items of interest during playback.

Furthermore, there is an option to trigger certain events at specified times. One may mark certain points where a video should automatically pause at, highlighting an object and then continuing playback after a specified amount of time. Other features include the bookmarking of certain positions in a video for direct access and the possibility to display multiple videos in a synchronised manner. The enhanced video plug-in leverages the default HTML5 video player and overlays it with a transparent `div` element for augmentation. Currently it utilizes the HTML5 video API to synchronise the creation and removal of overlays but a SMIL-based implementation might be used in the future.

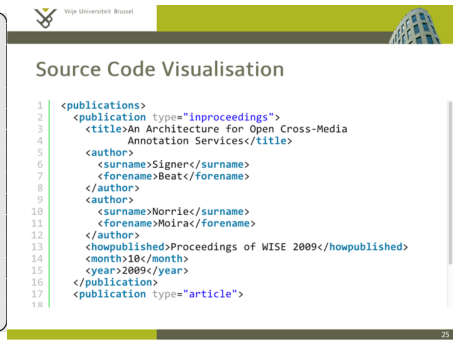
```
<video source="vid.mp4">
 <caption start="0:00" duration="1500ms">
  Lecture 3 - Butterfly Species
 </caption>
 <pause start="0:43" duration="5s">
  <caption>
   The peacock butterfly (aglais io) ...
  </caption>
  <highlight x="30%" y="9%"
             width="35%" height="40%" />
 </pause>
</video>
```

Lst. 3.7: Enhanced video plug-in

Source Code Visualisation Plug-in

We have previously mentioned the issues involved with visualising complex resources such as source code. Our MindXpres source code plug-in exports a `code` tag allowing the presenter to paste their code into a presentation and have MindXpres visualise it nicely through syntax highlighting using the SyntaxHighlighter[4] JavaScript library. Whenever the content engine encounters a `code` tag, this plug-in is invoked to beautify the code. It also automatically adds vertical scrollbars for larger segments of source code as illustrated in Listing 3.8.

```
<code>
 <publications>
  <publication type="inproceedings">
   <title>An Architecture for Open Cross-Media
          Annotation Services</title>
   <author>
     <surname>Signer</surname>
     <forename>Beat</forename>
   </author>
   <author>
     <surname>Norrie</surname>
     <forename>Moira</forename>
   ...
</code>
```



Lst. 3.8: Source code visualisation

3.4.7 Discussion and Future Work

MindXpres currently supports transclusion and cross-media content reuse through the use of plug-ins. For example, the image or video plug-in can visualise (and enhance) external resources, a dictionary plug-in could retrieve

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

definitions on demand via a web service or we might create a plug-in that lets us import content (e.g. PowerPoint slides) from legacy documents at compile time. Nevertheless, the introduction of generic reuse tags in our document format is actively being investigated. This would allow the presenter to transclude arbitrary parts of other MindXpres presentations. While the focus has been on the cross-media aspect of resources that can be used in a presentation, the cross-media publishing aspect might also be considered in the future via alternative compiler output formats.

The creators of MindXpres are aware that the current authoring of MindXpres presentations faces some usability difficulties. The average presenter cannot be expected to construct an XML document or any CSS themes. In order to address this issue and further evaluate MindXpres in real-life settings, a graphical MindXpres authoring tool is currently being developed. They further intend to provide a central plug-in repository which would enable novice users to find, install and use new plug-ins via the authoring tool in a simple manner. In the long run, the use of monolithic documents is to be revised and a move towards repositories of semantically linked information based on the RSL hypermedia metamodel will be executed (Signer and Norrie, 2007b). This will promote content reuse and sharing, while also creating opportunities for context-aware as well as semi-automated presentation authoring where relevant content is suggested by the authoring tool.

3.5 Layout

Proper layout is incredibly important when trying to transfer knowledge and information through written and visual media. Layout can help clarify boundaries and relations between pieces of information, by grouping and separating them appropriately. Layout is one component of a presentation's design, that — combined with other decisions — determines the number and nature of the visual representations of the information the creator wants to communicate, along with its format[5]. The layout of a presentation can have a tremenduous influence on its effectiveness in communicating information to, and obtaining information from, the audience it is meant to interact with. The importance of individual objects can be emphasised or minimised, and the connection between obects can be clarified or blurred. A well laid out presentation can provide a narrative for the viewer to discover, inferring correct links between the objects along the way, and to accomplish tasks quickly and correctly, increasing the presentation's effectiveness.

[5] The way the visual objects are realised (e.g. as text, graphics, UI widgets...), and their attributes (e.g. color, texture, font...)

Creating a good layout is almost never easy. People often spend more time on the layout of their presentation than the content. Most, if not all, decisions in layout are made by human beings. Some of them are professional designers who spend years learning and figuring out how to create effective layouts, and even then they may take hours or days to create even a single screen of a presentation. In fact, the more someone knows about proper layout and design, the more time they may spend perfectioning their work. However, sometimes time-critical information must be communicated and the layout process is too expensive and too slow to address these situations. This can be a serious problem (see also section 2.2.1). Many software packages have been developed to make this process easier, to get better results, to give more or less control to the creators. Many different approaches have been taken, and yet most of them still involve having a human being make the final decisions on the layout.

There are tools like PowerPoint, which give you some guidelines and some templates but generally let you do your own thing. If your own thing is entirely different from any best practices on layout, nothing will stop you. Other tools like LaTeX give you complete control over every aspect of the layout, while setting some sensible defaults so that you can get a good-looking layout without much effort, while still letting you do whatever you want once you overcome the steep learning curve that separates the casual users from the experts. There are tools that combine the power of LaTeX with the comfort of WYSIWYG editors, bringing the casuals a bit closer to the experts. But all of those tools have one thing in common: every aspect of every layout they create has, at some point, been designed and decided upon by a human being.

Aesthetics are a natural phenomenon, and the creation of aesthetically pleasing layouts is therefore a manifestation of our instincts. As with most instincts, it has proven difficult to translate this into a concept that can be understood by a computer. Moreover, it is still difficult to explain it in human terms, which — according to a popular quote often attributed to Albert Einstein — proves we don't fully understand it ourselves.

When we look to other technologies, we do find some automated layout implementations. For example, the web has had to adapt to mobile devices with small screens over the past few years, and has done this gracefully by creating the concept of responsive design. In short, this allows websites to adapt their layout to any screen, no matter the size. While this is often a hard-coded difference, where effectively two or more versions of the same webpage are created aimed at different screen sizes, some websites take a more dynamic approach based on constraints. As the space the page is to be displayed on gets smaller, the layout algorithm may decide to display

content below other content instead of side-by-side, it may scale images to fit the screen, it may even switch fonts and font sizes if necessary.

This constraint-based technique is described in a few papers (Lok and Feiner, 2001, Hurst et al., 2009), but has — to our knowledge — not been applied in any presentation software so far. This is surprising, as presentations often look like they could use some of this magic. A proper constraint-based layout algorithm could allow any user to drop content onto a slide, without worrying about clarity or even legibility, and the algorithm could take care of the rest. Of course, there are some limits in traditional slideware that may hinder this approach: if a user decides to put more content on a slide than there is physical room available, the algorithm could either make the content smaller or split it across several slides, but either solution may bring its own problems up. An advantage of ZUI's is that no matter how small the content gets, we can still zoom in to make it clear again[6].

3.5.1 Algorithms

When researching layout algorithms, one will often come across the very active field of graph layout (Di Battista et al., 1998). We will not go into the specifics of this field, as most of the issues with which it is concerned are specific to problems caused by the explicit visual representation of graph edges — for example, the minimisation of edge crossing (Di Battista et al., 1990, Shahrokhi et al., 1996). The same applies to automated layout as referring to automated circuit layout for VLSI chip fabrication (Hu and Kuh, 1985, Lengauer, 1990) as well as automated placement of pieces to be cut from a bolt of cloth used to produce clothing (Milenkovic et al., 1991). Contrary to presentation layouts (including graph layouts), these layouts do not attempt to make themselves understandable to the human mind, but rather are designed to meet the requirements of a fabrication process. While some techniques used therein definitely apply to our more general problem of automated presentation layout (e.g. general constraint solvers) others decidedly do not (e.g. bin-packing techniques (Hofri, 1980) that result in minimal area layouts at the expense of maintaining visually obvious relationships between objects).

In the remainder of this paper, we first discuss some of the simple ap-

[6] It should be noted that MindXpres in its current form does not support this level of zooming. While the software can zoom out to provide an overview of the presentation while zooming in on the separate components, it is not yet possible to zoom in or out extremely to reveal 'hidden' parts of the presentation. This is something we encourage to look into and change, because it can greatly improve both our layout solution as well as the whole MindXpres experience in general.

proaches to presentation layout used in current commercial software in Section 3.5.2. Next, we focus on the two methods that have been adopted by research systems that address automated presentation layout: constraints and learning. As described in Section 3.5.3, most research layout systems assume that a layout can be described by a set of constraints. Much of this work addresses applying constraint solvers and developing new ones. In addition, many researchers have focused on how to determine or extract the right constraints as well as how to to express them to a computer program. Systems that are not constraint-based generally fall into the category of learning systems, which we discuss in Section 3.5.4. These systems are sometimes trained by experts and other times by the user as the system assists in the layout process. In Section 3.5.5, we introduce some of the issues involved in evaluating presentation layouts. Finally, Section presents our conclusions and ideas for future work.

