RWTH AACHEN UNIVERSITY

Semantic Navigation in Digital Media

PhD Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Thorsten Karrer

Semantic Navigation in Digital Media

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorglegt von

Diplom-Informatiker Thorsten Karrer

aus Hamburg, Deutschland

Berichter: Prof. Dr. Jan Borchers
Prof. Dr. Michael Rohs

Tag der mündlichen Prüfung: 24. Mai 2013

Contents

List of Figures

List of Tables

Abstract

With the advent of ubiquitous computing devices and their associated capabilities for capturing photos, videos, audio, and text, user-generated media content and its large-scale distribution have become a reality. While then years ago the production and dissemination of most digital media was practically restricted to commercial content providers and a few enthusiasts, nowadays, most people have the means to create, publish, and consume high-quality media. What is interesting, however, is that despite this shift in paradigm, one of the most fundamental underlying mechanisms—navigation in digital media—has hardly changed at all and is in many cases still cumbersome and difficult.

Users' navigation goals are mostly formulated in terms of the actual contents of a medium; finding the point in a movie where two story lines finally meet or accessing the part of a text where a certain argument is being made are useful and clearly defined goals. Navigation interfaces, however, are still formulated in terms of the generalizable form of a medium; they allow users to move to certain timestamps or page numbers. This means that the goal of a navigation task is rarely concerned with the functional entities that current navigation techniques operate on. As a result, the users have to provide the capability to convert between their semantic navigation goals that refer to the content and a syntactic expression of that goal that is compatible with the interface language. This conversion, of course, is non-trivial and usually has to be found out by the users through trial-and-error or exhaustive search.

One of the obvious reasons for this mismatch of languages between semantic user intent and syntactic navigation interface is the analog heritage of the latter: Interfaces to control and navigate analog media do not have access to the contents of these media and, consequently, have to operate on the form as the common structure. Computer interfaces for digital media, on the other hand, could have the ability to inspect the content and allow users to formulate in semantic terms. Still, most of these interfaces are directly modeled after their analog counterparts without leveraging this potential of the medium's digital representation.

In this thesis, we present an interaction model for navigation in digital media that represents the interface together with the form and content of the medium. This combination of structural representations of the interface and the medium is novel. It gives us a holistic framework in which we can describe existing navigation interfaces, analyze their shortcomings for semantic navigation goals, and guide the creation of semantic navigation interfaces that allow to directly express such goals. In addition, we propose a simple four-step design guideline that builds upon this framework and helps interface designers to create semantic navigation interfaces.

The viability of both the interaction model and the design guideline is demonstrated in the context of four research projects that each create and evaluate a semantic navigation interface for a different type of medium. The interfaces and navigation techniques that have emanated from these projects have already been shown to outperform conventional methods as well in measured efficiency as in perceived ease of use and subjective preference. While each thus is a contribution in itself to the respective field, their main purpose in the context of this work is to each exemplify and illustrate one specific step of our four-step guideline in detail.

Überblick

Mediale Inhalte, die von Endnutzern erzeugt und bereitgestellt werden, sind ein fester Bestandteil der Informationslandschaft geworden. Dies ist spätestens seit der weitreichenden Verfügbarkeit von digitalen Aufnahmegeräten für Bilder, Videos, Ton und Text der Fall. War die Erstellung und Verbreitung von digitalen Medien bis vor zehn Jahren noch kommerziellen Anbietern und einigen wenigen semiprofessionellen Endanwendern vorbehalten, so können heute nahezu alle Benutzer Medien in hoher Qualität produzieren, verteilen und konsumieren. Trotz dieses Paradigmenwechsels sind einige der grundlegenden Interaktionsmechanismen – insbesondere die Navigation innerhalb digitaler Medien – weitgehend unverändert geblieben und damit nach wie vor unhandlich und kompliziert.

Die Navigationsziele von Nutzern beziehen sich üblicherweise auf den tatsächlichen Inhalt eines Mediums. An genau die Stelle eines Films zu springen, an der zwei Handlungsstränge zusammenlaufen, oder den Teil eines Textes zu finden, in dem ein bestimmter Sachverhalt erklärt wird, sind sinnvolle und klar definierte Ziele für Mediennavigation. Die Interaktion mit gegenwärtigen Benutzerschnittstellen bezieht sich jedoch immer noch lediglich auf die gemeinsame Form eines Medientyps. Benutzer können damit beispielsweise zu Zeitstempeln oder Seitenzahlen navigieren. Somit stehen die Navigationsziele der Nutzer und die Funktionsangebote der Benutzerschnittstellen auf unterschiedlichen konzeptionellen Ebenen. Dies hat zur Folge, dass die Benutzer selber ihre semantischen Navigationsziele, die sich auf den Inhalt beziehen, in syntaktische Formulierungen, die mit den Ausdrucksmöglichkeiten der Schnittstelle kompatibel sind, umwandeln müssen. Diese Umwandlung ist weder intuitiv noch einfach und läuft daher oft auf eine erschöpfende Suche im betreffenden Medium hinaus.

Ein Grund für diese vorherrschende Diskrepanz zwischen den Sprachen, in denen sich Nutzerabsicht einerseits und Interaktionsmöglichkeiten andererseits ausdrücken, ist die historische Entwicklung digitaler Navigationsschnittstellen als Nachfolger analoger Aufnahme- und Abspielgeräte. Letztere haben

traditionell keine Möglichkeit, das Medium inhaltlich aufzuschlüsseln, und können daher gezwungenermaßen nur Eigenschaften des Medienträgers beeinflussen. Diese Einschränkungen gelten für computergestützte Mediennavigation oft nicht; hier könnte der Medieninhalt durchaus analysiert werden, um semantisch formulierte Navigationsziele zu verstehen. Dennoch sind die meisten digitalen Navigationsschnittstellen immer noch direkte Abbilder ihrer analogen Vorfahren.

Gegenstand dieser Dissertation ist die Erstellung eines Interaktionsmodells für die Navigation in digitalen Medien, welches sowohl die Benutzerschnittstelle als auch Form und Inhalt des Mediums zusammen darstellt. Diese Kombination von strukturellen Repräsentationen des Mediums einerseits und der Benutzerschnittstelle andererseits ist neuartig: Sie ermöglicht eine einheitliche Beschreibung existierender Navigationstechniken und die Bewertung dieser Techniken im Kontext semantischer Navigation. Darüber hinaus lassen sich aus dem Modell Empfehlungen für das Design von semantischen Navigationsschnittstellen ableiten. Ein weiteres Ergebnis dieser Arbeit ist somit eine derartige modellgestützte vierschrittige Anleitung.

Die Anwendbarkeit des Modells und der Designempfehlungen wird im Rahmen von vier Forschungsprojekten demonstriert, in denen jeweils eine semantische Navigationsmethode für einen anderen Medientyp entwickelt wurde. Die Benutzerschnittstellen, die aus diesen Projekten hervorgegangen sind, sind nachweislich effizienter und einfacher in der Bedienung als konventionelle Techniken zur Mediennavigation, und werden darüber hinaus von den Benutzern präferiert. Obwohl jedes dieser Projekte somit einen eigenen Beitrag zu dem jeweiligen Forschungsfeld darstellt, steht in dieser Dissertation der illustrative Aspekt dieser Teilarbeiten im Rahmen des Modells und der Designempfehlungen im Vordergrund.

Acknowledgements

My first and foremost thanks go to my advisor, Jan Borchers, who would always be ready to give advice or encouragement—whatever was needed most at the moment. More importantly even, he created the best working environment an aspiring PhD student could hope for; his group is a place of great individual freedom, noncompeting collaboration, and unquestioning support for interesting ideas. I have loved working at i10—maybe that is one of the main reasons for staying here so long—, and this is for a great part due to the way Jan choses to head this group.

I also want to thank Michael Rohs, who was kind enough to offer me a position in his newly formed group and, although I had to decline, who still agreed to act as my second advisor on this thesis. Reviewing such a large document while at the same time attending to the many duties and administrative acts that come with building up a new research group cannot be taken for granted, and I very much appreciate Michael's commitment to do this for me.

The four projects that form the backbone of this thesis would not have been realizable at this scale if it had not been for the help of my thesis students, colleagues, and co-authors. The idea for DRAGON could only be turned into software because Malte Weiß offered his extensive knowledge on optical flow and his coding skills to create the first flow field tracker. Moritz Wittenhagen's help in the later DRAGON projects, first as a student and later as a colleague, was invaluable, and I will sincerely miss fighting with him over variable naming. Regarding FLY, I have to thank Leonhard Lichtschlag for his commitment to this project. He wrote the first software prototype as part of his diploma thesis work, and he has since long become the driving force behind FLY. The same thing is true for Jan-Peter Krämer and STACKSPLORER; without his help in running the experiments and analyzing the data, our tools for efficient call graph navigation would still be ideas only. Thanks also to all thesis students that have contributed to this research, especially Alisa Novosad, Anne Kathrein, Andreas Nett, Christian Brockly, Joachim Kurz, Thomas Heß, and Xiaojun Ying.

Apart from the research that made it into this thesis, I worked on other projects that were all either insightful, inspiring, necessary for funding, fun, challenging, welcomely distracting from my thesis, or any combination thereof. I would like to thank the people whom I could collaborate with in these projects (in no particular order): Moritz Wittenhagen, Leonhard Lichtschlag, Benjamin Walther-Franks, Florian Heller, Till Quadflieg, Gero Herkenrath, Yvonne Jansen, Sarah Mennicken, Chatchavan Wacharamanotham, Jonathan Diehl, Jan-Peter Krämer, and Max Möllers. Working together with you has been an honor and a pleasure.

Thank you all!

Conventions

Throughout this thesis we use the following conventions.

In cases where material was published both as a thesis under the guidance of this author and as a peer-reviewed paper at a conference, the latter will be given preference in terms of citation, because it is easier to get access to for the reader.

Text conventions

Definitions of technical terms or short excursus are set off in colored boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Chapter 1

Introduction

The primary use of digital devices has changed.

If we, as computer scientists, look today at what people use their computers and other digital devices like smartphones and tablets for, we might be surprised. What used to be the tools of our trade—and, as such, primarily intended to help us do our work—has become a part of people's everyday life. These devices are now being used not only for managing databases, keeping stock of large inventories, or solving large numerical problems for simulations; their primary application has shifted towards non-work activities like communicating with friends and family, listening to music, enjoying a movie, or browsing through the latest gossip on various social networks.

People use computers to consume and create digital media.

Nowadays, people use computers to consume digital media. What may be even more important is that in the last few years one of the biggest promises of the computer revolution is finally starting to be fulfilled: people now have the ability to *create* digital media; they can share self-created music, videos, and photos on YouTube or flickr, they can share their opinion through web sites or social micro-blogging platforms, and they can do it at a level of production quality that would have been recognized as professional not many years ago. This is a stark contrast to how media used to be created, distributed, and consumed when they were still represented by their traditional analog forms and were fast in the hands of professional authors, producers, and publishers.

Interaction with digital media resembles interaction with analog media.

While some of the media we use today have actually evolved in a computerized environment—hypermedia come to mind—most of the media we consume and create on our digital devices are still just digital representations of these traditional analog media: audio, video, text, images, animations, and presentation visuals are just some examples. There is nothing wrong with that; these types of media have informed, entertained, and served us well, and there is no reason not to keep them around. Many of them, however, have inherited some of the limitations of their analog ancestors that we have taken as granted for so long that we often do not realize how much we restrict ourselves when dealing with them.

Navigation in media has not changed much between analog and digital.

One particular area where this is true is when we are navigating those media: however small the resemblance between buying a music cassette at a record store and subscribing to a channel at a service like last.fm[1]—getting to a location inside an audio or video track interestingly still requires operating the same transport controls that have been present on tape recorders since 1935 when AEG invented the Magnetophon K1 (cf. Figure 1.1).

Admittedly, some additional elements have been added to most interfaces like, e.g., timeline sliders or scrubbing interfaces for time-based media. However, they have only brought back capabilities that the original analog counterpart already possessed by directly moving the physical object the medium was stored on.

Our goal is to propose a theory for navigation in digital media.