3.5.2 Simple techniques

Almost all contemporary user interfaces are built with a UI toolkit (e.g., Sun JFC/Swing (**?**), Microsoft Foundation Classes (**?**), and their ancestors, such as Xtk (**?**) or Tk (**?**)) that provides basic functionality, such as creating buttons and windows. Toolkits include layout managers (also known as geometry managers) to assist the UI designer (programmer) in designing the layout of objects in a managed container, without having to specify the absolute position and size of each object. Layout managers allow the programmer to say, in effect, add button or add button to this part of the window, and optionally specify additional numeric constraints. This makes it possible to create layouts that adapt to changes in the containers size.

A layout manager chooses positions and sizes at run time for the objects that it controls, governed by a set of constraints imposed by a simple layout policy built into the manager and parameters specified by the programmer. Typical layout policies include strict horizontal (row) or vertical (column) layout; row-major or column-major layout in which objects wrap to the next row or column to avoid exceeding the managed containers bounds; border layout in which objects can be specified to reside in the north, south, east, west, or center of the managed container; and grid layout in which objects are specified to reside at one position (or straddle multiple positions) in a programmer-specified 2D grid. Programmer-specified parameters indicate preferred, minimum, and maximum widths and heights of objects; and spacing, both between objects and between objects and the containers edges. Managed containers can be nested inside of other managed containers, and treated just like any other objects.

A programmer designs a parameterized layout as a hierarchy of managed containers, chosen for their layout policies, and further constrained by programmer-specified parameters. Thus, a layout manager does not actually design a layout, but rather instantiates a layout at run time from the structure and parameters specified by the programmer. Designing a simple layout (e.g., four buttons displayed in row-major order) is easy for a programmer, but designing a complex hierarchical layout, while possible, is tedious and difficult, especially if it needs to behave robustly when resized. The popularity of the layout-manager approach stems primarily from the ease of implementing the managers themselves and the relative ease with which they can be used by programmers to define simple parameterized layouts. In addition, because the final layout is determined at run time, the presentation can work well under a wide range of possible window sizes, with lower bounds on usability imposed by the minimum effective sizes of the objects being laid out.

Commercial word-processing and presentation systems intended primarily for sequential presentations (e.g., Microsoft Word, Publisher, and PowerPoint, Quark Express, and LaTeX), provide a set of preauthored style templates (and the ability to create new ones) that can be applied to existing material; for example, by matching markup tags present in the material to be processed with corresponding tags in the template. In comparison with the parameterized layouts of UI toolkit layout managers, most of these template-based systems are simpler. They emphasize the format of the material to be presented, relying on the order in which objects are specified as input to directly determine the order in which formatted objects appear in the presentation, with the exception of their handling of floating objects. Floating objects, such as tables or figures, can move relative to surrounding objects (typically text). Typically, a set of rules is used to determine the final position of the floating object; for example, If the height of the object is more than 60% of the height of the page, then put the object on a separate page; otherwise, put the object at the nearest paragraph break. However, users of these systems often move floating objects around by hand to guide or overrule simplistic placement policies.

In the following sections, we discuss the two techniques that have been explored in automated layout systems: constraint satisfaction and learning.

3.5.3 Constraint satisfaction

The vast majority of research in automated layout to date focuses on constraint-based methods (e.g., (**????????**)). The idea that layouts can be represented as a set of constraints is very intuitive. For instance, consider the constraint

relationships depicted in Figure 1. The goal of a constraint-based automated layout system is to take such a constraint network and generate a set of positions and sizes for each of the components in the network. Any constraint-based automated layout system can be characterized by the kinds of constraints it uses; how they are described, obtained, and resolved; and how the system addresses constraint inconsistencies, loops and other hazards that might prevent the solution from converging. In the rest of this section, we explore these issues.

Types of constraints

In a constraint-based automated layout system, most constraints can be classified as either abstract or spatial. By abstract, we mean that the constraint describes a high-level relationship between two components that are to be included in the layout (e.g., TEXT1 REFERENCES PIC1). In contrast, spatial constraints enforce position or size restrictions on the components (e.g., CAPTION1 BELOW PIC1). Spatial constraints can be fed directly into a constraint solver for resolution, whereas abstract constraints must be processed and reduced to spatial constraints before the layout can be realized. The process of generating spatial constraints from the abstract constraints is often the most challenging part of creating an automated layout system.

The choice of employing abstract or spatial constraints depends on the nature of the layout that the system is designed to generate. Many research systems employ both abstract and spatial constraints because they are general multimedia presentation tools (**???**). Other research initiatives are embodied in the form of libraries such as subArctic (**??**) and the Garnet toolkit (**?**) that include extensions of layout manager type functionality with simple spatial constraints. Although automated layout systems for more limited environments can create effective layouts using only spatial constraints (**?**), the same can not be said for attempting to employ abstract constraints without spatial constraints. Figure 2 depicts what the difference in output might be between a system that considers only abstract constraints versus a system that takes into account both abstract and and spatial constraints. This issue is discussed further in Section 3.5.3.

Abstract constraints Abstract constraints are descriptions of high-level relationships between the various components that are to be placed into the layout. Abstract constraints such as TEXT1 REFERENCES PIC1 and TITLE IS IMPORTANT are sufficiently high-level that content authors can easily specify them and are particularly effective for use in interactive systems because the author of the content needs no additional technical or

artistic skill to specify them.

Although one might think that TEXT1 REFERENCES PIC1 implies that B and C are relatively close to each other in the generated layout, abstract constraints in and of themselves do not specify the position and size of the components of a layout. This is because the mapping between abstract constraints and spatial constraints is performed by a translation component before spatial constraints are passed to the numeric constraint solver. The abstract spatial constraint translator can choose to map the abstract constraint TEXT1 REFERENCES PIC1 into any set of spatial constraints. This might mean that the PIC1 is placed right next to text TEXT1, or that PIC1 is placed on a different page and some visual cue is left to guide the end user to it from TEXT1.

Spatial constraints Spatial constraints are relations that directly express the geometric structure of the presentation. For instance, we may wish to force a certain block of text to appear beneath another block of text that the user is assumed to have read first. Another instance of spatial constraints would be to force all objects to occupy a space that is of a certain size or an integral multiple of that size.

There are a number of reasons why we would want to impose spatial constraints. Perhaps the most prevalent reason is to improve upon the visual quality and aesthetics of the presentation. Many early automated layout systems are created from the perspective of computer science and mathematics alone (**?**). Such systems tend to treat the problem as a purely theoretical question of tiling and use optimization techniques to find a solution (**?**). These kinds of systems will often not take into account simple legibility rules (e.g., text should be placed into columns that run down the entire page rather than having blocks of random size packed onto the screen) and style guidelines (e.g., all captions go beneath their associated figures and spacing between a figure and surrounding text block should be the same everywhere). Figure 2 exemplifies what might happen in a system that employs abstract constraints without spatial constraints. A system that considers only abstract constraints will not be able to generate layouts with the same aesthetic appeal as systems that consider both because the system has no visual restrictions on where components of the layout are placed.

The method by which the components are represented may place some kind of limitation on what kinds of spatial constraints may be used. One such issue that may arise is sometimes referred to as the Cousins Problem, an example of which is shown in Figure 3. This problem may arise if the data structure in which the components are being stored does not allow referencing

one components children from a different components child.

It seems intuitive that we would want to use concepts from graphic design to create legible and pleasing output. Some systems enforce the presentation to conform to a grid system, similar to those used to lay out newspapers (**??**). In a grid system, every screen or page of the presentation is divided into an array of upright rectangles. Each object must occupy one or more complete rectangles. Figure 4 is an example of output from Feiners GRIDS system (**?**) which designs layout grids that enforce a consistency between screens or pages of a presentation. One complication with employing grid systems is that a graphical component may need to be cropped, padded with a border or have its aspect ratio changed This is because uniform scaling of the object may not be sufficient to make the object occupy an integral number of grid rectangles.

Automated layout systems for well-defined environments, such as network diagrams or label placement, often employ spatial constraints exclusively (**??**). These systems consider issues such as the proximity of the components being placed, the distance between a label and its target, and the possibility of confusing the end user by placing multiple labels that are sufficiently near the same target that the end user doesnt know which label is associated with it. Abstract constraints are often used for formatting labels (larger cities have bigger names), but are generally not used for layout directly.

Some systems allow the user to specify abstract data constraints separately from spatial constraints. This allows for a logical single presentation to have many different skins, opening the door for a single presentation to be displayed using different media (**?**). This is particularly effective for interactive layout systems that might want to maintain a separation between content authors and layout experts. A similar approach that leverages this concept is to build the spatial constraints into the system, thereby eliminating the need for human intervention to specify spatial constraints for each layout to be generated. Feiners GRIDS is an example of a such system (**?**).

Spatial constraints are sometimes used to increase the efficiency of the constraint solver. For instance, a constraint might be placed on all objects of a certain type that permits them to be resized in only one direction (**?**). Similarly, imposing the constraint of a quantized display permits the use of fast fixed point and integer programming techniques when resolving the set of constraints.

Expressing constraints

Intuitively it would seem best to define a formal grammar to describe the method by which the constraints are expressed. This approach benefits from

being able to leverage a rich body of existing research for manipulating and parsing constraints. A rich grammar might be very flexible and expressive and translate into better layouts (?). An example of a grammar for use in a layout system is shown in Figure 6. However, powerful and expressive grammars may also be difficult to use. This is especially true of grammars or ontologies that attempt to be extremely general and all-encompassing. In addition, a very complex solver may be necessary to process the information present in such a system. As one might imagine, it is extremely difficult to create a system that describes the set of all possible high-level relationships between components of a presentation, although this has been explored for the use of automatic graphics generation (?).