Over the years, of course, a large body of research has been conducted to create interfaces that leverage the advantage of digital representations of media to facilitate new and better forms of navigation (examples include [Ahlberg and Shneiderman, 1994; Hürst and Jarvers, 2005; Lee, 2007; Goldman et al., 2008]), but no overarching theory and concept of why this is difficult and how navigation in digital media should be designed has been defined. It is the goal of this thesis to propose such a theory, together with an *interaction model* [Beaudouin-Lafon, 2000] and a set of design guidelines. These are intended to send a theoretical foundation for creating digital media navigation interfaces that allow users to browse, search, and move through these media

[1] http://www.last.fm

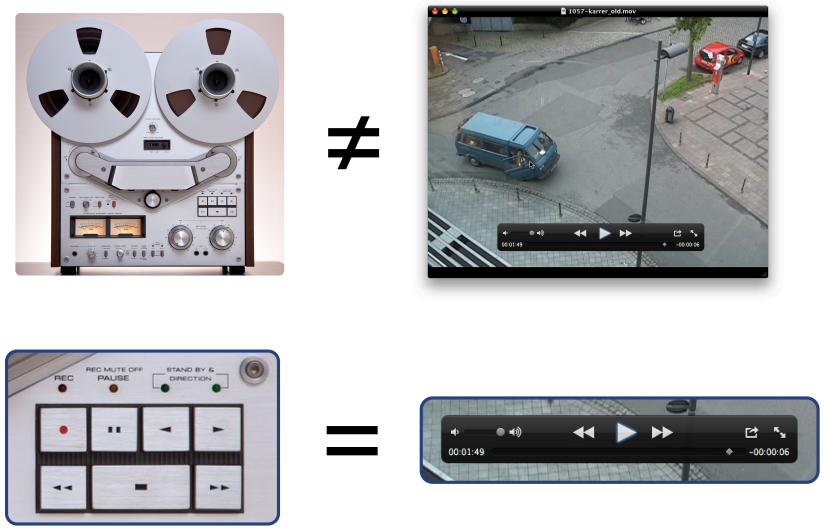

Figure 1.1: Although the technical background has changed considerably, modern media player navigation interfaces are conceptually similar to traditional navigation interfaces for analog media.

in a faster, easier, and more streamlined fashion than those interfaces that follow the dominant design of the media's analog counterparts could allow.

Our theory is based on the duality of syntax and semantics in digital media and direct manipulation interaction.

The central idea in this context is to regard digital media as dual structures that consist of syntactic and semantic spaces, where the former are defined by the type of the medium and the latter by its content and the navigation task. Using this framework, we can set out to identify along which syntactic or semantic subspaces existing navigation techniques operate and along which—usually different—subspaces possible navigation goals of users are collocated. We will demonstrate that it is possible to create navigation interfaces and techniques that allow access to those semantic subspaces that are directly aligned with the navigation goals of users for certain tasks. Additionally, if these subspaces can be represented in a linear or planar euclidean domain, we can apply the concepts of direct manipulation [Shneiderman, 1982] to our navigation interface to gain the benefits of orientation towards the goal and rapid reversibility of actions. The interfaces created this way turn out to be more efficient and enjoyable to use, and they can

offer creative interaction with digital media that would otherwise not be possible.

1.1 Structure

In the following section, 2.1 "Background and Related Work", we will first describe in more detail the idea of media as dual representations over syntactic and semantic spaces and give a brief overview over the concepts and advantages of direct manipulation techniques. Then, we will argue how most existing navigation techniques only allow accessing the medium via a few select dominant domains and propose how we imagine navigation in digital media to be defined over non-dominant, typically semantic and content-defined domains while respecting the principles of direct manipulation in the associated interfaces. This will lead to a theory and generative model of such navigation principles, which we will introduce in chapter 2 "Theory".

In the four main chapters, we report on four medium to large research projects that were carried out to each analyze a different aspect of the theory and model in the context of a different type of medium:

PERSONAL ORCHESTRA allows semantic navigation in music through conducting.

- Chapter 3 "Time-based Media: Orchestral Music"
  PERSONAL ORCHESTRA is a conducting simulator where the user steps into the role of the conductor and can control an audio/video recording of a real orchestra by performing conducting gestures with an infrared baton. Although the act of conducting is a narrowly focussed and specialized case of media navigation, it is almost impossible to perform with standard audio or video navigation techniques. As such, PERSONAL ORCHESTRA is a good introductory example to show how a conceptually subtle change in navigation can lead to dramatic changes in the interaction and can require substantial technological advancement for its implementation. The latter point is illustrated through the PHAVORIT time-stretching algorithm.

- Chapter 4 “Time-based Media: Video Scenes”
  The DRAGON video player enables in-scene navigation by interacting with the objects in the video via direct manipulation. Thus, the navigation paradigm shifts from controlling time as an intermediate domain to directly controlling the constellation of objects in the video scene by dragging them along their motion trajectories. DRAGON specifically illustrates how mappings between the conventional dominant navigation domains and the content-defined subspaces can be established and how direct manipulation techniques can be applied even in mixed-domain, e.g., spatio-temporal, navigation contexts.

DRAGON allows semantic navigation in video by direct manipulation of the content.

- Chapter 5 “Hybrid Media: Presentation Visuals”
  Presentation visuals are commonly created in the form of slide decks. These media are exemplary in how the transfer from the analog to the digital representation has left the navigation possibilities limited to a small subset of what could—and should—be possible with digital media. FLY is an alternative approach for navigating visual supporting material for presentations that uses direct manipulation and zoomable UI approaches to leverage rapid exploration and spatial memory. The research around FLY is focussed on the evaluation aspect of navigation interfaces and includes studies with different users—authors and audiences—, different tasks—creation of presentation visuals and learning from presentations—, and the changes in presentation strategies that emerge from the new navigation of the medium.

FLY promotes a novel format for presentation visuals to enable semantic navigation.

- Chapter 6 “Non-time-based Media: Source Code”
  Source code is a digital medium in which efficient and easy navigation—or lack thereof—can have serious economical impact. Our STACKSPLORER and BLAZE plug-ins for the popular Xcode IDE offer quick and direct navigation along the edges of the call graph for C and Objective-C source code. Relaxing the constraints for true direct manipulation interfaces while still keeping their core ideas allowed us to create a navigation tool that compares favorably to established call graph navigation methods, like call hierarchy tree views. In the STACKSPLORER project we emphasize the need to analyze existing task structures and work-

STACKSPLORER and BLAZE facilitate call graph-based semantic navigation in source code.

flows of people using the medium to learn about their conceptual models and goal subspaces.

In the remainder of the thesis we give a summary of all four projects and conclude by reviewing our theory and generative model for navigation in digital media in the context of these examples. We also draw parallels to some emerging commercial products and design concepts before closing with a brief outlook on possible future work in this area.

1.2 Scope

Before we can discuss how to change navigation in digital media, we first have to ground our discussion on a couple of definitions. In particular, we need to specify what is meant by digital media (and what is not) and how these are built on a syntactic and semantic level to show how most navigation techniques offer access to the contents of a medium only through the easy-to-implement but difficult-to-operate syntactic domains. We also briefly revisit a definition of direct manipulation to be able to see how this interaction concept can be made a core part of our navigation techniques.

1.2.1 Definitions

There is no common definition for 'digital media'.

Finding existing definitions for digital media—ideally, scientifically dependable ones—is more difficult than one might think: Most commercial bodies that claim to deal with digital media (e.g., the *Digital Media Conference*, the *Digital Media Association*, or the *Digital Media Alliance Florida*) do not offer a definition at all, and even theoretical [McLuhan, 1964], technical [Feldman, 1997], or scientific works [Buckingham, 2006] either take digital media as a given or define it in terms of hardware or transmission formats that act as containers for media content.

'Digital media' should capture three basic properties.

For the discussion to follow, we need a definition of the term *digital media* that captures a number of core properties and requirements:

- Firstly, and most intuitively, digital media should relate to computers or other digital devices; it is important to distinguish them from traditional analog media such as a printed newspaper or physical projector slides.
- Another vital point is that digital media are vessels that carry *content* and that the kind of medium is defined in terms of the structure of that content. This sets our notion of digital media apart from the conception of digital media as physical carriers of data such as DVD-ROMs or hard disks.
- Also, we require digital media to be agnostic of any form of storage strategy, storage location, source encoding, or channel encoding of the content they carry, whereas *content* is only what is meaningful for humans. The medium thus constitutes itself as an appropriate human-readable representation of such content together with a context in which the content is meant to be used and a conceptual model of this context and usage.

To exemplify: Two videos of soccer games, one a WebM served through YouTube[2] and one on a Blu-ray Disc encoded as H.264, are still both the same kind of digital medium. Conversely, a photo of the last summer trip and a map of Barcelona, both stored as JPEGs on a website, are different types of media.

With these requirements in mind, we formulate the following definition of digital media:

DIGITAL MEDIA:
Information stored in a computer system, represented in a form that can be read and manipulated by humans for a specific use

Definition: *Digital Media*

Examples of digital media that fit both this definition and our intended discussion are: text, images, geographical maps, websites, videos, audio clips, typeset documents, or presentation visuals, but also more structured media such

[2] http://www.youtube.com

as spreadsheets, source code, or listings. While the definition could be extended to also include collections of digital media items—and a large body of research is directed at supporting browsing of such collections—this is not the focus of our work. As such, most of our experiments and observations do not directly apply to the navigation of media collections.

1.2.2 Structures of Digital Media

Current representations of digital media can be modeled as vector spaces.

In order to understand how navigation in digital media commonly works, we propose to play fast and loose with mathematical rigor for a moment and imagine an instance of digital media—say, a video—as a subset of a multi-dimensional vector space. For the technically inclined, the dimensions of this space come to mind immediately: Since a video can be represented as a stack of images shown in some temporal succession, it can surely be fully represented as a discrete three-dimensional time-space volume with added dimensions for pixel colors. Also, inside the computer, videos are actually represented in a very similar way; looking up a presentation time or frame number yields a pixel buffer that can be rendered to the screen. Understanding any video as a subset of such a time-space volume thus makes sense technically, it is mathematically sound and consistent, and it generalizes to every type of video, regardless of content.

We talk and think about media in terms of our conceptual models of the content.

What is important at this point, is to realize that while the content of a video can be described in terms of the axes of this vector space, very few people would actually do so; for most instances of digital media, we would rather describe their contents in terms of the conceptual models that we build based on the real world concepts that the contents represent. For a movie, we think in terms of characters, the plot, maybe geographic locations (Figure 1.2); for a single scene in the movie we might consider dialogues, the interplay between objects and persons in the scene, or how atmosphere and tension are built up. Regarding the content only in terms of frame numbers, pixel locations, and color values just does not fit well with our extensive human capabilities to interpret and describe the contents of digital

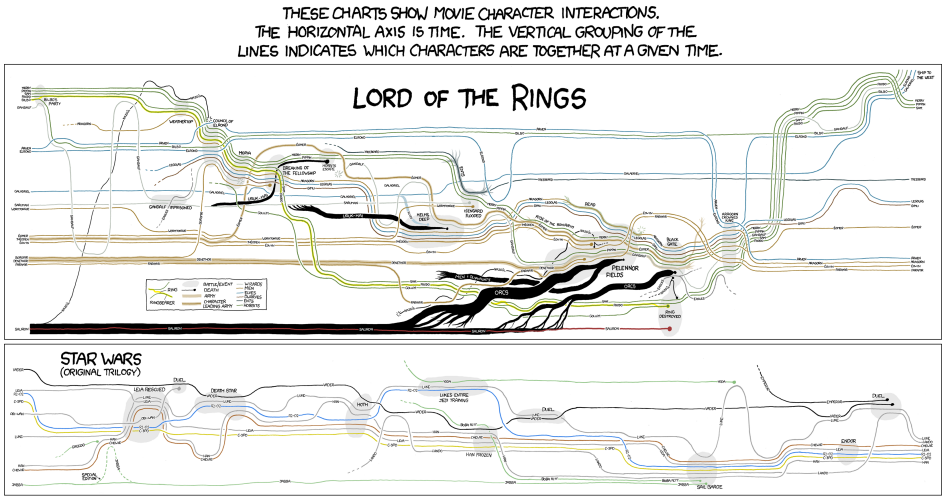

Figure 1.2: An alternative representation of movie timelines. Instead of relying on syntactic temporal measures, these are based on the actual content. (Image courtesy of xkcd)

media, unless, of course, one is concerned with describing video encoding or rendering.

Theoretically, however, all these rather non-technical properties of the content of a digital medium are somehow still located inside that mathematical vector space representation of that medium because that representation completely defines it. They just are not aligned with the axes of the space and may be incompatible with the sets over which these axes are defined. The location of the ball in a soccer match, for example, resembles a discontinuous set of snake-like sub-volumes in the time-space volume of a video that shows the match (Figure 1.3). Similarly, in animation movies, the atmosphere is often partially conveyed by using a certain color palette for certain scenes, which results in locally connected color 'blobs' when plotting the color space of such a movie over its time 1.4.

The content of a medium is contained but does not have a concise representation in this space.