Another extremely powerful approach is to express the constraints in terms of Boolean predicates (?). This approach alleviates some of the concerns that arise from the more expressive grammar and relational grammar approaches by limiting the space of what can be expressed. The use of Boolean predicates also eases the process of solving the set of constraints as the input needs little or no translation before being passed to the solver.

Obtaining constraints

One of the most important practical issues in implementing a constraint-based automated layout system is determining where to obtain the constraints. Approaches that have been tried range from fully automated to the computer making suggestions.

Many automated layout systems implement abstract constraints by gathering them from structured input data (Mackinlay, 1986, ?, ?, ?). These systems take tables of numeric data and automatically create presentations. The structure of the data provides all the relationships that are needed to generate the layout. Other systems that are designed for multimedia layout have languages to explicitly specify the abstract constraints to describe relationships such as author-of, description-of and precedes between the components (??). The assumption that input data is readily available in a structured form is becoming increasingly valid because the information that we create is beginning to be stored in more structured formats (?). Such work is prevalent in the layout modules of automated graphics generation systems (?).

Interactive specification Interactive constraint specification systems are also extremely popular, but suffer from the obvious limitation that they require user input. Some systems are designed to help graphically naive content

authors create professional quality layouts. Others are meant to reduce the amount of time a graphic designer needs to spend on a layout.

Most systems that take interactive input do so for spatial constraints (**???**). This is because it is easy to create graphical user interfaces that allow the user to interactively place or adjust components on the screen. Although providing a graphical user interface to specify abstract constraints is not unheard of, abstract constraints tend not to need adjustment. Roths SAGE system (**?**) allows a user to associate database records with visual elements.

Some of the interactive systems require the user to specify the high-level design of the document and then automatically generate the final result (**?**). This approach is very useful in situations where the goal is to enable a content author to create layouts without the need for intervention by a layout expert. Some other systems take the opposite approach where the system produces the initial layout and allows the user to refine it (**?**). These systems are more applicable in situations where the goal is to save the amount of time that a graphic designer needs to spend to create the layout.

Automated extraction Fully automated constraint extraction is the least explored method of obtaining constraints. Some work has been done in integrating natural language techniques with automated layout (**?**). This is particularly effective if natural language generation is being employed to create the content. Since the content generators are computer programs, it is much easier to have the generator send abstract that describes relationships between components as well as markup the text with flags denoting which parts are particularly important.

Another method that has been explored is to extract abstract constraints from the entity relationships found in SQL databases (**?**). Unlike the natural language generation system that passes additional information to the layout system, in this approach abstract constraints are derived from data structures that were originally intended for use elsewhere.

Constraint solvers

A constraint-based automatic layout system must have some way to resolve the constraints with which it is presented. Formally, automated layout techniques all solve a form of the constraint satisfaction problem (CSP) (**??**). Both randomized and deterministic algorithms have been applied to solve the problems in this field. In general, the constraint solvers employed can be categorized as applying either a local or a global methodology.

Local techniques Local constraint solvers attack the constraint satisfaction problem bottom-up. This might be compared to inductive reasoning, where a small subset of the universe is first solved. Two routes can be taken to solv the rest of the constraints and create the final presentation. The first approach is to solve many small subsets of the constraints independently and then run a second resolution phase to combine the results. An alternative approach is to iteratively resolve constraints at the border between the constraints that have already been solved and those that have yet to be considered (**?**).

Using a local-resolution technique can be problematic if the solver encounters local minima (**?**). By resolving small subsets first, the solver may make decisions that bring the system to a suboptimal final solution. The advantage, however, is that local-resolution techniques usually execute much faster than global techniques.

Global techniques Global constraint solvers attack the constraint satisfaction problem from the top down. Unlike local techniques, global techniques generally do not suffer from the problem of local minima, but require more computation time. To address the issue of additional computation time, numeric solvers that use iterative approximation techniques have been applied (**?**). Randomized computation techniques (e.g., genetic algorithms and simulated annealing) have also been applied for label placement (**?**).

Inconsistency policies

If the set of constraints is sufficiently large, there is a strong likelihood that there will be some problems. In particular, some constraints may contradict others and possibly make the system of constraints unsolvable. A resolution policy must be specified to generate a layout in these cases.

Some systems take a very simple approach to inconsistency by avoiding it. Rather than bogging the system down with inconsistency handling, the grammar used to articulate the constraints is designed in such a way that cycles cannot occur (**?**).

Another popular method for handling inconsistency is to apply priorities to the constraints (**?**). By permitting each constraint to have an inherent priority, the system can make intelligent decisions about which constraints to drop should a contradiction be encountered. A problem can still occur here if two conflicting constraints have the same priority. In this case, the system can use the AI technique of applying a tie-breaking strategy (e.g., first-come first-served or pick one randomly) so that the layout can be generated (**?**). Priorities are critical for generating effective layouts in systems where there are complex networks of both abstract and spatial constraints. For example,

the enforcement of the grid in a system that employs design grids must take precedence over all other constraints.

3.5.4 Learning techniques

Machine learning has been applied to many automated multimedia authoring systems, including speech synthesis (**?**) and natural language generation (**?**), as well as to graph layout (**?**). However, most automated layout systems do not leverage the large body of existing work by the AI community in machine learning.

Automated layout systems that do have some form of learning tend to use it during the interactive specification of constraints (**??**). These systems do not implement full machine learning systems. Rather, they try to learn based on interacting with the user. The Marquise system (**?**) allows a user to set the system into a training mode where the relative locations of components are demonstrated to the system. Spatial constraints that will be used to generate the layout are then extracted from this interaction with the user. If the constraints cannot be extracted, the system falls back to having the user specify the position explicitly as a LISP function. Bornings (**?**) ThingLab is similar in that it allows for demonstration of constraints, but also adds the ability to demonstrate animation.

Some recent work in automated graphics generation has also explored the use of learning techniques (**?**). Zhou divides the space of rules that need to be acquired for presentation generation into three categories: information learning space, visual learning space and rule learning space. Visual learning space is directly related to spatial constraints, and thus is similar to Myers and Bornings work. Unlike Marquise and ThingLab however, Zhous system employs full-strength machine learning that can be fully automated by providing the system with a large dataset of presentations designed by a layout expert in addition to the interactive methods seen in other work that employs learning techniques.

3.5.5 Evaluation techniques

Whenever a piece of software is used to perform a task that is traditionally believed to be reserved for human experts, the question that will always be asked is whether or not the computer is as good as the human. In the field of layout, good may refer to the usability (e.g., whether tasks can be accomplished more quickly, with fewer errors, with greater user satisfaction), as well as the aesthetics of the presentation (e.g., whether end users or graphic designers think that the results look as good as ones produced by humans).

In some sense, all interactive layout generation systems have a module that handles the evaluation of how good the layout is: the human user. By using a computer-based evaluation mechanism, we could evaluate the layout automatically without relying (as much) on the user, ideally feeding back the results to redesign a layout that is not deemed good enough.

Evaluation of user interfaces, independent of who or what designs them, has been explored by a number of researchers (**?????**). The focus of this research has primarily been divided between creating heuristic inferences and quantitative metrics. Heuristics are of course more flexible, whereas metrics are easier to incorporate into computer systems and hence more common in automated evaluation mechanisms.

Sears has explored the application of metrics to automated layout systems (**?**). He employed metrics to evaluate how usable an interface is based on the amount of mouse movement that is needed between button clicks. Figure 7 shows screen shots from Searss system. The additional information provided by these metrics are embodied in the form of spatial constraints. Fitts Law (**?**) implies that buttons that are often clicked sequentially should be placed close to each other spatially, since this reduces the time needed to move between them.

Other research has addressed adding evaluation to the automated layout process in the context of user modeling (**?**). This work proposes that the interface not only include or exclude certain elements based on the type of user, but that it also change the layout. This information may be gathered at run timeas the frequency of use of different parts of the system change, the layout could be modified to reflect this. An appropriate user model could make it possible to adapt the output for the user. Note, however, the potential for change to create a less, rather than more, effective UI by clashing with the users mental map of the user interface.

3.5.6 Conclusions and future work

We have illustrated the range of research that has been accomplished in the field of automated layout, from simple techniques to research systems. As data presentation needs rapidly increase, the field of automated layout will become increasingly important. In time, we feel it is inevitable that the more powerful techniques found in the research systems will make their way into popular software packages. Reviewing current research, we see a number of rich possibilities for future work.

Integration of natural language techniques with automated layout systems has been explored to some extent, particularly with natural language generation. However, similar work has not been pursued with image and video

understanding or speech recognition. It may be possible to extract abstract or spatial constraints for automated layout by applying well known vision or speech recognition techniques to multimedia components of a layout.

Another interesting possibility is considering how to handle constraints that are wrong. Most systems have a user specifying the constraints in some manner at some point in the layout pipeline. The problem is that the user might simply make a mistake and not really mean what what he or she specified. In a system that has support for ranking constraints by priority, the user might also assign incorrect priorities. Constraints that are automatically extracted opens up even more doors for feeding incorrect information to the system. The use of natural language understanding, image understanding or speech recognition to extract constraints by definition means that there will be some probability of error in the constraints. Enabling an automated layout system that would be capable of handling these kinds of problems might involve applying AI techniques from adversarial game playing (**?**). Algorithms from computational biology and genomics may also be applicable because this kind of problem is encountered during gene sequencing (**?**). Constraints extracted by an automated process can be verified by running multiple extraction algorithms and having them vote. A similar approach is employed by an object-recognition technique called the Hough transform (**?**). By applying algorithms to defend against constraint error, a system might be made robust enough that errors in the constraints could cause little or no loss of effectiveness in the presentation.

There are other kinds of constraints that might be worthwhile to consider, some of which be obtained through hardware capture of information for user models. For example, real-time systems that automatically generate a user interface would benefit from knowing the users distance from the display medium (information that was taken into account at the beginning of the layout design process of (**?**), but not computed automatically). This could be determined from any of a variety of head-tracking technologies. Eye tracking could also be used to extract additional constraints based on where the user is looking. What parts of the display the user has actually seen would be useful information to pass to the automated layout system.

4. APPROACH

In this chapter we explain the different approaches we tried in order to reach our goal and find a solution for the problem we described. As you will see, this was not immediately a straightforward process but rather one of trial and error. The goal was clear, the starting point was clear as well, but as often in computer science, there is more than one way to get from point A to point B, and it is not always clear which way is the best, easiest, most efficient or most effective.

Since we're talking about the approach here, and not the implementation (for that, see chapter 5), we start by describing in broad terms what needs to be done and how this should be done, then we refine until we have a full set of specifications ready for implementation, where the last details will be ironed out.

Unfortunately it is possible to refine an approach until it is ready for implementation, and only find out during implementation that the approach you've chosen will not work. This happened during our work on creating an automated layout system. Luckily we still had time to go back to the drawing board, and we did not have to restart from scratch; large parts of our approach were correct, the basic layout process we thought out was still a viable part of the approach, but it turned out we would have to split up the conversion and layout parts into two separate processes, rather than implementing them as two steps of the same process.

Specifically, we had thought at first to figure out the ideal layout during conversion, when we would have all the separate components, by immediately putting them in the right place. This idea was partly conceived after looking at the HTML code generated by the MindXpres compiler, thinking we would generate the same HTML code in our conversion process. It turned out we could bypass the MindXpres compiler this way, but that wouldn't be necessary: we could just as well generate MindXpres XML and have the compiler take care of the rest for us.

We also found during implementation that generating a layout in Java would not easily give us the results we were hoping for. However, at this point we had realized generating MindXpres XML would be a better option, so we could have MindXpres take care of the layout for us. Except MindXpres

didn't do fully automated layout yet, the layout system was mostly template-based, so we decided we would need to write our own MindXpres plug-in that would solve this problem for us.

4.1 Conversion process

The first part of the approach is fairly straightforward in its basic explanation: we had to convert PowerPoint presentations into MindXpres presentations. This involved finding out how PowerPoint presentations are structured, getting the parts wee need out of that structure, and then putting those parts together in de MindXpres structure.

It appeared soon enough to us that the nature of this process resembled that of a compilation process. A compiler takes source code and transforms it into a working program with the semantics described by that source code. The compilation process consists of several steps. First the source code is tokenized, which means the symbols in the code are identified one by one and classified in certain categories.

Then the tokens are processed by a parser into an intermediary form called a parse tree. A parser looks for certain predefined patterns in the source code. These patterns are part of the source code's language syntax. As such, these two steps analyse and validate the source code's syntax. If part of the code does not match any pattern, the parser and the compilation process stop and the user gets a message saying the code's syntax is invalid.

When a parse tree is constructed, the compilation process can alter it, to improve it. Certain patterns in the parse tree may be replaceable by different patterns with the same outcome, but with more optimal execution. This part of the compilation process is optional, and is called compiler optimization. Optimizations can consist of many things, depending on the language. For example, some languages guarantee tail call optimization, where infinite loops can be constructed by letting a function call itself as its last statement without causing a stack overflow. This is something the compiler (or interpreter) can optimize during this part of the compilation process.

After this, the parse tree can be written out to produce the desired output. Every node in the tree has a well-defined equivalent in the target language's syntax. The target language can be Assembly, which consists of the exact instructions a CPU needs to carry out a program, or it can be another programming language. Many compilers of higher-level languages translate their language into C, for several reasons: the C compilers that translate C into Assembly have been optimized so much that it is easier to rely on them than to put an enormous amount of effort into optimizing another language;

C compilers exist for most — if not all — CPU architectures, which means translating a language into C makes it compatible with all those architectures, while it would cost a lot more effort to write different compilers for every architecture you would want to make your language available on.

The conversion tool that is the purpose of this thesis, can be described in a similar succession of steps. As a first step, we take a PowerPoint presentation and take it apart into its components, effectively walking over each component, classifying them and registering their content type, original position and size, and any other specific properties. This can be seen as the tokenization phase, after which we end up with a series of 'tokens' or, in our case, presentation components.

We then turn this series of 'tokens' into a 'parse tree', an intermediary structure that reflects the relation between the components and the hierarchy of the presentation, which may consist of chapters, sections, slides and component groups. In PowerPoint this structure is fairly simple, so the creation of this 'parse tree' is a straightforward process.

However, in MindXpres we are not limited to the rigid hierarchy of sections and slides, so at this point we can actually start manipulating our tree and improve upon it, for example by moving parts around, nesting components in different ways, grouping them in other ways than they originally were, etc. In compilation terms, this is the optimization phase, where the compiler can manipulate the program to run more efficiently, to replace parts of it with other functionality, or to add features the source didn't explicitly specify (e.g. garbage collection, but also spyware components (Scahill and Begley, 2015)).

As we discuss in section 4.2, this seemed like the right time to incorporate automated layout generation into the conversion process. As we see later in section 4.3.1, it turned out it wasn't. In the end, no significant 'optimizations' or manipilation of the tree structure were implemented. Later on we would utilize this optimization phase to enable automated layout in another way, without actually performing the layout here, but at this point it would not affect the end result in any way.

To finish the conversion process, we can traverse our component tree and generate a MindXpres presentation from it. This can be done in several ways, since our intermediary form is in no way dependant on or bound to a specific format. Since the MindXpres compiler was unavailable for a long time during our research and implementation, we decided it would be best to go straight to HTML5, so that we could test the conversion process without relying on the MindXpres compiler. This worked out fairly well, although manually constructing HTML5 to work with the MindXpres JavaScript library proved difficult. We ran into several issues, often mostly due to our lack of knowledge

of the inner workings of MindXpres, but we managed to get a presentable result that emulated the original PowerPoint presentation quite well.

Afterwards, we altered our conversion tool to generate MindXpres XML instead, which was a lot simpler since we would rely on MindXpres to provide our layout and other things for us through the MindXpres compiler. This approach allowed us to use the full power of MindXpres, including our own plug-in for automated layout. At this point, the optimization phase was also revisited, and leveraged to introduce specific XML tags around component groups that would trigger our automated layout plug-in.

4.2 Compiler optimizations

Since the conversion process resembles that of a compiler, it seemed logical at first to make automated layout a part of that process, as some kind of 'compiler optimization'. During this phase in the process, the component tree would be manipulated and altered, with the express purpose to improve upon its structure and properties, so as to get a better end result. Our improvements in this case would then consist of the automated layout.

As a first attempt, we tried to traverse the component tree, giving each object new coordinates and sizes based on their original coordinates and sizes, as well as the coordinates and sizes of objects around them, so that they would fit together on every slide as well as possible. This seemed an easy solution, but the results were sub-optimal. On top of that, we soon realised that we were in essence creating another template-based system that would generate slides and presentations based on predefined ratios and rules, which was exactly the opposite of what we were trying to do. As such, we abandoned this approach in favor of a constraint-based algorithm as described in section 3.5.1.

This involved a technique that at first sight may seem like yet another template system, but actually is completely different: defining constraints for every component, in the form of margins, maximum sizes and other limits, and then calculating a way to satisfy all constraints while fitting content together on each slide. The similarities with template-based systems exist in the presence of predefined constraints, ratios and rules, but the important difference is that these constraints are defined relative to the component itself, without specifying anything absolute about location or size. For example, we would retain the aspect ratio of an image, without specifying its size, so that the image may be scaled to accomodate other components in a dynamic layout. As another example, we might specify there needs to be a certain distance between a component and any other components, relative

to its size. We could also specify a certain relation between components, ensuring components stay in each others vicinity, one should always be left of the other, no other components may be placed between them, etc. Using these rules, we would then programmatically calculate the best layout using those components, but without any other bias. These constraints would be based only on the original situation, never on any suggestions from us or other developers or authors, which makes all the difference with traditional template-based layouts.

While this is clearly a better method, it turned out the compiler optimization phase was not the best place in the process to take care of this. While we had the necessary data to calculate the layout, we would have had to generate the layout along with the MindXpres presentation, after which the presentation could not be altered anymore without breaking the layout. This defeated the purpose of exporting to MindXpres, which was to allow the presenter to edit, extend and improve their presentation further using MindXpres. What we needed was a way to get MindXpres itself to generate the layout, even if we wanted to add components to the presentation afterwards, and even if we wanted to create a new MindXpres presentation instead of starting from PowerPoint. After all, how would we convince people to drop PowerPoint for MindXpres's automated layout capabilities if they could only use that functionality by starting from PowerPoint?