To better distinguish between these two concepts of representing the content of digital media, we will call the former, technical one the *syntactic structure* of a medium and the one that is built upon a conceptual model—which usually depends on people's tasks and goals as we will see later—the *semantic structure*. This distinction as well as these desig-

We differentiate between the semantic structure and the syntactic structure of digital media.

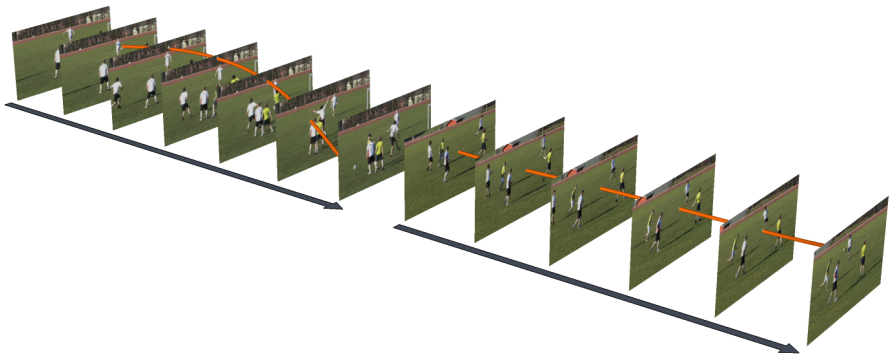

Figure 1.3: Video volume of a soccer match. The position of the ball spans a non-continuous subspace (orange) inside the volume. The discontinuity occurs at the cut between different camera shots (blue arrows).

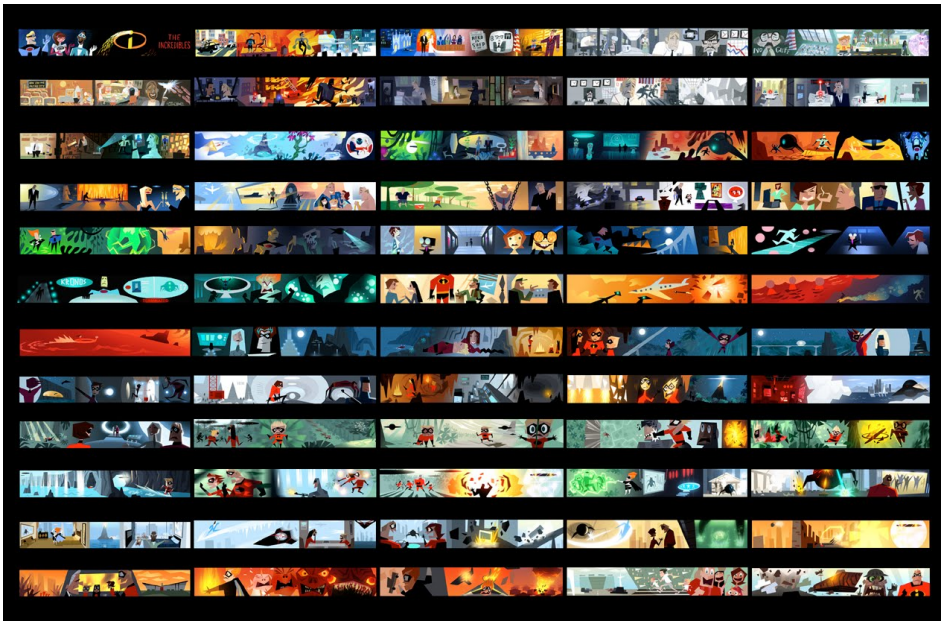

Figure 1.4: Color scripts, like this one from Pixar's *The Incredibles*, define the mood and atmosphere as conveyed by colors over the length of a movie.
Source: [Amidi, 2011]

nations are closely related to similar concepts in existing work on user interface models [Foley et al., 1997; Moran, 1981], direct manipulation interfaces, in particular, Shneiderman's syntactic/semantic cognitive model [1976; 1977; 1982], the concept of semantic and articulatory distances in interface languages by Hutchins, Hollan, and Norman [1985], and Lee's [2007] definition of syntactic and semantic time in time-based media; a more detailed description and comparison will be presented in chapter 2 "Theory".

1.2.3 Navigation in Digital Media

Current navigation controls are tied to the syntactic structure.

With these different ways of understanding the structure of the contents of digital media, we can now look at the established navigation techniques and interfaces for digital media. Figures 1.5–1.8 show some current standard navigation interfaces encountered on the OS X platform for common types of digital media. One thing that is immediately apparent is that most of these interfaces mediate the access to the contents of the medium through its *syntactic structure*.

Controlling the syntactic structure is compatible with a wide range of media types.

This, of course, has many advantages: such navigation techniques are universally applicable for whole classes of digital media that can be represented in the same syntactic spaces, and they can be easily implemented because they are often compatible with the storage, encoding, and memory structures inside the computer. On the other hand—and this already alludes to the central claim of this thesis—for many navigation tasks, moving only along the 'axes' of the syntactic structure can make navigating digital media unnecessarily difficult.

Users have to re-formulate their semantic goals in terms of the syntactic structure of the medium before they can start navigating.

If the semantic structure of the medium, which is induced by the user's conceptual model, is not aligned with its syntactic structure, any formulation of a navigation goal has to be first mapped into syntactic space where the actual navigation can be carried out. It is easy to see that this mapping alone is non-trivial and may in many cases even be impossible to determine a priori! To give a simple example, consider the task of navigating to this very example in an eBook version of this thesis: Even with the knowledge that it is an example that is meant to show how difficult it can be to map a semantically formulated navigation goal into

Figure 1.5: QuickTime X interface for navigating video or audio.

Figure 1.6: iTunes interface for navigating audio.

a location in syntactic space, the task will not be straightforward. We might consult the table of contents and the index or refer to a full text search for a query phrase possibly containing some subset of the words 'example', 'semantic', 'syntactic', 'goal', and 'mapping'. Other than that, we are pretty much out of options barring a linear navigation through the medium, constantly checking if we have reached our goal. In none of these cases, we have constructed the mapping of our goal ourselves: We have either used a pre-made mapping from an intermediary syntactic

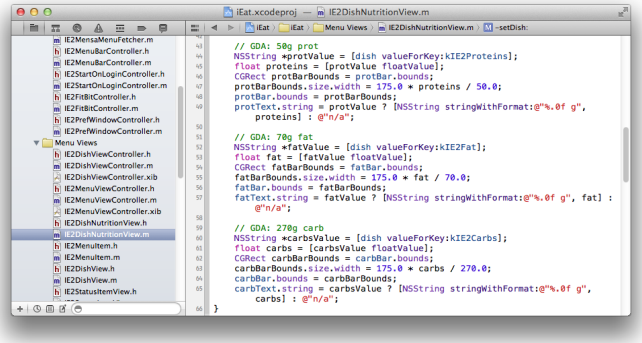

Figure 1.7: Xcode interface for navigating source code.

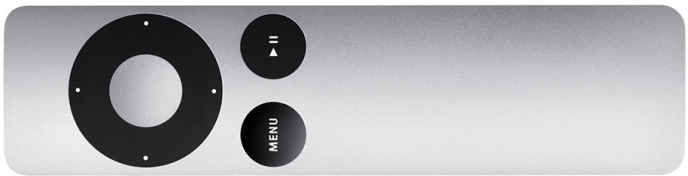

Figure 1.8: Apple remote control interface for navigating slide deck presentations.

structure that we hoped to be more closely aligned to the semantic one and which was kindly provided together with the medium. This is the case with the table of contents, the index, and the space of search queries. Or we have reduced our problem to evaluating the inverse mapping—which is easier as we do it all the time—over the full range of the navigable syntactic subspace.

Some media offer pre-made mappings of some semantic goals into the syntactic structure.

From the example, we can see that representations of the medium that are more closely related to the conceptual model of the user and that come with a pre-made mapping to the syntactic structure can help to navigate the medium; they are even needed unless we want to resort to an exhaustive exploration. Consequently, they are common not only for books but also for other digital media, for example, as site maps for websites or scene selection menus for DVD and Blu-ray movies.

Such indexes can only support a finite number of semantic goals but not the whole possible space.

There are, however, still some limitations with these approaches. In order to remain limited in size, these 'index' representations only offer access to the contents of the medium at very sparsely sampled locations. This means, in particular, that even if these representations are compatible with the conceptual model, they cannot span the full semantic space but are only disconnected 'islands'. Thus, if the navigation goal cannot be directly associated with any entry in the intermediary structure, it does not help. Ideally, we would need a navigation technique that allows the user to move through the semantic structure in a meaningful goal-oriented manner, with a sense of direction and distance towards the goal, and with the ability to correct wrong navigation actions on-the-fly. We will see in the next chapter that these requirements are very close to the properties of *direct manipulation* interfaces; most of the current direct manipulation navigation interfaces, however, seem to be designed in a way that assumes the goal to be expressed through the syntactic structure.

Before we can formulate these observations and ideas into a formal model for navigation interfaces, we will first summarize a number of established interface and interaction theories that will provide the framework for our own work. These theories supply a clearer definition and a more thorough explanation of the distinction between semantic and syntactic parts of interfaces and of advantageous ways to structure the interaction in a similar way.

Chapter 2

Theory

Over the course of the lifetime of HCI as a science, a number of theories and models of how to design and structure interfaces have emerged. These frameworks provide valuable guidance and a common vocabulary to classify and discuss interfaces and interaction techniques. In this chapter, we therefore briefly recap a selection of existing theories in the context of which we can then formulate our model for navigation in digital media.

2.1 Background and Related Work

Helpful theories include interface structure models and interaction modality types.

Among the established theoretical models and frameworks in HCI, there are two basic ones that are especially helpful to discuss higher level concepts, like, for example, *direct manipulation*, that we need for our navigation model: The first is the idea to formally represent the *interface* of any interactive system as a stack of conceptually independent layers. The second is the distinction between the two primary modalities of how one can *interact* with such a system, which is either by means of a form of conversation with the system or by manipulating a model-world representation of some problem space. These two approaches at classifying and describing interactive systems incidentally also represent the two main viewpoints on the field of HCI—from the com-

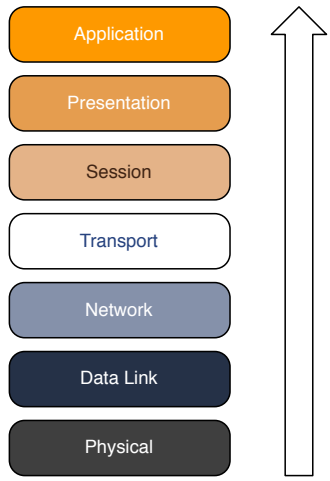

Figure 2.1: Open System Interconnection (OSI) layer model. Responsibilities are divided in a way so that each layer serves the layer above and is served by the layer below. The layers gradually change from being concerned with concrete, physical aspects at the bottom to abstract, conceptual aspects at the top.

puter science side, describing the structure of the artifact, and from the side of psychology, describing the nature of the human activity.

2.1.1 Interface Layer Models

Layer models are a common structuring approach throughout computer science.

In computer science, one recurring approach for modeling complex systems is to stratify the system into conceptually separate components. Ideally, these components build upon each other but can be independently described and analyzed, represent each a part of the system under a different level of abstraction, and can be reasoned about in their own contexts of formalization. Such models can be found in software engineering [Gosling et al., 1989] or in communication networks, with the Open Systems Interconnection (OSI) layer model [Zimmermann, 1980] (Figure 2.1) being perhaps the most well-known and prominent example.

Layer models for user interfaces of interactive systems have been proposed as well over the last decades; for the discussion at hand, it is helpful to introduce those by Foley

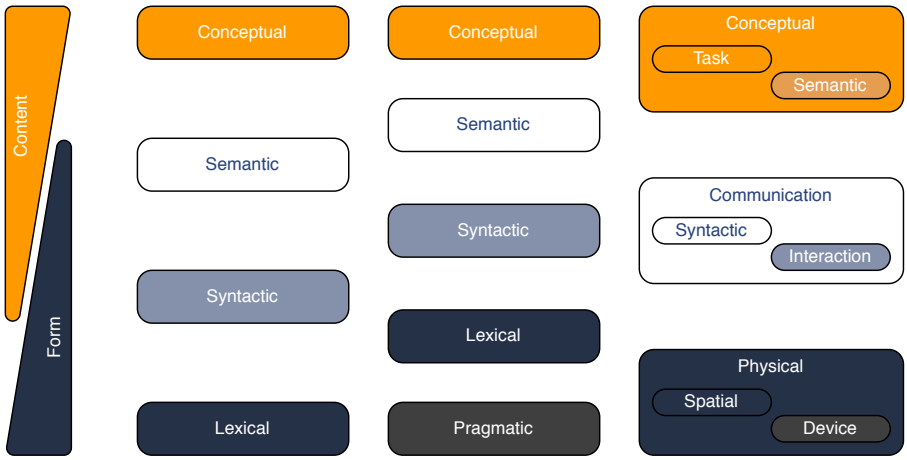

Figure 2.2: Three different *layer models* representing the interface as a means of communication between the user and the computer. From left to right they were proposed by Foley et al. [1997], Buxton [1983], and Moran [1981].

et al. [1997] (Figure 2.2, left) and Moran [1981] (Figure 2.2, right).