In the end, we decided to change our approach again. We took the automated layout out of the conversion process, instead opting at this point in the process to only add the necessary layout triggers in the form of an enclosing XML tag around the components that would need to be included in the automated layout. As such, the generated MindXpres XML would include those tags, and a plug-in (described in section 4.3.1) would then generate the layout at runtime.

4.3 Using MindXpres

One of the primary goals of MindXpres is to separate content from layout, allowing the author of a presentation to focus on the content while MindXpres takes care of the layout. The way it does this is currently mostly through the compiler, which decides the width, height and coordinates of content, relative to the container the content belongs to. The plug-ins responsible for handling components and containers currently don't mess with those settings, but technically, they could. The compiler decides the measurements and coordinates based on templates. The solution we were looking for was a layout engine that could take any content and put it in an appropriate

layout without any directions from the user. As such, we had to enhance MindXpres's layout engine to use constraints, based on the size of the content, and try to find an optimal position for every component it is given.

4.3.1 A MindXpres plug-in

We did this by creating an invisible container plug-in. Containers are a way of grouping components, other containers, etc. in MindXpres. This means they have control over their child elements, which gives us the opportunity to override the layout of those elements. A container plug-in thus allows us to implement our own layout system. Since it's a new element, it doesn't override existing elements as it would have done if we had, for example, rewritten the 'slide' plug-in. The user can decide for themself whether or not to use it, and it can be used anywhere in the presentation: wrap the whole presentation in it, or just a small part, whichever works best for your purposes. It also won't break existing presentations that don't use it, while those presentations can very easily be altered to take advantage of it.

An important aspect of this is that containers can be nested. This means we can create slide-based presentations, which can contain our `autolayout` container, which then contains the slide's contents, thus creating an optimal layout of the content per-slide. Another way of using it could be without slides, throwing all content together in one `autolayout` container, and letting it take care of the layout for the whole presentation at once. It should be noted here that the `autolayout` container makes each of its child nodes focusable separately, to compensate for arbitrary resizing it may perform on large objects in order to fit them next to other content, by using the focus functionality to automatically zoom into these components when necessary.

We call it an *invisible* container plug-in because it does not introduce any visual content, shape or indication for itself. Compare with the `slide` plug-in which obviously puts some kind of slide-look around the content it encompasses, and it becomes clear what we mean by this: although the content within is obviously affected by our plug-in, there is no visible indication of its presence to the audience.

The plug-in uses the compiler's numbers to decide relative locations between components, as well as size ratios, and then finds a way to display those components in a way that the display order makes sense (or at least matches the intended order as closely as possible), that no overlapping occurs (since we don't have the animations that PowerPoint might have used to display one piece of information and then another on top of it), and resizing everything if necessary in order to fit within the specified container. While this may seem like a bad idea since content can get illegibly small this

way, keep in mind that we can rely on the ZUI[1] to focus on each component separately, or on groups of components, while PowerPoint obviously can only display the whole slide at once.

In this manner we would generate MindXpres presentations that were immediately usable, while also being adjustable; and on top of this, we would allow the automated layout process to be used in other MindXpres presentations that were not originally converted from PowerPoint slides. The goal of this plug-in would thus be to provide automated layout functionality to MindXpres presentations, and to allow any MindXpres author to use it simply by putting an `<autolayout></autolayout>` container around the components they want the plug-in to act on. This approach has the additional advantage that the container can be used multiple times throughout the presentation, while also allowing other parts of the presentation to have a manual layout.

Since it is possible to nest containers, which means a number of components could be grouped together in an `autolayout` container, then the result could be put into another `autolayout` container together with other components — other `autolayout` containers, perhaps — to generate an automated layout for an overview of the different groups. Compare it with traditional slideware, where components are grouped together in slides, then the slides might be put next to each other in an overview — except our approach drops the slide boundaries, while still maintaining the ability to group components together to show a relation or link between them.

4.3.2 An automated layout algorithm

As discussed earlier, our first approach included an algorithm where content would be placed on slides according to certain rules, trying to attain a mythical 'perfect' layout based on the golden ratio, symmetry, centering and other general guidelines we would find in advice on creating presentations. It turns out that, while following those guidelines as a human being is generally a good idea, a computer has different ways of calculating a good layout. The issue here can be compared to other problems in computer science; for example, people in the robotics department have tried for decades to create a robot that behaves exactly like a human being, and people in artificial intelligence have tried to create an AI that thinks like us. However, we've found time and time again that computers simply aren't very good at acting, thinking or being human, just like we aren't good at being computers. Making a computer act like a human makes it disadvantaged — almost by definition, just

[1] Zoomable User Interface

like we are severely handicapped when we try to perform typically automatable, repetitive and/or math-intensive tasks. A computer's true power only shows when you let it do what it's good at, which is the repetitive stuff, the mathematically complex stuff, etc. Trying to make it generate presentation layouts like a human would, is asking for subpar results.

If we approach this problem keeping in mind a computer's strength and weaknesses, we arrive at a different approach. This involves calculating sizes, ratio's, positions, margins and other numbers, of which the formulas are actually not too hard to come up with as a human, but which the execution is definitely more of a computer task. We start off by checking each component, and noting its original location and size. We then try to find components that are in proximity of each other, and figure out their original layout: above/below each other, next to each other, overlapping... Then, we try to put them together, possibly resizing them to match each other's sizes, and trying to match their original relative locations while introducing a certain rigidity, or consistency, by aligning them properly and puzzling them together as neatly as possible.

This last part may sound weird, but it really is something to take into consideration, especially when the amount of components might be much bigger than what should fit on an average traditional slide. You could put all components in a row, just displaying them side-by-side, but that is not very aesthetically pleasing. Instead, we opted to try and keep components close to each other. This was finally achieved by finding the location closest to the starting point that would fit the component being considered, while still taking into account the earlier constraints about relative location and size. Thanks to the ZUI in MindXpres, this makes for interesting layouts that still remain manageable, and provide a nice overview of all content when zoomed out.

5. IMPLEMENTATION

To implement the *ppt2mxp* conversion tool that is the subject of this thesis, we chose the Java programming language (Gosling and McGilton, 1995), version 8. Although the author has significant experience with lots of other, more interesting, more compelling, more fun languages, several reasons pushed us towards Java, the least of them being its ease of use. Of course, Java *is* easy to use — it would not have become as popular as it is nowadays if it wasn't. It has a fairly clear and logical syntax, a consistent structure, and an extensive standard library. At conception in 1995, its performance was abysmal, but throughout the years it has steadily improved and somewhere between Java 5 (then still called 1.5) and 6 (when they dropped the '1.' prefix) it became an industry standard.

Quite a number of IDEs have been created to further improve developers' experience working with Java. Netbeans, Eclipse and IntelliJ come to mine, although there are many others, and of course you can still write Java using a standard (or advanced) text editor such as Notepad or VIM. While the author usually prefers the latter for any kind of text editing — this very document was written entirely using VIM — the weapon of choice when it comes to Java is currently IntelliJ. The way IntelliJ practically writes more than half of the code automatically for you is something no other IDE has been able to match. Naturally, this is the author's personal opinion and should not be seen as fact, but if you're looking for a new Java IDE, it's definitely worth checking out. The prospect of using IntelliJ for this thesis has definitely contributed to the decision of using Java. It should be noted that, had another Java IDE been required, this thesis might never have seen the light of day.

The vast and extensive amount of libraries available for Java was obviously one of the more important reasons to make this choice. The existence of the Apache POI library (see section 5.1) was a huge help in reaching our goal; without it, we would have had to figure out the very obfuscated .ppt file format structure, which undoubtedly would have taken up more time than was available to us. Other libraries like Spring, which allows the programmer to use and reuse components without writing complex systems to instantiate them, further increased our resolve to make Java our primary technology

choice.

However, Java is not the only technology used here. MindXpres is written entirely in HTML5, so any tool that somehow relates to MindXpres sooner or later needs to use HTML5 as well. The widely accepted HTML5 standard makes MindXpres presentations highly portable and runnable on any device with a recent web browser, including smartphones and tablets (Roels and Signer, 2014).

In the following sections we discuss how the various technologies were used to create the *ppt2mxp* tool.

5.1 Taking PowerPoint apart

When converting one file format into another, the first part of the process involves getting the data you need out of the original file. This can be very complicated, as some — usually proprietary — file formats are deliberately designed to discourage this. They obfuscate data, encrypt it, and structure it in illogical and unexpected ways, amongst other techniques. The PowerPoint file format unfortunately is such a format, as Microsoft wouldn't want to risk other companies making software that would work with PowerPoint files. Of course, over the years people have managed to crack the format, enabling the conversion of PowerPoint presentations into other formats, although the conversion does not usually guarantee to yield results that mimic the original version perfectly. Luckily, we don't want a perfect conversion, we want a better one.

We found Apache POI library very helpful in this part of the implementation. The POI[1] Library is a Java library that provides an API to access Microsoft document formats. The most mature (and most popular) part of it is HSSF[2], which is used by Java developers worldwide to access Microsoft Excel spreadsheet data, as well as export data into Excel spreadsheets.

For our purposes, we relied on HSLF[3], which provided us with a full API to access the contents of a PowerPoint presentation's contents in a myriad of ways. We could access all images at once, or every bit of text from the whole presentation, but the most interesting to us was the ability to access contents on a per-slide basis. Getting a list of the slides in a presentation first allowed us to group contents within their immediate context, under a node per slide in our component tree. As such, we could loop over the presentation's slides, converting them one by one, by placing the contents of

[1] Originally "Poor Obfuscation Implementation" (Sundaram, 2004)
[2] Horrible SpreadSheet Format
[3] "Horrible SLideshow Format"

each slide in a MindXpres slide equivalent.

5.1.1 Bullets

That was unfortunately not the end of it. While HSLF does give us access to all the text in a presentation, or per slide, it was not immediately clear to us how it distinguished between 'normal' text and bullet lists. This meant for a long time our conversion process was incomplete, as all bullets from the original PowerPoint presentation appeared as incoherent text runs in our converted result. We found out about the `RichTextRun` class, which had all the tools and properties to detect bullets and their indentation level, but we only discovered very recently that we could extract `RichTextRun`s from the `TextShape`s we were getting out of the slides.

```
1  BulletList ul = new BulletList();
2  Stack<BulletList> listStack = new Stack<>();
3  int indent = 0;
4
5  for (RichTextRun run : textShape.getTextRun().getRichTextRuns()) {
6    ListItem li = new ListItem();
7    StringContent txt = new StringContent(StringUtils.strip(run.getText()));
8
9    if (run.isBullet()) {
10     if (run.getBulletOffset() > indent) {
11
12       indent = run.getBulletOffset();
13       listStack.push(ul);
14       ul = new BulletList();
15       listStack.peek().addItem(ul);
16
17     } else if (run.getBulletOffset() < indent) {
18
19       indent = run.getBulletOffset();
20       ul = listStack.pop();
21
22     }
23   } else {
24     while (listStack.size() > 0) {
25   //  Current component is not a bulletlist or bullet, go to the top bullet level
26       ul = listStack.pop();
27     }
28   }
29
30   li.getContents().add(txt);
31   ul.addItem(li);
32 }
33
34 while (listStack.size() > 0) {
35   // Current component is not a bulletlist or bullet, go to the top bullet level
36   ul = listStack.pop();
37 }
38
39 return ul;
```

Lst. 5.1: Converting bullet points

The code in Listing 5.1 shows how we extract bullet points from a `TextShape` object and convert them into the nested bullet format we need. As you may

notice, the original bullet points are not nested in any way: they are all on the same level of the original object tree, no matter their indentation level. This indentation level is also stored in the bullet object. However, we wanted a more elegant solution where the indentation would be deduced from the level of nesting, much like HTML has always done.

The solution loops over the list of bullets, checking their indentation level and building a stack of nested bullets accordingly. When increasing the indentation level, a new bullet list is started and added as an item on the existing bullet list. When indentation decreases, we go back to the parent list and continue there. This can go down several levels, and all the way back up of course. Since the objects don't have a direct link to their parent, we use a stack to keep track of this at all times.

5.1.2 Animations

Another challenge was dealing with animations and other ways people managed to put way more content on one slide than would be advisable. The animations could not be transferred to MindXpres since MindXpres has its own way of transitioning from each component to the next in the form of a ZUI[4]. It would technically be possible to implement additional animations as a separate plug-in for MindXpres to provide the equivalents of the animations in Microsoft PowerPoint, but that is beyond the scope of this thesis. So we could not provide the same animations, but some people use those animations not just to show off but to actually show multiple pictures and blocks of text, one after the other, on the same slide. Without animations, this content would either not be visible or it would become a serious layout issue in MindXpres.

Our first solution tried to limit the amount of objects one slide can contain, and any additional content should be put on extra slides automatically. A downside of this is that we had no way of guessing the correct order in which the content should appear, so what may have been an intrinsic choreography of pictures in PowerPoint might become an incoherent jumble of images in MindXpres. Another solution would be to scale all content until it all fits next to each other on one slide, and then rely on the ZUI to show the pictures one by one, but in this case the same problem with order of appearance manifests itself. In the end, we decided it would be best to accept that no conversion algorithm is going to be perfect, and the author can always manually change the order around after the conversion is done.

With this in mind, we now render the components in the order we get

[4] Zoomable User Interface

them from HSLF, hoping that this resembles the original order closely. The automated layout takes care of any overlapping that might have occurred originally, so we don't have to worry about that.

5.2 Generating MindXpres

Generating MindXpres presentations was the final goal of the first phase of this thesis. This seemed a fairly easy task at first, until we learned that the MindXpres compiler would not be available to us for most of the year. This meant we would either not be able to view-test our generated presentations, or we would have to convert them to browser-ready HTML5 ourselves. We chose the latter option, as not being able to see our results would not be very helpful in implementing and tweaking our conversion tool. As a result, this task became much more complicated, as we had to emulate the compiler's work ourselves. Luckily we already decided we would be working with a Java object representation of the original presentation as an intermediary form, a so-called component tree, which meant we could easily change the output of our conversion tool without affecting the rest of the conversion process, and on a per-component basis.

5.2.1 Playing MindXpres compiler

Since the MindXpres compiler was not functional during most of this thesis' implementation, we decided to generate an HTML5 file much like the MindXpres compiler would, including the MindXpres JavaScript library and plug-ins. This required us to first learn how MindXpres works on the inside, which proved to be a steep learning curve but gave us more insight into the software than we would've gotten if we only had to generate MindXpres XML and leave the rest to the compiler.

Improving the ZUI

As an exercise, we changed the way the ZUI works. Originally, MindXpres used the CSS3 `transform: scale()` property to enlarge or reduce the whole view, giving the impression of zooming in or out. This is an obvious approach, simple in its execution and quite foolproof. However, the downside is that you can't zoom in very much, because currently browsers do not leverage the advantage of vector graphics and fonts even if you do use them, and obviously raster-based content doesn't scale much anyway. Instead, browsers render the content at its initial scale, and then treat the result as one big image when

scaled or otherwise transformed afterwards. This means you get extremely pixelated content when zooming in too much.

Through some refactoring, we were able to change this to use the `transform: translateZ()` property instead, along with the `transform: perspective()`[5] and the `transform-style: preserve-3d` properties. This means we're now effectively rendering the presentation in 3D, and moving our viewpoint around in the 3D space to center each slide or component in turn.

We believe this opens the door for even more visually impressive presentations, where content can be placed on different points along the Z-axis. This allows for example to place multiple slides behind one another, making for impressive zoom transitions between slides. The downside of this is that the overview may not always show all content, as some content can overlap, but we trust the author uses this feature wisely when manually adjusting the position of their slides. It may for example be useful to group slides together in this way, when there is too much content to show on one slide but creating a second, separate slide may break the flow of the presentation. In any case, our automated layout plugin won't currently generate slides positioned this way.

Plain HTML5

After investigating the inner workings of MindXpres and studying some example presentations, we were ready to start generating our own presentations based on our component tree. This meant every possible component would have to be written out as valid HTML, with the necessary attributes for each generated tag and with any child components enclosed. Since our component tree nodes are nested the way the final HTML should be nested, this was not a problem.

Our implementation currently includes `compile()` methods on every component object, which is consistent and easy to understand, but which might not be the best way to implement this depending on future goals. Instead of having a separate layer taking care of the output, currently it is a cross-cutting concern, which as we all know is not a desirable design pattern. We have to walk through the entire tree in any case, so performance will always be $O(n)$ at best.

The current implementation has the advantage of extensibility, where new components can easily be added and it is immediately obvious to any new developer how these new components should generate their equivalent HTML code. However, for replacing the output with a different format it

[5] Not to be confused with the `perspective: <number>` property, which yields different results

would be better if this functionality was separated from the components and gathered in a distinct `Writer` class instead. Switching formats would then be as easy as dropping in a new `Writer` class that generates a different format. Since we initially did not expect to switch outputs, and because we started implementing the conversion of one component and then added components as we needed them, the current implementation — focused on simplifying addition of new components — was easier for us to work with. Perhaps the refactoring of this implementation is an option for future work.

MindXpres XML

Generating MindXpres XML should be simpler than the HTML5 output, although in the end there is probably not much difference. We won't have to generate unique ID's for every component, and we won't have to generate the preamble content (which includes the MindXpres library itself), but the structure and content should remain pretty much the same. Instead of `<div>` tags we can now use MindXpres-specific tags. All that adds up to a much more easily readable result.

The advantages towards the user would be even bigger. Using a hypothetical MindXpres editor, the output of our conversion tool could be edited in such a tool immediately, while presumably such an editor would not work on plain HTML5. The reuse of content — one of the main goals of MindXpres— would also be much more plausible using the XML format.

5.3 Creating layouts

Implementing an automated layout is not an easy task. At any point in the process opportunities arise to use some kind of template, some sort of design choice that would appear to make things easier, but turn out to be restrictive when applied to edge cases. There's also not always a clear distinction between implementing a template and implementing an automated layout. If you decide to put all of your components in a row next to each other, have you just implemented a template or not? What if you make them all the same height, so the row will look aesthetically pleasing when looking at it as a whole? What if you don't?

It becomes more clear when we have to work within a defined area, such as a slide container. In this case, we have to fit our content within the slide; this could be seen as a template decision but it is not one our algorithm makes, so the algorithm itself just tries to fullfill the constraints it is given. We can then calculate the relative sizes of our components, scale them down or up together (using the same scale factor) until their combined size equals

the area we need to fill, and puzzle them together in a way that fits. If no way is possible, we scale everything down some more and try again, until we find something that works.