Foley et al. propose a four layer model for user interfaces.

In their standard text book on computer graphics, Foley et al. [1997] describe a layer model dividing the interface into its conceptual, semantic, syntactic, and lexical levels. The interface in their model consists of two languages: the input language—expressed in the manipulation of input devices or other modalities of issuing commands—in which the user communicates with the computer, and the output language—expressed in visual shapes, text, or sound—in which the computer communicates with the user. Languages generally consist of a set of *meanings*, a set of *forms*, and a mapping that connects the two [Fillmore, 2003]. Interface languages being no exception, the layer model of Foley et al. builds on that separation—while the conceptual and semantic layers of an interface constitute the *meaning or content* of the communication between user and computer, the syntactic and lexical layers capture the way of *conveying* that meaning:

The users' mental model is captured by the *conceptual layer*.

1. The *conceptual* layer captures the basic capabilities of the underlying application. This includes what kinds of objects are used to represent the content, how these

Figure 2.3: Interface of the QuickTime 7 media player.

can be operated on, their relationships, and which tasks can be accomplished. It thus forms what is sometimes called the 'user model' or 'mental model' of the system [Norman, 1988]. For a simple audio player application like the one in Figure 2.3, for example, the conceptual layer would include that pieces of audio can be loaded into the player, that one piece can be played back at a time, that users can fast forward and rewind through the current piece, and maybe that there is a way to navigate to a certain point in the piece.

The conceptual layer affects all lower layers of an interface.

Shneiderman and Plaisant [2010] state that 'decisions about mental models affect each of the lower levels' of an interface. This is an important point: Since the conceptual layer of the audio player does not include the capability for, say, adjusting the tempo of the playback or adding musical expression by controlling the micro-timing of a classical piece of music, the interface will provide no means to facilitate these tasks. In chapter 3 "Time-based Media: Orchestral Music" we will see an audio player interface with a conceptual layer that deals with exactly these capabilities.

Often, the conceptual layer is modeled after that of a known non-digital tool, limiting the interface.

The conceptual layer of an interface is often based on a mental model of non-digital tools traditionally used for the same or closely related tasks, with which the user is familiar. Such *interface metaphors* are very common—the interface of most operating systems has been following the *desktop metaphor* for decades now—and can help the interface designer to communicate the mental model of the application to the user. There

is, however, also the danger of users interpreting the metaphor more literally than intended or of limiting the theoretical capabilities of applications to the technical capabilities of the tools they imitate [Halasz and Moran, 1982]. The latter point is especially true for many media navigation interfaces; the audio player example above limits the use of the application to whatever was already possible with a tape recorder in the 30s from which the interface metaphor was clearly derived (cf. Figure 1.1). We will encounter this specific problem again in chapter 5 "Hybrid Media: Presentation Visuals".

2. The *semantic* layer contains the functional descriptions of the entities in the conceptual layer. It thus attaches meaning to every expression in the language of the interface and specifies the functional requirements, context, and results of each possible action. In our audio player, for example, users can play back a piece of audio by invoking a 'play' command or by revoking a 'pause' command; both require a piece of audio to be loaded, the latter is only possible if a previous play action has been paused at some position in the timeline, and it is expected to start playing again from that very position. The conceptual layer together with the semantic layer comprise what we described above as *meaning and content* of the communication between user and computer.

   The functional description of each interface element is part of the *semantic layer*.

3. The *syntactic* layer defines the sequencing rules for the lexical atoms of the interface language. This connects the temporal order and spatial layout of input and output primitives to their corresponding functions in the application. The command to move the playhead of the audio player to the beginning of the current piece might be constituted by the sequence of 'move mouse pointer over the rewind button, click, and quickly click again'. On the output side this could be responded with by 'briefly highlight the button, move the graphical representation of the playhead to the leftmost position on the timeline, and stop the sound'.

   The grammar of interaction is defined on the *syntactic layer*.

   The syntax of any language is usually a hierarchical construct, and—on a larger scope—interfaces are usually hierarchies of languages. Highlighting a button

Depending on the scope of analysis, the boundary between semantic and syntactic layer is fluid.

to indicate activation is a syntactic concept in the language that constitutes the interface of an application. For a language that expresses only the button as an interface element, this may rather belong to the semantic layer as the functional description of what a button does to convey the meaning of 'activation'. For that language, the spatial arrangement of lines and surfaces that form the graphical shape of that button is a syntactic concept. Foley et al. offer a definition that allows that kind of dependence on scope and context: The syntax is the temporal and spatial sequencing of such units of meaning that 'cannot be further decomposed without loss of meaning' [Foley et al., 1997].

The actual lexemes of the interface language reside on the *lexical layer*.

4. The *lexical* layer finally deals with how these units of meaning are precisely formed. On the input side, this may include how the available input devices have to be manipulated to produce location, activation, or text input. On the output side, these are graphical, textual, or aural primitives from which the semantic units of meaning are assembled. The syntactic and lexical layers represent the *form* of the communication between user and computer.

There exists a number of other user interface layer models which are either very similar or extensions of this model:

Buxton's layer model is similar but with more emphasis on input devices.

Buxton [1983], for example, argues for a more differentiated treatment of the lexical layer which mixes up the construction of units of meaning with their spatial embedding into the context on the output side and the physical means and gestures to do so on the input side. He proposes refactoring that layer into two distinct components, one dealing with the *lexical* issues, such as 'the ordering of lexemes and the nature of the alphabet', and the other dealing with the issues of *pragmatics*, such as 'gestures, space, and devices' (Figure 2.2, middle). Within this extended framework it becomes possible to extend the discussion about an interface to the subtle differences in the physical operation of input devices or their spatial arrangement in the work space, both of which undoubtedly have a strong influence on how the user perceives and uses a system.

Moran [1981] proposes a stack model with three components, dividing the interface into the *conceptual*, *communication*, and *physical* components, each being built up by two sublevels (Figure 2.2, right). In his nomenclature, the conceptual entities of the system make up the semantic level, which—together with the task level describing the user's task and its context—forms the conceptual component. The communication component consists of the syntactic and the interaction levels. The syntactic level defines the structure of the language while the interaction level deals with the mappings between the physical interactions of the user and the logical interactions in the user interface. Finally, the physical component is divided into the spatial level, which deals with the layout of information or control elements on the output side, and the device level, which is responsible for input and output devices and their physical properties.

The model by Moran differs mostly in terms of what constitutes the users' mental model.

While the components and levels of Moran's model can be easily mapped to the four layer model by Foley et al., they have different views on what constitutes the *user's mental model* of the system: In the four layer model, the mental model is fully represented by the conceptual layer and thus is believed to be formed by the objects of interest, their properties, and their relations alone, and is independent of the concrete functionality the system offers and its physical bindings. In the three component model, the mental model is the combined effect of the whole interface.

2.1.2 Activity and Interaction Models

A well-known and often quoted interaction model, Norman's *seven stages of action* [1988] takes a slightly different approach. It focuses rather on the different layers of the process of interacting with a system than on the interface itself. Norman divides the cyclic pattern of action and perception that users employ to reach a goal into seven stages; three of them describe the execution of an action, three the evaluation of its outcome, and one the goal itself (Figure 2.4):

The *seven stages of action* describe the steps during the interaction with an interface.

1. *Forming the goal*
   According to Norman, each interaction with a system starts off by the user forming an abstract goal. This

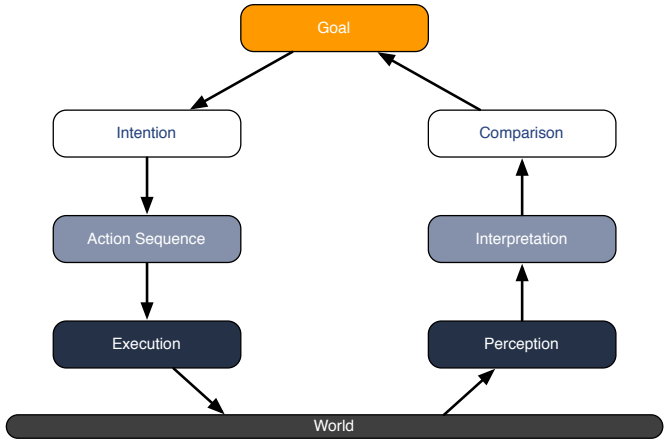

Figure 2.4: The *Seven Stages of Action* are an interaction model that is helpful for discussing the individual steps of any communication between user and computer and for categorizing breakdowns in such interactions.

goal will likely be expressed in terms of the user's mental model of the task and context; as such, it does not necessarily correspond to any specific interface command the system offers. In Foley's four layer model, this stage would thus correspond to the conceptual layer of the interface. To stick with the above-mentioned example of the audio player software, a goal would be to navigate to the start of the second movement of Beethoven's *Grande Sonate Pathétique* and listen to the wonderful *adagio cantabile*.

2. *Forming the intention*
   This stage represents coming up with a plan to reach the goal and includes assessing the functional requirements for each step of the plan as well as formulating the sequences of steps of how to fulfill them. All steps devised at this stage can be mapped to—possibly complex—commands in the interface but may still be formulated in abstract terms. Foley's semantic layer roughly captures the interface entities relevant for this stage. In our example, the intention could be to load the audio file containing the sonata, then to look up the starting time of the second movement in a booklet, to drag the playhead on the timeline slider to the position representing that time, and to hit the play button.

3. *Specifying the action sequence*
   After the intention is formed, the steps of the plan have to be reconstructed by sequences of those interface commands the system offers. This stage is the bridge between the semantic world of the 'meaning' of the action and the syntactic world of its 'form' as a string of elementary inputs to a system. As such, the stage acts on Foley's syntactic layer and would result in a defined sequence of click locations and precisely specified drag operations in the audio player example.

4. *Executing the action sequence*
   Finally, the action sequence is physically executed and thus imposes some change on the world (or the system the user is interacting with), to which the world then reacts by altering its observable state in some way. This stage corresponds to the lexical layer in the four layer model, because every executed atomic action can be expressed as a lexical token or *lexeme* in the language that constitutes the interface

5. *Perceiving the state of the world*
   The change in the state of the world is then observed by the user, in the first instance only as changes in Foley's lexical layer. These constitute what is normally called *feedback*, and it can be conveyed through visual, auditory, or haptic channels. It is important to keep in mind that humans will only associate the feedback given by a system with an executed action if the response time of the system to generate that feedback is very small[1], usually in the order of 100ms.

6. *Interpreting that perception*
   The type and the spatial and temporal order of the individual lexical tokens that make up the perceived feedback are used to derive a semantic meaning in this stage. In this process, the users form an assumption over the state of the system in terms of their mental model as opposed to the perceived state of the system, which is only syntactically defined. For the audio player, for example, the observation of a playhead that moves continuously on the timeline without manipulation by the user may be interpreted as the system

[1] Most of the results regarding this kind of *perceived causality* are based upon the early experiments of Michotte [1963], a detailed overview can be found in [Scholl and Tremoulet, 2000].

being in the playback state. In this case, of course, the auditory feedback would be much more important.

7. *Evaluate the outcome with respect to the original goal*
   The assumed state of the system in the user's mental model is compared to the desired state described in the initial goal. If the two are congruent, the goal has been reached, and the action is completed; if not, the goal may be adjusted or not, and a new cycle of the seven stages begins.

The seven stages of action can explain the different phases present in certain interactions.

The most important difference between this model and the ones described above is that it gives us a framework to describe the interaction with a system during both *open-loop* and *closed-loop* tasks. Also, we can use this model recursively for hierarchical goal structures: For example, the positioning of the playhead on the timeline at a certain time as described in the example above can be thought of as a subgoal. We can imagine that during that dragging operation, while the user is homing in on the target position, the same stages are traversed in rapid cycles on a lower cognitive level, always checking if the target position has been reached and coming up with new intentions how to perform the next submovement. Describing such closed-loop tasks in terms of recursive planning, execution, and evaluation is compatible with lower level models of user perception or action, such as *Fitts's Law* [Fitts, 1954] or the *human processor model* [Card et al., 1986].

The model gives a classification scheme for interaction breakdowns.