Whether using automated layout or not, the suggestion to not put too much content on one slide remains. While an automated layout system may be able to fit all content on one slide, too much content will still cause information to be conveyed less effectively. Due to technical limitations of currently existing browsers, scaling content down and then zooming in to make it fill the screen is not always an option: browsers treat the content as rasterized images rather than vectorized graphics, and scaling them does not yield ideal results. Scaling an image down and then zooming in on it will show you a blurred version at best, and a great big coloured blob at worst. Using the `perspective()` and `translateZ()` properties yields much better results, but this isn't easily usable within a slide container as the content would be rendered *behind* the slide, making the slide seem empty as seen from the front. If we want the content to be visible on the slide, we need to render it within the size limits of the slide, which gives the same blurred results as scaling.

The true power of automated layout generation becomes apparent when no such boundaries are posed. If the content does not need to be scaled down, we can use it all in its original size whether those sizes are similar or not. Depending on which options the author of the presentation turns on, we can then still resize content to match sizes, among other things.

5.3.1 Using constraints

The basis of our automated layout approach is the use of constraints. As a first step in the process, we assign each component a certain 'bounding box', a set of constraints that dictates how close other components can get to this particular component. This distance can be decided arbitrarily, or have a hard-coded value, but we decided to take a more dynamic approach and let the distance be 10% of the width or height, whichever is the largest. This way, the padding around the content is equal on all sides, and proportionate to the size of the content.

The next step in the process decides the size of the components. There are two mechanisms that may be applied here together, separately or not at all. One mechanism is the equalization of sizes, where we resize all components in such a manner that they end up all being the same or similar sizes. We can decide to apply this either to the width, the height or the surface area, depending on the effect we hope to create. If we want to put all components size by side, making them all the same height might make for an aesthetically

pleasing result, for example; if we want to group them together in a raster-like manner similar surface areas would usually be a better idea.

The other mechanism is the scaling of components, in which we scale everything up or down equally, maintaining relative proportions between components while reducing or increasing the total used surface area. This is especially useful when we have to generate an automated layout that needs to fit within a predefined container with a fixed size, such as a slide container.

The default approach when no such fixed-size container is present, is to not apply either of these mechanisms, while we apply both mechanisms together when we do have to fill this predefined area: we then calculate an average surface area size between all components, resize all components to match this average, and finally scale everything up or down so that the total area fits the container.

It is possible to override this behaviour: when no fixed size is defined, we can specify that all components should be sized equally, be it in width, height or surface area. The latter is the default but this too can be overridden: specifying that the components should be placed in a row or a column instead of a raster-like structure will automatically choose the matching equalizing method. Conversely, when there is a fixed size, we may specify to retain the original proportions and only scale everything up or down equally to fit the area.

5.3.2 Other ways

During implementation, we tried several other ways to improve upon the automated layout, all of which we ultimately decided to abandon in favor of the current approach. There were various reasons for abandoning these paths. Often the reason was that we realized we were influencing the automated layout process with human design decisions, which is exactly what we were trying to avoid. Sometimes what we were trying did not yield the results we hoped for, and sometimes the implementation became too complex to continue or what seemed like a good idea in theory turned our to be impossible in practice.

One such idea was to divide components in a raster-like structure, but spread over a set of rows according to a normal distribution — most components in the middle rows, a few components on the first and last few rows. We realized soon that on one side, this would become a kind of template (a very dynamic one, but still), and on the other side, this was a lot more complex than we initially thought. While this feature has been abandoned in the current implementation we do think it may still be an option for future work though: although it is a template, it seems dynamic enough that it

would still match the spirit of our automated layout approach.

Another idea was to use the Z-axis to make components seem equal in size when viewed from a certain angle as an overview, then zooming in on each component to reveal their true size. This was abandoned purely for complexity reasons: it is not enough to just place larger components further back, you also need to adjust the x and y coordinated to make it seem like the component is placed right next to another when in reality it is a lot further back. This would not be necessary with an orthogonal camera mode, but the HTML5 rendering engines only have a perspective camera mode, which makes this idea too complex to implement within our limited timeframe.

6. CONCLUSIONS AND FUTURE WORK

Microsoft PowerPoint remains the most popular presentation tool worldwide. We believe this is mostly because people are generally afraid of change, and would rather stick to their habits. Since all of their existing work is stored in the PowerPoint format, they keep using PowerPoint to access that content as well as create new presentations. On top of that, the process of creating a presentation is still heavily burdened by the layout of the content we want to present. Layout is an important and vastly underrated aspect of presentations in general, which uses up an astonishing amount of time during the creation of a presentation.

6.1 Contribution

We proposed an approach for converting existing PowerPoint presentations into MindXpres presentations, along with a way to take control over layout away from the author and improve upon flawed human design by programmatically calculating ideal content placement and size. We delivered a proof of concept implementation that puts this approach into practice, first letting us show a PowerPoint presentation's content in MindXpres, then showing us the possibility of applying an automated layout algorithm to that an any other MindXpres content at will.

Considering the first part of this thesis, which consists of the conversion between PowerPoint and MindXpres, we can conclude that conversion from any other presentation format into MindXpres is a feasible concept. Closed-source formats will obviously be more of a challenge than their open-sourced cousins, especially if no API has been created for them as we had the fortune with PowerPoint and Apache's POI/HSLF implementation. That said, open-source formats may be more easy to take apart but if no API exists for them it would still require a substantial amount of effort. Having an existing API readily available has definitely helped us a great deal in our efforts.

Converting these other formats into MindXpres remains an important goal in the endeavour to raise awareness and increase popularity of MindXpres's features and possibilities. Our implementation provides a way to convert PowerPoint.ppt presentations, but Microsoft PowerPoint has switched to

using the Office Open XML-based .pptx format in recent years, so newer PowerPoint presentations cannot currently be imported into the MindXpres system.

Our implementation is written in a way that should make it straightforward to adapt for other formats, provided there is a way to get the separate components out of those formats. If slide-based conversion is desired, then obviously a way to extract the components on a per-slide basis is also required. Additionally, our approach using an intermediary form during the conversion process allows for adaptation of the tool to generate other output formats as well. The current implementation generates HTML5 which includes the MindXpres standard library and a set of plugins, but the original goal of generating MindXpres XML files should be easily attainable; the only reason we did not implement this was the unavailability of the MindXpres compiler, which made it impossible for us to test the generated XML files.

As for the second part of this thesis, concerning the automated generation of presentation layouts, we have discussed why this is necessary. When creating traditional slideware, as well as using more advanced and modern presentation tools, layout remains a problem that for many presenters becomes the biggest timesink in their work. On top of that, the layout they create isn't always a good one, and bad layout has been proven to have negative impact on the effectiveness of a presentation. As such, having a way to automatically generate a layout would save a lot of time while also improving the information transfer effectiveness of presentations.

We have demonstrated such an automated layout mechanism based on theory and research found in related works, which we adapted and improved upon for our purposes. Our constraint-based approach considers every component separately, to combine all components into a layout where no overlapping exists, components can be grouped together, clear margins are put in between content and surrounding limits in the form of slides and other fixed-size containers are respected.

The implementation of this mechanism is far from complete, and may still be improved upon in several ways, which will be discussed in section 6.2. It does however provide the most basic form of automated layout, which may not always succeed in generating an aesthetically pleasing layout but at least attempts to combine content in a way that makes the content easy to focus on, thus increasing effectiveness of the presentation. It also succeeds in letting the presenter focus on the content rather than the layout. It thus reduces stress and arguably increases quality of presentations, especially when we look at time spent creating the presentation versus its effectiveness.

6.2 Future Work

In this thesis we have presented a proof-of-concept implementation of both a tool to convert PowerPoint presentations into MindXpres, and an algorithm for generating an objectively effective layout. Due to the limited time available for this thesis, we were not able to go into the finer details of these tools, and the result can seem rather unpolished. However, within this limited timeframe we did deliver a solid core containing the most important features, in a way that allows future research to improve upon it and easily add any missing details.

6.2.1 Other formats

Our conversion tool currently allows to convert PowerPoint.ppt files into MindXpres presentations, bypassing the MindXpres compiler. The tool internally uses an intermediary structure to store the presentation's content, and this facilitates the implementation of conversion tools for other formats. As such, it might be a good idea to extend the tool to convert other popular formats like PowerPoint.pptx files, Apple Keynote presentations and many others.

It would also be a good idea to change the output of the conversion tool to generate MindXpres XML files to compile further using the MindXpres compiler. In and of itself this would not seem advantageous, but with MindXpres IDE's and other editing tools in mind it would be better to have XML files which would be editable using those tools, rather than raw HTML5 which presumably would not be readily available in any IDE.

6.2.2 Integration

Speaking of editors, it would be interesting to integrate the conversion tool into such an editor. This would allow MindXpres users to just open their PowerPoint files in the MindXpres editor, immediately providing access to its contents and letting the user edit the presentation as if it had always been an MindXpres presentation. This would greatly improve usability of the conversion tool as well, since it currently does not have a graphical user interface and thus needs to be invoked from the command line.

6.2.3 Improving the automated layout

There are many ways in which the automated layout algorithm may yet be improved. Jock Mackinlay's work (Mackinlay, 1986) includes significant research on how to use artificial intelligence to create effective graphical visual-

isations. Combining his work with ours could potentially improve the results of our algorithm. An interesting angle here might be the use of a learning AI, which can be trained on sets of good and bad layouts, or observe the user's actions and try to mimic their behaviour.

BIBLIOGRAPHY

I. Parker, "Absolute PowerPoint: Can a software package edit our thoughts?" *The New Yorker*, pp. 76–87, May 2001. [Online]. Available: http://www.newyorker.com/archive/2001/05/28/010528fa_fact_parker?currentPage=all

S. M. Drucker, G. Petschnigg, and M. Agrawala, "Comparing and managing multiple versions of slide presentations," in *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '06. New York, NY, USA: ACM, 2006, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/1166253.1166263

G. Bajaj. (2013, Feb.) 30 million PowerPoint presentations? [Online]. Available: http://blog.indezine.com/2013/02/30-million-powerpoint-presentations.html

E. R. Tufte. (2005, Sep.) PowerPoint does rocket science — and better techniques for technical reports. [Online]. Available: http://cgvr.cs.uni-bremen.de/miscellaneous/presentation_tips/edward_tufte/ppt_does_rocket_science.shtml

R. Roels and B. Signer, "Mindxpres: An extensible content-driven cross-media presentation platform," in *Web Information Systems Engineering — WISE 2014*, ser. Information Systems and Applications, incl. Internet/Web, and HCI, vol. 8787. Springer International Publishing, 2014, pp. 215–230. [Online]. Available: http://link.springer.com/book/10.1007%2F978-3-319-11746-1

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

D. K. Farkas, "Toward a better understanding of PowerPoint deck design," *Information Design Journal*, vol. 14, no. 2, pp. 162–171, 2006.

T. H. Nelson, "The heart of connection: Hypermedia unified by transclusion," *Commun. ACM*, vol. 38, no. 8, pp. 31–33, Aug. 1995. [Online]. Available: http://doi.acm.org/10.1145/208344.208353

A. Holzinger, M. Kickmeier-Rust, and D. Albert, "Dynamic media in computer science education; content complexity and learning performance: Is less more?" *Journal of Educational Technology & Society*, vol. 11, no. 1, pp. 279–290, Jan. 2008.

S. Lok and S. Feiner, "A survey of automated layout techniques for information presentations," 2001.

C. Adams, "PowerPoint, habits of mind, and classroom culture," *Journal of Curriculum Studies*, vol. 38, no. 4, pp. 389–411, 2006.

A. Gross and J. Harmon, "The structure of PowerPoint presentations: The art of grasping things whole," *IEEE Transactions on Professional Communication*, vol. 52, no. 2, pp. 121–137, Jun. 2009.

L. Good and B. B. Bederson, "Zoomable user interfaces as a medium for slide show presentations," *Information Visualization*, vol. 1, no. 1, pp. 35–49, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1057/palgrave/ivs/9500004

L. Lichtschlag, T. Karrer, and J. Borchers, "Fly: A tool to author planar presentations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 547–556. [Online]. Available: http://doi.acm.org/10.1145/1518701.1518786

J. Lanir, K. S. Booth, and A. Tang, "Multipresenter: A presentation system for (very) large display surfaces," in *Proceedings of the 16th ACM International Conference on Multimedia*, ser. MM '08. New York, NY, USA: ACM, 2008, pp. 519–528. [Online]. Available: http://doi.acm.org/10.1145/1459359.1459428

J.-C. Chen, W.-T. Chu, J.-H. Kuo, C.-Y. Weng, and J.-L. Wu, "Tiling slideshow," in *Proceedings of the 14th Annual ACM International Conference on Multimedia*, ser. MULTIMEDIA '06. New York, NY, USA: ACM, 2006, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/1180639.1180653

B. Signer and M. C. Norrie, "Paperpoint: A paper-based presentation and interactive paper prototyping tool," in *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, ser. TEI '07. New York, NY, USA: ACM, 2007, pp. 57–64. [Online]. Available: http://doi.acm.org/10.1145/1226969.1226981

L. Nelson, S. Ichimura, E. R. Pedersen, and L. Adams, "Palette: A paper interface for giving presentations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '99. New York, NY, USA: ACM, 1999, pp. 354–361. [Online]. Available: http://doi.acm.org/10.1145/302979.303109

R. Spicer, Y.-R. Lin, A. Kelliher, and H. Sundaram, "Nextslideplease: Authoring and delivering agile multimedia presentations," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 4, pp. 53:1–53:20, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2379790.2379795

D. Edge, J. Savage, and K. Yatani, "Hyperslides: Dynamic presentation prototyping," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: ACM, 2013, pp. 671–680. [Online]. Available: http://doi.acm.org/10.1145/2470654.2470749

P. Garcia, "Retooling PowerPoint for hypermedia authoring," in *Proceedings of Society for Information Technology & Teacher Education International Conference 2004*, R. Ferdig, C. Crawford, R. Carlsen, N. Davis, J. Price, R. Weber, and D. A. Willis, Eds. Atlanta, GA, USA: Association for the Advancement of Computing in Education (AACE), 2004, pp. 4098–4099. [Online]. Available: http://www.editlib.org/p/14625

D. Austin, "Beginnings of PowerPoint: A personal technical story," 2009. [Online]. Available: http://www.computerhistory.org/collections/catalog/102745695

B. Parks, "Death to PowerPoint," 2012.

C. Sü and B. Freitag, "LMML — the learning material markup language framework," in *Proceedings of Workshop ICL*, Sep. 2002.

J. Fisler and S. Bleisch, "eLML, the elesson markup language: Developing sustainable e-learning content using an open source XML framework," in *WEBIST 2006 — International Conference on Web Information Systems and Technologies*, Apr. 2006.

J. H. Canós-Cerdá, M. I. Marante, and M. Llavador, "SliDL: A slide digital library supporting content reuse in presentations," in *ECDL 2010*, Sep. 2010.

K. Verbert, X. Ochoa, and E. Duval, "The ALOCOM framework: Towards scalable content reuse," *Journal of Digital Information*, vol. 9, no. 1, pp. 1–24, Jan. 2008.

D. Raggett, "Slidy — a web based alternative to Microsoft PowerPoint," in *XTech 2006*, 2006.

M. D. Bush and J. D. Mott, "The transformation of learning with technology: Learner-centricity, content and tool malleability, and network effects," *Educational Technology*, vol. 49, no. 2, pp. 3–20, Mar. 2009.

H. Haller and A. Abecker, "imapping: A zooming user interface approach for personal and semantic knowledge management," *SIGWEB Newsl.*, no. Autumn, pp. 4:1–4:10, Sep. 2010. [Online]. Available: http://doi.acm.org/10.1145/1850770.1836295

R. J. Dufresne, W. J. Gerace, W. J. Leonard, J. P. Mestre, and L. Wenk, "Classtalk: A classroom communication system for active learning," *Journal of Computing in Higher Education*, vol. 7, no. 2, pp. 3–47, 1996.

B. Signer and M. C. Norrie, "Interactive paper: Past, present and future," in *PaperComp 2010*, Sep. 2010.

E. Reuss, B. Signer, and M. C. Norrie, "PowerPoint multimedia presentations in computer science education: What do users need?" in *USAB 2008*, 2008.

R. Roels, C. Vermeylen, and B. Signer, "A unified communication platform for enriching and enhancing presentations with active learning components," in *ICALT 2014*, Jul. 2014, pp. 131–135.

B. Signer and M. C. Norrie, "As we may link: A general metamodel for hypermedia systems," in *Proceedings of the 26th International Conference on Conceptual Modeling*, ser. ER'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 359–374. [Online]. Available: http://dl.acm.org/citation.cfm?id=1784489.1784522

N. Hurst, W. Li, and K. Marriott, "Review of automatic document formatting," in *Proceedings of the 9th ACM Symposium on Document Engineering*, ser. DocEng '09. New York, NY, USA: ACM, 2009, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1600193.1600217

G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

G. Di Battista, W.-P. Liu, and I. Rival, "Bipartite graphs, upward drawings, and planarity," *Inf. Process. Lett.*, vol. 36, no. 6, pp. 317–322, Dec. 1990. [Online]. Available: http://dx.doi.org/10.1016/0020-0190(90)90045-Y

F. Shahrokhi, L. A. Szkely, O. Skora, and I. Vrt'o, "Drawings of graphs on surfaces with few crossings," *Algorithmica*, vol. 16, no. 1, pp. 118–131, 1996, algorithmica, 16(1):118131, July 1996. [Online]. Available: http://dx.doi.org/10.1007/BF02086611

T. C. Hu and E. S. Kuh, "VLSI circuit layout: Theory and design," 1985, IEEE, USA, 1985.

T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout.* New York, NY, USA: John Wiley & Sons, Inc., 1990.

V. J. Milenkovic, K. M. Daniels, and Z. Li, "Automatic marker making," 1991, in T. Shermer, editor, Proceedings of the Third Canadian Conference on Computational Geometry, pages 243246, August 610 1991.

M. Hofri, "Two-dimensional packing: Expected performance of simple level algorithms," *Information and Control, 45:117, 1980.*, no. 45, pp. 1–17, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995880908177

J. Scahill and J. Begley. (2015, Mar.) The CIA campaign to steal Apple's secrets. [Online]. Available: https://firstlook.org/theintercept/2015/03/10/ispy-cia-campaign-steal-apples-secrets

J. Gosling and H. McGilton, "The Java language environment," 1995. [Online]. Available: http://www.stroustrup.com/1995_Java_whitepaper.pdf

E. Sundaram, "Excelling in excel with Java," 2004.

J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110–141, Apr. 1986. [Online]. Available: http://doi.acm.org/10.1145/22949.22950