Another advantage of the seven stages of action model is that it gives us a terminology to classify interaction breakdowns. According to their location in the seven stages, these are commonly called *gulfs of execution* if they occur somewhere between the user's goal and the physical action that affects the world, or *gulfs of evaluation* if they occur somewhere on the feedback and evaluation branch of the model (Figure 2.5).

Gulfs of interaction can happen on both sides of the model.

Of course, multiple gulfs are possible on each of the two sides of the model: On the execution side, it may be difficult to formulate a plan to reach the goal because the semantic functionality represented by the system is a poor fit for the user's mental model, it may be difficult to map an intention to a sequence of actions in the interface because the available syntactic actions do not map well to the semantic

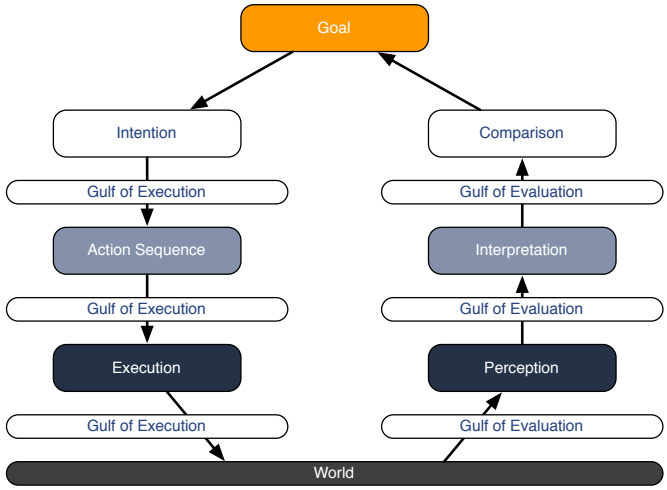

Figure 2.5: Depending on the side of the interaction cycle on which they occur, the causes for interaction breakdowns can be classified as either *Gulfs of Execution* or *Gulfs of Evaluation*.

steps the user has to undertake, or it may be difficult to execute an action sequence because of physical properties of the interface, like small buttons or timeouts. On the evaluation side, a corresponding set of gulfs can arise, again being caused by the respective differences between the physical, syntactic, and semantic representations of the system and the user's expectations and abilities. Hutchins et al. [1985] provide a very helpful formalism regarding these semantic and syntactic gulfs of execution and evaluation in the context of *direct manipulation*, which will be a central building block for our model of media navigation interfaces (cf. 2.1.4 "A Theoretical Framework of Direct Manipulation" (p. 33)).

Users either interact with the computer as a tool to manipulate a representation of a world or as an agent in a conversation.

When looking at models to classify the interaction itself, independent of the layer structure of the interface, we can see that there are two fundamentally different ways of how users can interact with a computer or other interactive systems. The distinction comes from the two different ways to understand the computer as an instrument to aid the users in their task:

One is to regard it as a passive tool that provides the facilities to create, shape, and modify digital representations of real-world concepts, data, or entities. These representations exist in the context of a virtual world, and the user can directly change their relevant properties through appropriate extensions of his physical capabilities into this world and without any intermediate agency.

The other is to see the computer as a (pro-)active assistant that can perform certain tasks on behalf of the users at their command. This form of interaction resembles a conversation where one party expresses their intent in a way that the other can understand, and the other acts upon it and presents a result in response. Consequently, these two concepts are usually referred to as *model-world* and *conversation-based* interaction.

The dominance between the two concepts has shifted from conversation-based to model-world over the years.

Historically, interaction with the first computer interfaces followed the conversation metaphor; the only way to interact with a computer was by writing programs, feed them to the computer (first in the form of punch cards, later directly at terminals), and receive—after some waiting time—a response. This kind of conversation was, of course, rather unsatisfying with its long response times and essentially context and state free character. Later, conversation interfaces evolved to resemble the character of human conversation more closely: The vocabulary changed to first mnemonic and later natural language commands or consisted of predefined commands that only had to be recognized and selected instead of remembered and produced. The stateful nature of human conversation with its ability of self-reference was adopted, so that dialogs could request additional information regarding the current established context of the conversation. And the rate of turn-taking improved to the point where it could accommodate human conversational habits (cf. [Miller, 1968] and [Card et al., 1991]).

Interfaces following the model-world metaphor were established later, when the input and output capabilities of computer hardware allowed a consistent representation of the virtual world containing the objects or data of interest—usually through computer graphics—and manipulation of these objects at interactive rates—usually through positional locator devices. Sutherland's Sketchpad [1963] is a prominent early system adopting a model-world approach.

This 'new' paradigm became vastly successful in a class of interfaces that were first characterized by Shneiderman [1982]. He coined the term *direct manipulation* for such model-world interfaces that fulfilled certain criteria of visibility, feedback rate, and choice of conceptual model.

Hybrid approaches between the two paradigms exist.

There was an ongoing debate lasting the better part of the 1990s about the strengths and limitations of the two interface metaphors and about which of the two should be followed in terms of interface design and research. A good example of this debate are the panel discussions between Ben Shneiderman and Pattie Maes at IUI'97 and CHI'97, excerpts of which have been published in the December issue of the Interactions magazine [Shneiderman and Maes, 1997]. Today, both interface paradigms coexist and have merged 'into multimodal interfaces, virtual environments, and ambient intelligence' [Salvendy, 2005] applications with many hybrid forms between the two [Frohlich, 1993; Kwon et al., 2011].

2.1.3 Other Models

In addition to the four models presented, there exists a number of other theoretical frameworks to classify and understand human-computer interaction and a plethora of 'golden rules' or other design guidelines for the creation of good user interfaces. The ones introduced in this chapter, however, are especially important for our work, because they provide a clear boundary between the semantic parts of the interface that are concerned with the conceptual representation of the problem and task at hand and the syntactic parts of the interface that give a form to these representations. This will later enable us to extend this stratification from the interface to the data that is accessed and manipulated through the interface—in our case, digital media—and to show how this allows us to construct navigation interfaces that can be more efficient and easy to use. The presented models also allow us to discuss in depth the ideas behind and properties of *direct manipulation*, one of the most important interaction concepts in HCI [Frohlich, 1993], a cornerstone of many current user interfaces, and one fundamental aspect of our own interface model.

2.1.4 Direct Manipulation

Shneiderman identified the concept of *direct manipulation* as the common characteristic of a number of well-designed user interfaces.

After examining a number of successful user interfaces that were reported to 'generate a glowing enthusiasm among users', Ben Shneiderman published a paper [1982], in which he analyzed the common characteristics of these interfaces. He proposes a design model based on these observations, centered around three concepts:

- Continuous representation of the object of interest.
- Physical actions or labeled button presses instead of complex syntax.
- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

Direct manipulation interfaces directly represent the objects of interest from the users' mental models.

The idea is that such *direct manipulation* interfaces allow the users to directly act on virtual reifications of the objects relevant to their tasks, and that these objects should be compatible with the users' mental model. In this way, users have a clear sense of context—in the sense of their own semantic understanding of the task and data—regarding the state of the system and regarding any action they can perform. Through the immediate display of the results of any action a user performs, the relation to its context remains intact, and it becomes immediately visible if that action furthers the users' goals; if not, each action can easily be revoked—usually by the physically reversed action, which also is an appropriate and natural way outside of computer interfaces.

In his paper, Shneiderman [1982] offers a list of potential advantages of interfaces with these properties:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.

- Error messages are rarely needed.
- Users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity.
- Users have reduced anxiety because the system is comprehensible and because actions are so easily reversible.

Division of Semantic and Syntactic Layers

The idea of supporting tasks on the semantic level comes from the area of programming languages.

Shneiderman relates this approach of supporting the users on the semantic level of their tasks to his earlier research on the structure of programming knowledge and the understanding of source code [1977; 1979]. There, he presents evidence that the knowledge of programmers can be divided into hierarchies of semantic and syntactic layers: The hierarchy of semantic layers captures language-independent programming concepts, ranging from knowledge about mathematical methodology on a high level, e.g., statistical analysis, to the sequences of logical statements that make up algorithms to implement such methodology on the lower levels. The syntactic layers contain language-dependent knowledge on the syntax of tokens and the grammar of statements.

Understanding and solving problems always happens at the upper semantic and conceptual layers and is then propagated down the semantic/syntactic hierarchies—first operationalizing the solution, then solving the functional dependencies of the individual steps, then transforming this into a sequence of actions before finally performing the actions and checking the results. This methodology is similar to that described by Pólya [1957], and it is compatible with Norman's seven stages of action.

Interaction becomes easier when users do not have to translate between semantic and syntactic representations of objects and tasks.

In the context of direct manipulation, this means that if the interface can indeed offer a representation of the users' task or problem directly in the semantic domain and have them manipulate the objects of interest in ways that directly translate to semantic concepts, the whole traversal of the semantic/syntactic hierarchy—or the seven stages of action—becomes much easier: the decomposition of semantic con-

cepts to syntactic tokens—when thought of as a mathematical transform—can be ideally the identity mapping. The same is true for the output of the system, where syntactic sequencing of the lexical tokens, their interpretation, and their translation into the semantic domain of the task can also be simplified. Direct manipulation in this form thus has the potential of bridging Norman's gulfs of execution and evaluation all at the same time; and the closer the mapping between the problem domain and its representation in the interface comes to the identity mapping, the faster a user can iterate through the seven stages.

Criticism

Criticism about direct manipulation is centered around its lack of meta representations that can be referenced again.

As already mentioned above, the idea of model world interfaces in general and the concept of direct manipulation in particular have also been criticized to complicate certain classes of tasks. Most of these could be solved much better using interfaces following the conversation metaphor. The central point is that there exists a tradeoff between the directness of actions including their being accessible on the higher layers of the task structure and the descriptive power of the interface language. In abandoning the indirection of *describing* tasks and actions through some intermediary representation, direct manipulation interfaces do not allow to refer to these tasks or actions later but only to their results. This lack of meta-expressiveness causes a number of tasks to be difficult or tedious when using direct manipulation and may be the main reason for conversation interfaces or mixed-mode interfaces to be still dominant in areas like programming, data retrieval, or statistics.

The following list of such tasks and other problems with direct manipulation has been compiled from similar discussions by Hutchins et al. [1985] and Frohlich [1993]:

- *Repetitive tasks*
  Because there is no way to describe a task and then use the description as an argument to an operation, in a pure direct manipulation interface there is no way to perform repetitive tasks short of performing the task over and over again. Also, direct manipulation inter-

faces forbid by their very design to employ some intermediary agency to perform the task for the user.

- *Acting on sets or classes of objects*
  Closely related to the first point, direct manipulation interfaces make it difficult to operate on groups of objects. Although many established interfaces, e.g., vector drawing programs or word processors, allow to first make a group selection by subsequently adding objects and let the user then manipulate all objects in the group, this method introduces a conceptual ambiguity: many operations could be performed on the group itself or on its member objects, and it is often not clear which will happen. Also, 'filtering' a set of objects to create a group according to certain characteristics requires to have some form of description of these characteristics, which is difficult in direct manipulation interfaces.

- *Acting on objects that are not visible*
  One defining concept of direct manipulation is that users act directly on their objects of interest, mostly through a visual representation of these objects. If the model world exceeds a scale that can be displayed as a whole, these objects may not be visible on the screen at a time, which first requires a navigation to make the object visible. One problem with that is that if the navigation is a direct manipulation operation as well, this becomes difficult too because the user cannot see whether 'their actions further their goals' if the goal is invisible; in such cases, direct manipulation may actually open up a gulf of evaluation. Abstract concepts that cannot be easily represented spatially—time comes to mind—also present a challenge.

- *Tasks requiring accuracy*
  Since every manipulation in the model world is directly controlled by the users' physical actions, tasks that require a certain degree of precision may be less suited for direct manipulation. Some interfaces employ mixed mode concepts and try to interpret user input according to the context to some degree; *grid snapping* or *smart guides* in graphics applications are widespread examples. The accuracy problem reveals that not only the user should be able to work in the semantic context of their mental model but also that

their input should be processed with the appropriate relative semantic granularity. A theoretical framework dealing with such semantic granularity in audio interfaces was, for example, proposed by Lee [2007]. Also, we present experimental approaches to automatic adaption of semantic granularity in video navigation in chapter 4.2.3 “Interactive Scoping and Trajectory Filtering”.

- *Tasks that refer to other tasks*
  Another effect of the lack of the possibility of self-description is that in direct manipulation a task can neither easily refer to itself nor to other tasks. This is true both for previous tasks and tasks that should be performed in the future. In such cases, the only way is to create explicit representation of actions and tasks as objects of interest. This, of course, only relocates the problem to another level as the manipulations of these task objects themselves are yet again not expressible in the interface language.

- *Performing multiple tasks at the same time*
  Because in direct manipulation the user is always ‘in the loop’ and cannot delegate tasks to an agency for asynchronous execution, the capacity for performing a number of tasks at the same time is limited by human physiological (input) and cognitive (control) constraints.

- *Direct manipulation discourages from providing new ways of thinking and interacting with a domain*
  Direct manipulation allows users to manipulate objects of interest in the context of the problem domain; this domain has to be familiar to the users in order to solve the task. Hutchins et al. [1985] argue that direct manipulation therefore might discourage users to develop new models of understanding of the task domain.

- *Direct manipulation transfers demands on task domain knowledge to the user*
  Similar to the restrictions that apply for performing multiple tasks, in direct manipulation interfaces users will usually have to solve the cognitive aspects of the problem or task themselves. One could say that while they offer directness and control in performing tasks,

> these interfaces do provide less assistance in understanding and solving problems—the demands on domain knowledge are transferred to the user.

For the topic of media navigation interfaces, some of these problems can be mitigated by adding conversation aspects, thereby effectively creating a mixed mode interface, and some just do not apply to navigation tasks. We will thus revisit the problems described here where such a discussion is relevant in the context of the individual interfaces.

A Theoretical Framework of Direct Manipulation

Hutchins et al. tried to capture different aspects of direct manipulation in a design space.

Three years after Shneiderman's original article on direct manipulation [1982], Hutchins et al. [1985] published a theoretical analysis of the interaction technique, trying to give a 'cognitive account of the advantages and disadvantages of direct manipulation'. Their concern was to develop a design space that could formally capture the feeling of directness associated with this class of interfaces.

The feeling of directness is influenced by *engagement* and the *semantic* and *syntactic* distances in the interface.

According to Hutchins et al., the two primary factors that influence this feeling are what they call *distance* and *engagement*. Distance, which is further divided into *semantic distance* and *articulatory distance*, is a more formal concept to describe the complexity of the transformations between the different semantic and syntactic layers, which we have mentioned earlier. Engagement, on the other hand, is much less formalized and describes the 'qualitative feeling that one is directly manipulating the objects of interest'.

There are two exemplifying questions Hutchins et al. put forward to illustrate the concept of semantic distance in an interface language: "Is it possible to say what one wants to say in this language?" and "Can things be said concisely?". Semantic distance is a measure of the answer to these questions, thus describing the closeness of the interface language to the conceptual model of the task. This measure gives a name to one the qualities of direct manipulation interfaces that Shneiderman [1982] claims was required: the degree of conceptual compatibility between the interface representation of the objects of interest and the users' mental model of those objects.

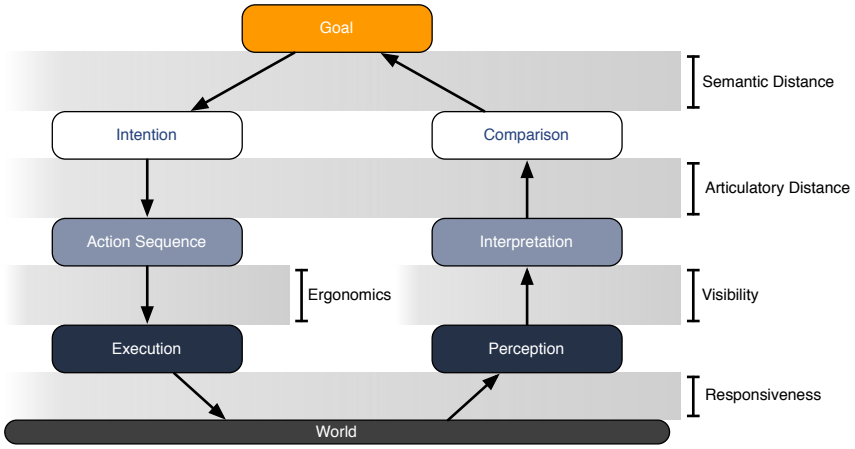

Figure 2.6: The gulfs in the *Seven Stages of Action* model can also be formulated in terms of the properties of *direct manipulation*. Direct manipulation interfaces with low semantic and syntactic distances and rapid feedback allow fast cyclic traversal of the stages, making them well-suited for control tasks such as navigation.

The semantic distance corresponds to certain gulfs in the seven stages of action.

At the same time, this idea solidifies our discussion on the connection between direct manipulation and Norman's seven stages of action; a large semantic distance means that the gulfs of execution and evaluation are wider or more difficult to bridge for the user. On the execution side, a small semantic distance lets intentions be formed more easily because the interface offers object representations that are directly expressed in the conceptual language of the user goal; on the evaluation side, a small semantic distance reduces the amount of processing the user has to do in order to determine from the system's output whether the goal has been reached or not (Figure 2.6).

Interfaces with a small semantic distance often depend on the actual task and thus may suffer from a loss of generality.

Systems with large semantic distance can, of course, still be used effectively, but the users will have to develop competence and habituation to automate response sequences and think in the system's language [Hutchins et al., 1985]. Also, creating systems with small semantic distance comes at a cost: Transferring the burden to translate between the semantic language of the user's conceptual model and the syntactic language of the system's internal representations from the user to the developer of the system makes the implementation of such systems much more difficult (a point

that we will explicitly pick up in our model for navigation interfaces). The tailoring of the interface towards a certain conceptual model obviously incurs a loss of generality—offering functionality at a lower semantic level could accommodate for a larger number of tasks. And it is important to keep in mind that the semantic distance depends heavily on the task itself. Hutchins et al. illustrate the latter point through an example of musical instruments, comparing a piano and a violin: If the task is to produce a certain note on the instrument, the semantic distance of the piano is much smaller—the concept of a note is perfectly represented by the keyboard of the piano providing one single dedicated way for each note to produce it. The violin undoubtedly requires more cognitive effort to form the intention, offering four strings and practically an infinite number of positions for the fingers to modify the pitch for each string. If the task, however, is to add expression to a note, say, by producing a vibrato modulation, the violin possesses a semantically much more direct embodiment of that concept while the piano cannot really produce the effect at all.

The syntactic distance can also be responsible for some gulfs in the seven stages of action.

The counterpart of semantic distance, *articulatory distance*, describes the complexity of the transformation between the semantic and syntactic representations of the objects and actions in the interface. A small articulatory distance means that there exists a direct structural relationship between the meanings of expressions in the interface language and their physical forms. In terms of Norman's seven stages of action, this measure corresponds to the widths of the lower level gulfs of execution and evaluation (Figure 2.6). While reducing the semantic distance is solely a concern of interface design, Hutchins et al. mention that reducing the syntactic distance also depends on the technological capabilities of the computer and its input and output devices.

Engagement is more difficult to measure and depends on several factors.

Apart from the semantic and articulatory distance, Hutchins et al. [1985] state that *engagement* is the second foundation for the feeling of directness in direct manipulation interfaces. While engagement is meant more as a qualitative term, they do list a number of requirements for it: First, semantic and articulatory directness facilitate the feeling of engagement. Second, the interface language of the system should be designed for inter-referential input and output; this means that the result of an operation can be used as an input for the next or another operation. Third,

the interface should be responsive at all times and must be unobtrusive in itself.

Rapport is a parallel concept to *engagement* in conversation-based interfaces.

It has been debated if this feeling of engagement is exclusive to model-world interfaces in general or even to direct manipulation interfaces in particular, and the lack of a dependable definition of the term does not help the discussion. Frohlich [1993] argues that a closely related concept can be attributed to certain conversation-based interfaces, which he calls *rapport*. From this point of view, both serve the same role as an interface characteristic of evoking a positive emotional stance towards the system—be it represented by a model world or conversational agent.

The design space by Hutchins et al. is spanned by the *combined semantic and syntactic distance* and the *classification* into model-world or conversation type of the interface.

In conclusion of their analysis, Hutchins et al. [1985] propose a design space for interfaces based on two axes: The first covers the different modalities and sorts interfaces into the model-world and conversation paradigms. The second is based on a combined measure of their semantic and articulatory distances (Figure 2.7). They claim that for users, the feeling of directness should increase over the diagonal, starting from traditional command line interfaces in the lower left (conversation-based and large semantic and articulatory distances) and leading up to direct manipulation interfaces in the upper right (model-world-based and small semantic and articulatory distances).

For our discussion, it is helpful to adapt the design space and keep the two types of distance separate.

While the design space in this form may offer some value for the discussion at hand, it does have a number of shortcomings: First, as Frohlich [1993] has pointed out, differentiating between conversation and model-world interfaces may not be appropriate when trying to attribute directness to a system or interface; there are many classes of tasks, like the ones stated above (2.1.4 “Criticism”), where model-world interfaces fall short and which could be argued to be solved in a much more direct way using a conversation-based interface. Second, in combining the two distance measures into one axis, the design space becomes an unnecessarily ‘blunt’ instrument when it comes to classifying interfaces and suggesting improvements. As we will see later, many common interfaces for media navigation succeed in minimizing one of the two distances but neglecting the other entirely. For an informed discussion about such cases, it thus seems beneficial to keep the two distance concepts separate. In the next section, we will introduce our

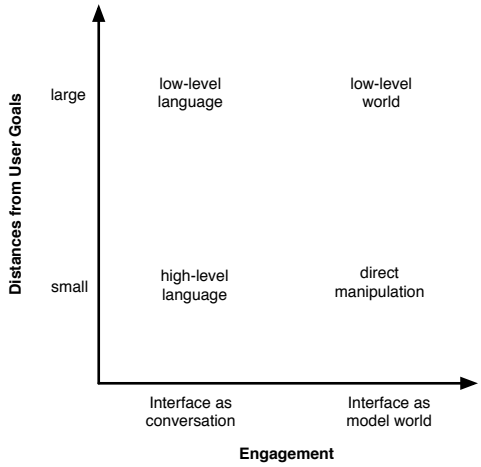

Figure 2.7: The design space by Hutchins et al. categorizes interfaces by their combined semantic and articulatory distances and their level of engagement. Conversation-based interfaces with a large combined distance are considered the least direct; *direct manipulation* interfaces are considered the most direct. Figure adapted from: [Hutchins et al., 1985]

own design space for media navigation interfaces, which builds upon that by Hutchins et al. but addresses some of its shortcomings and is focused on navigation.

Edge Cases and Exceptions

In front of this background we can now discuss two classes of interfaces that are often referred to as direct manipulation but can be argued often not to fulfill all of the requirements we have identified so far. This discussion is especially interesting in the context of this thesis, because a large majority of media navigation interfaces fall in either or both of these classes, and we believe that they fall short in their promise to provide efficient and direct support—in the sense of direct manipulation—for media navigation tasks. The two types of interfaces in question are interfaces that are based on *technical metaphors* and interfaces that allow *continuous manipulation on the syntactic but not on the semantic layer*.

Technical metaphors can help in designing direct manipulation interfaces but impose the risk of limiting the interface.

Direct manipulation is built upon the idea of allowing users to manipulate objects of interest that directly correspond to entities of their mental models of the task. This is achieved—as Hutchins et al. [1985] have pointed out—by carefully designing the representations of these objects in a way that reduces both the semantic and articulatory distances. Shneiderman himself notes, however, that finding such a representation that allows for a trivial transformation between the semantic and syntactic structures is the difficult part: 'the trick in creating a DM system is to come up with an appropriate physical model of reality' [Shneiderman, 1982]. An often adopted approach is to employ technical metaphors, mimicking the visual appearance and functionality of physical artifacts from the real world that could be used to solve a similar task and letting the user manipulate their virtual counterparts in the model world. This approach not only runs the risk of limiting the capabilities of the computer to what physical—and sometimes very old—technologies can provide [Halasz and Moran, 1982; Ramos and Balakrishnan, 2003; Arment, 2010]; it also limits the users to think about their problems in terms of these tools—a number of good examples can be found in [Mathis, 2010]—, and, most importantly, it does not necessarily establish a direct manipulation system in a narrower sense. On the contrary, these interfaces often sacrifice articulatory directness to keep up the metaphor (Figure 2.8).

Technical metaphors are often linked to traditional physical tools that support a task suboptimally.

Interfaces that are modeled after technical metaphors only represent the conceptual layer of the technical implementation of a tool for the task instead of the conceptual layer of the task itself (Figure 2.9). The mental model of a task or problem, however, does not need to include traditional physical tools that can be used to solve it: Thinking of organizing one's contacts does not involve images of leather bound address books. Thinking about deleting information is not irrevocably linked to the idea of a paper basket, as it is commonly found in desktop user interfaces. And thinking about visually supporting an oral presentation has nothing to do with a slide transparency. We will come back to this example in chapter 5 "Hybrid Media: Presentation Visuals" (p. 185) and show how such technical metaphors can often precisely counter any advantages that direct manipulation aims to provide.

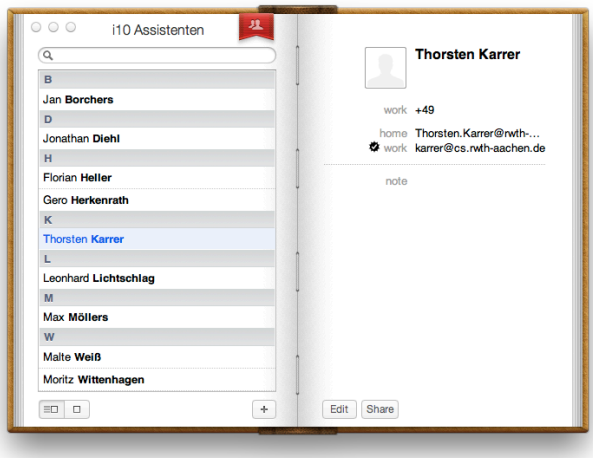

Figure 2.8: The contact list application in Apple's OS X 10.7 adopts the technical metaphor of a physical address book. As a result, the interface breaks the visual and interaction consistency of the platform. It also requires the user to unnecessarily re-adopt interaction techniques that are linked to the physical origin of the metaphor: to change the current group of contacts, the user has to turn the pages using the red book mark. This widens the articulatory distance compared to older versions of the application that did not adopt the metaphor and displayed a group selection list next to the contact list.

Applying direct manipulation to abstract values does not reduce the semantic distance in an interface.

The second class of interfaces is similar in that these interfaces are not representing objects of interest from the users' mental model on their conceptual layer but usually abstract objects that have their roots in technical implementations. In contrast to metaphor-based interfaces, however, these objects are not skeuomorphisms in the sense that they do not mimic any physical embodiment of tools but represent syntactic or meta objects. A typical example of such an interface is a slider to control the value of some variable: The slider may be understood as a direct manipulation interface for the *value as a mathematical concept* and, as such, is appropriate if the underlying semantic concept is either purely numeric or the interface is used for a closed-loop control task only. A volume slider would be a good example for the latter. If the numeric value of the slider is used to represent a more complex semantic concept, e.g., the position in a piece of music, the interface does not help to reduce the semantic distance. In the four layer model, these inter-

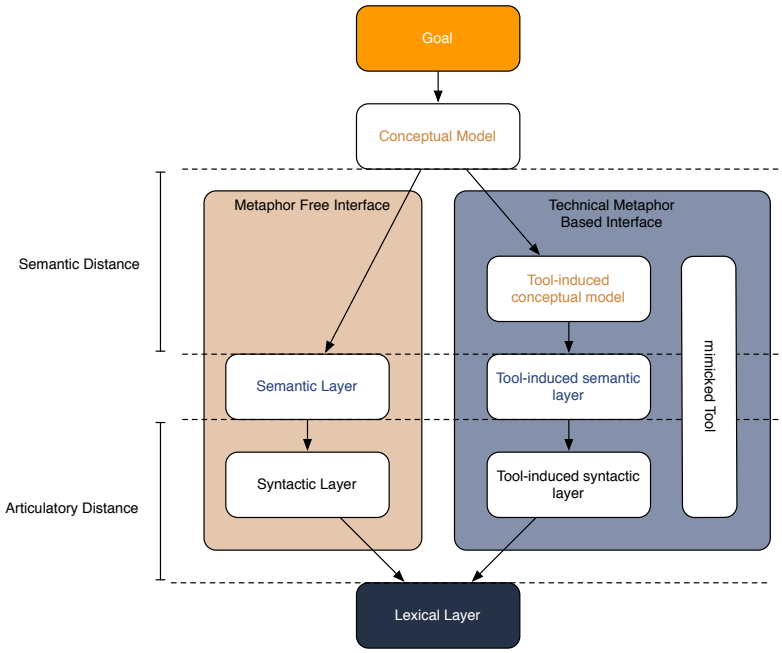

Figure 2.9: Interfaces that adopt technical metaphors (right path) may require users to re-formulate their conceptual model of the task and problem in terms of the represented tool. This may lead to a higher semantic distance than a direct representation of the original conceptual model in the semantic layer (left path). At the same time, the possibilities for interface design and choice of interaction techniques are narrowed down to what fits the metaphor, potentially excluding designs with a smaller articulatory distance.

faces cannot span the whole layer stack; they only replace symbolic commands on the syntactic layer with continuous input methods or, in other words, switch from linguistic input to spatial input (cf. [Foley et al., 1997]) and remain essentially conversation-based but not direct manipulation interfaces (Figure 2.10).

These interfaces can reduce the syntactic distance, but the results of actions may become difficult to predict.

Such interfaces are nevertheless very useful in many situations, and they do effectively reduce the articulatory distance between the intention to quantify an abstract value and the action of doing so; de-coupling the interface from the semantic layer in this way, however, comes at a cost: The range of the value and the granularity of its control can only be determined if the semantic concept behind the

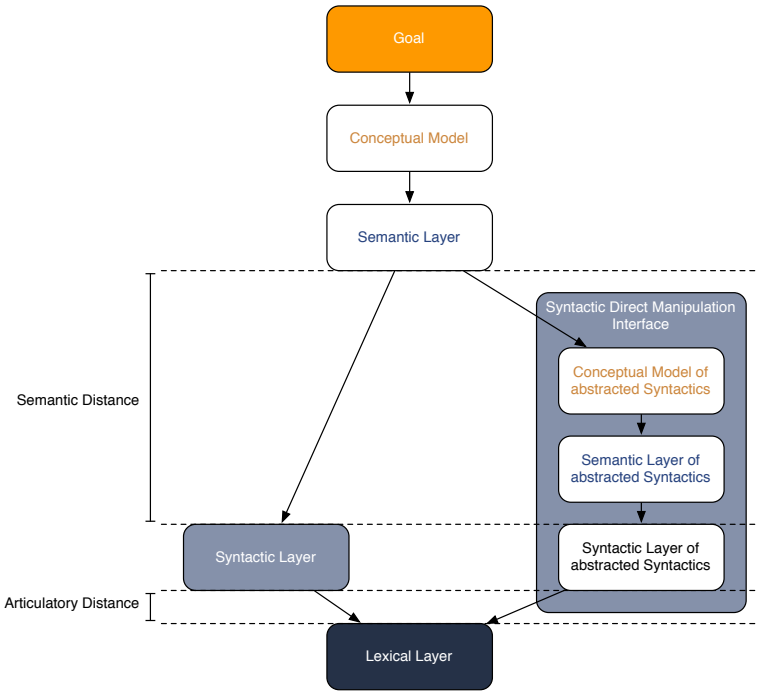

Figure 2.10: Interfaces that apply the concept of direct manipulation only to the syntactic layer (right path) may benefit from a reduced articulatory distance. The potential of direct manipulation to reduce the semantic distance, however, remains unexploited.

value is available. Also, while higher dimensional semantic concepts may be mathematically mapped to a lower-dimensional numeric representation, the predictability of such interfaces can be much worse than what a real direct manipulation interface may be able to offer. Of course, many common media navigation interfaces are designed this way—especially those for time-based media where a slider is used to manipulate the syntactic value of the current presentation time. In chapters 3 “Time-based Media: Orchestral Music” (p. 65) and 4 “Time-based Media: Video Scenes” (p. 89), we will examine such interfaces and present alternatives that allow direct manipulation navigation on the semantic layer.

2.2 A Model for Digital Media Navigation Interfaces

Many successful interfaces and interaction techniques can be described in the terms of the theories above. Mapping the interface language to one of the layer models or analyzing the interaction using a model like the seven stages of action usually gives us a very clear idea about which tasks are supported at which semantic level, which of the gulfs of execution and evaluation are bridged by the system and which have to be crossed by the user, and what kind of conceptual model is suitable for interacting with the system. In this light, it is surprising that out of the group of standard media navigation interfaces so many convey a feeling of clumsy and indirect interaction. To analyze why this is the case and to develop a different way of designing these interfaces, we propose a new model of media navigation interfaces that combines parts of the existing theories with our ideas from the introductory chapter about the internal structure of the media themselves.

2.2.1 Interface Model and Design Space

We use a simple three-layer interface model.

For our design space, we adapt the scheme put forward by Hutchins et al. [1985] by de-factoring one of its axes and replacing the other. A simple three-layer construct similar to the layer models discussed above serves as the underlying interface model (Figure 2.11).

Describing the interface as a stack of *conceptual*, *semantic*, and *syntactic* layers with everything related to the lexical layer being factored into the syntactic layer is sufficient for our purposes and allows a more concise discussion; the lower layers can, however, be split if necessary—for example, for the analysis of the effects of different input devices on media navigation.

In this way, we classify media navigation interfaces by their *semantic distance*, *syntactic distance*, and *control granularity*; the resulting space is depicted in Figure 2.12.

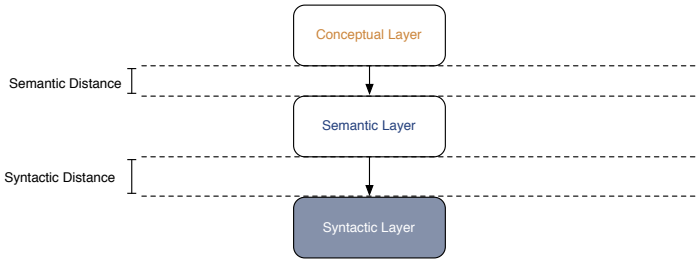

Figure 2.11: We build our *interface model* for media navigation on a simple three-layer variation of the model by Foley et al. [1997].

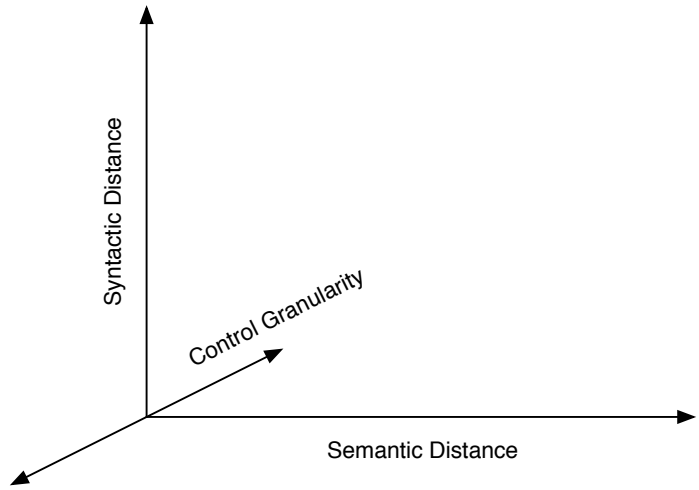

Figure 2.12: Our design space for media navigation interfaces. In contrast to Hutchins et al. [1985], we distinguish between semantic and syntactic distance in the interface and replace the engagement axis with the control granularity.

- *semantic distance*
  Similar to the direct manipulation design space by Hutchins et al., the semantic distance is the complexity of the transformation from the users' conceptual model of the task to the objects of interest as subjects of the interface language. Thus, the semantic distance is closely correlated with the widths of the gulf of exe-

cution that is located between the goal and the formulation of an intention and the gulf of evaluation when relating the interpreted result to the goal as described in Norman's seven stages of action model.

- *syntactic distance*
  While the semantic and articulatory distances in their sum span only one single axis according to Hutchins et al., we regard them as separate in our design space. As we will see in the chapters that analyze concrete navigation interfaces, a small articulatory distance does not imply a small semantic distance or vice versa. Existing as well as our new interfaces can be described much more precisely if these concepts are kept apart.

  To be more consistent in our nomenclature, however, we call the articulatory distance the *syntactic distance*; it still describes the complexity of the transformation between the objects of interest together with all possible actions on these objects and the syntactic representations of both. In the seven stages of action, this distance constitutes the gulf of execution between the formulation of an intention and the construction of a suitable action sequence and the gulf between having perceived a result and interpreting it.

- *control granularity*
  In contrast to the existing models described above, we extrude the design space along a measure of control granularity. Even with low semantic and syntactic distances, it makes a difference at which granularity any manipulation of the objects of interest can be performed. The criticism about direct manipulation interfaces being ill-suited for tasks that require a certain accuracy is a direct result of many direct manipulation interfaces operating on a granularity that is determined by technical or syntactic factors instead of semantic appropriateness.

  Thus, there is a sweet spot for the control granularity in every interface: If it is too high, each atomic manipulation will further a user's goal only on a small scale, resulting in a large number of cycles through the seven stages of action. If it is too low, there is a danger of overshooting the goal (or the goal being inaccessible altogether), which then requires the formulation of a new intention for every necessary 'correction cy-

> cle' through the seven stages. Formally, the control granularity can be defined as the logarithm of the ratio between the input resolution on the syntactic layer of an interface and its functionally mapped image on the semantic layer. This measure is ideally zero, which happens if one syntactic unit in the input space maps to one semantic unit in the output space.

We do not include engagement, responsiveness or inter-referentiality in our design space because these measures do not represent interesting tradeoffs.

In contrast to the design space by Hutchins et al., we did not specifically include their concept of *engagement*. This is for two reasons: First, engagement as such is very difficult to precisely define, capture, or measure. Parallels have been drawn between engagement in model world interface and rapport in conversational interfaces, and it seems that engagement is rather an emergent than a designed quality of an interface. As such it is not well-suited as an axis in a design space. Second, Hutchins et al. state that engagement as a quality relies on responsiveness of the interface and on the input and output of the interface language being inter-referential. While both of these concepts are technically better suited to serve as axes in a design space because they can be measured, they are not helpful to characterize media navigation interfaces: Responsiveness is a less interesting property because it does not really apply to the interaction design but rather to its implementation. Everything else being equal, an interface that is more responsive is always preferable to one that is less responsive, so we can simply demand that every interface should be as responsive as technically possible. Inter-referential input and output can be seen as a given in navigation interfaces; the output of any action is a change in location in the medium, and one of the inputs of every navigation action is the current position. We thus replaced the engagement axis with an axis measuring the granularity of the navigation actions, which has more discriminative power in the space of navigation interfaces, as we will see later.

2.2.2 Media Model

While for many types of interfaces an interface model as the one introduced above is sufficient, the act of navigating in digital media can usually not be described through the

An interface model alone is not enough to describe the interaction with digital media.

interface model alone. The reason for this is that a majority of the navigation tasks are defined through the content of the medium; such tasks become meaningless if they are to be formulated only in terms of the interface. For example, navigating to the scene in the original Star Wars movie where the Death Star is being blown up is a task that is not directly reflected in the interface model, unless the interface is either designed specifically for this movie or it accepts a conversational description of the navigation goal. Both of these are entirely possible, although they do not reflect the general case: Navigation by video tapestries [Barnes et al., 2010] are one way to combine the media and interface models by using the content of a video for a navigation interface. And performing navigation by giving conversational descriptions of video content—albeit of a much more basic kind—is something we have worked on ourselves, and we will present such a system later in this thesis (see chapter 4.3.2 "Alternative Choices for the Conceptual Model").

We propose a *media model* to represent the form and content of the medium.

The important thing to understand, however, is that the users' conceptual model for the navigation task changes with the semantic content of the medium. Therefore, to completely model such tasks, we need a way to formalize the structure of the medium in a similar way as we did for the interface. This second model, the *media model*, is structurally similar to the interface model; the difference is that its sole purpose is to describe the semantic and syntactic construction of the medium—the content and the form—together with the semantic mapping between the behavior of the semantic entities of the content and the syntactic representation of this behavior (Figure 2.13). In this model, we can now express the ideas from the first chapter (cf. 1.2.2 "Structures of Digital Media") in a formal way.

The syntactic structure consists of the support domain and the sample domain; the content is codified by a mapping between the two.

- *Syntactic Structure*
  The syntactic structure of a medium is defined by two domains: the *sample domain* over which the 'samples', abstract carriers for the content, are defined and the *support domain* along which the samples are arranged. Samples, in turn, can often be again described in this way, thus allowing the syntactic structure to be nested. We can see that any digital medium can be fully represented by a mapping from the support domain into the sample domain. Mathematically, this is

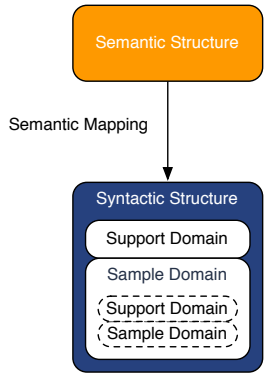

Figure 2.13: Our *media model* reflects the idea that media have both syntactic form and semantic content. Navigation goals of users are usually expressed in terms of the semantic structure of a medium while most navigation interfaces only allow access to its syntactic structure.

the most simple way to describe a medium, and actual implementations of digital media are mostly based on this concept (Figure 2.14 shows the syntactic structure of digital audio):

Images, for example, are defined as color samples—often in an additive color space such as RGB—that are arranged over a two-dimensional spatial support domain.In a video, such images serve as samples and are arranged over a one-dimensional temporal support domain as individual frames. Text documents are represented as groups of characters that form lines from which, in turn, files are composed. And documents for visual presentation support are often represented by a linear stack of two-dimensional slides that can contain text, images, and video.

The syntactic structure depends only on the type of medium, not on the content.

It is important to note that this syntactic structure of a medium is independent of its content. This allows different media of the same type to be handled in the same way, both in terms of implementing access to the medium technically and in designing a user interface that guarantees that every part of the medium is navigable. Consequently, media navigation interfaces that

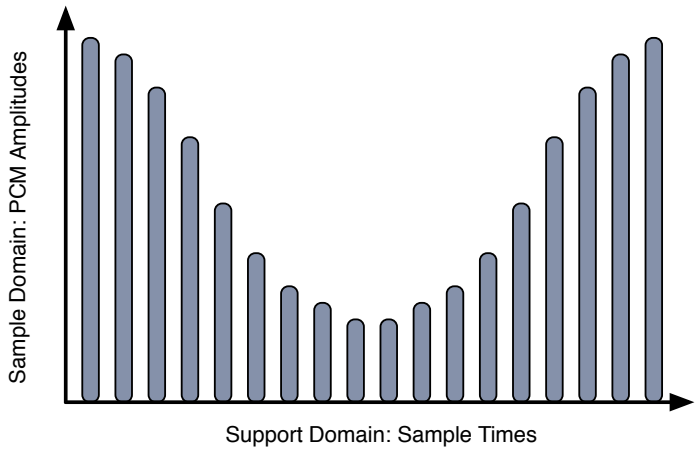

Figure 2.14: The syntactic structure of linear PCM audio is relatively simple. The samples are amplitude values that are mapped to the temporal support domain of sample times (often multiples of $\frac{1}{44100}s$).

are based on the syntactic structure are still what Utterback and Abernathy [1975] called the *dominant design* and thus remain unchallenged for the most part.

The content, together with the context of the user's current task, determines the semantic structure.

- *Semantic Structure*
  The semantic structure of a medium is much less formalized and is defined by the content of the medium and the context in which the medium is being used. It is closely associated with the user's conceptual model of the medium for a given situation and can thus be seen as a space that is spanned by all possible conceptual navigation goals. If, for example, a user navigates through a video that shows a soccer match, and she is interested in finding the frame where a player touches the ball with his hand, the semantic structure would encompass all positions of the ball and could look similar to Figure 1.3. The dimensionality of the semantic structure can be very high, but we will see later that in the context of most navigation tasks a low-dimensional subspace is sufficient to contain most of the navigation goal space. At the same time, not all semantic structures have to span the full medium; often, the goals of navigation tasks only lie in a range of what can be addressed by the supporting domain.

For example, developers who have to understand an unfamiliar piece of existing object oriented source code for maintenance purposes commonly employ a two-phase navigation pattern (cf. 6.2.1 "Analyzing Navigation Behavior"): From a starting point that is possibly related to the problem they are trying to solve, they first follow the call structure of the code in a linear sequence until they find a method that they believe is involved in the problem. Then they change their navigation behavior and start exploring the callers and callees of that method recursively in a depth-first manner [Sillito et al., 2008]. The semantic structure of object oriented source code for maintenance tasks, which we found can be approximated by a subset of the call graph, therefore differs considerably from the syntactic structure of source code in general, which is described in terms of files and code lines.

- *Semantic Mapping*
  As the syntactic structure can fully describe the medium, we can see that any semantic structure must also have a representation in the syntactic space. This means that we can define a mapping (or, more precisely, a relation) from the semantic structure into the syntactic structure, which transforms any semantically expressed navigation goal into a syntactic location in the medium. Such a mapping is needed to facilitate accessing the semantic structure in current technical representations of digital media, and it also gives information about the navigation and access granularity appropriate for the semantic structure.

The semantic mapping associates goals that are expressed in the semantic structure with their position in the syntactic structure.

Obtaining the mapping directly, however, is usually infeasible, because it already requires the semantic structure as the domain of definition. Below, we will propose different ways of first constructing a mapping from the syntactic to the semantic structure and then inverting it, which is much easier (cf. 2 "Generating New Interfaces Using the Combined Model").

Determining the semantic mapping is not straightforward.

2.2.3 Combined Model

We combine our interface and media models.

With the interface model and the media model at our hands, we can now attempt to formulate a combined model for navigation in digital media. This combined model will allow us to describe navigation interfaces in the context of common navigation tasks for different types of digital media; it will also reveal potential mismatches in many widespread media navigation interfaces and suggest approaches to resolve such problems.

Interfaces can have their conceptual layer either associated with the semantic or the syntactic structure of the medium.

The core idea of the model is to associate the conceptual layer of the interface model with the according structural component of the media model. Obviously, two different associations are possible: Either the interface abstraction builds upon the semantic structure of the medium, thus representing in the interface the content-dependent objects of interest relevant for the navigation task (Figure 2.15). Or the interface abstraction is based upon a concept taken from the syntactic structure of the medium and represents this concept—usually a subspace of the support domain—in a more universal, content-independent way (Figure 2.16).

The *real semantic distance* for a task lies between the semantic structure of the medium and the semantic layer of the interface.

This formulation of the model brings three advantages: Firstly, including the media model allows us to determine the semantic distance of the navigation interface much more realistically, because the content and the navigation task is included. We therefore define the *real semantic distance* as the distance between the semantic structure of the medium and the semantic layer of the interface. In the case of syntactic association, this amounts to the sum of the semantic distance of the interface as defined in the interface model and the added semantic distance that results from the cognitive burden for the user of transforming the content dependent navigation goal into a location in the syntactic structure (Figure 2.16).

To allow semantic navigation, we need to transform input between the syntactic layer and the semantic structure.

Secondly, this allows us to judge the technical complexity of the chain of transformations between the syntactic layer of the interface and the syntactic structure of the medium that has to be implemented to build the interface: the concatenation of the syntactic mapping φ_{syn} and possibly the semantic mapping φ_{sem} (Figures 2.17 and 2.18). As for most digital media only the syntactic structure is represented technically,

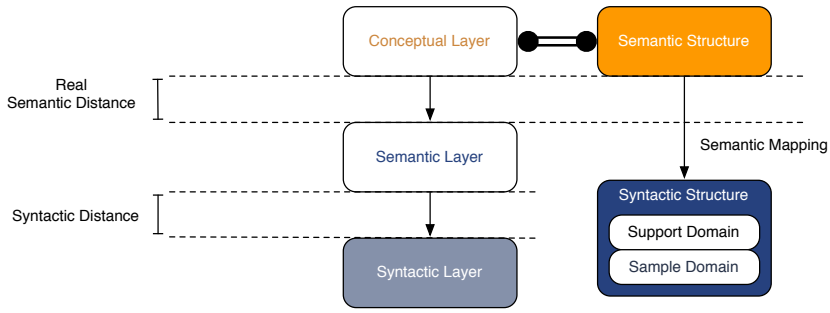

Figure 2.15: Combined media and interface model where the conceptual layer of the interface is associated with the *semantic* structure of the medium. In this case, the semantic layer of the interface is directly concerned with the objects of interest defined by the navigation task and the content of the medium. The semantic distance is determined by how well these objects are represented.

implementing this transformation is necessary to allow navigation through the medium in the first place.

The control granularity is determined by the derivative of this transformation.

Thirdly, since this transformation maps every control input on the syntactic layer to a change in position in both structures of the medium, it also determines the control granularity of the interface. This allows us to analyze the control granularity with respect to the granularity of the semantically relevant behavior of the objects of interest in the medium as defined by its semantic structure.

The combined model shows how the advantages of direct manipulation can be leveraged for semantic media navigation.

Another important characteristic of navigation interfaces can be analyzed with the help of this model. If the semantic layer of the interface is associated with the semantic structure of the media model, the navigation goal and the way to reach it can be made visible in the interface. Following the above argument in the context of direct manipulation (cf. 2.1.4 "Direct Manipulation"), this allows users to perform navigation control gestures much faster, because both input and feedback are directly given in terms of the conceptual model of the navigation, therefore making it immediately clear if an action furthers a user's progress towards the goal

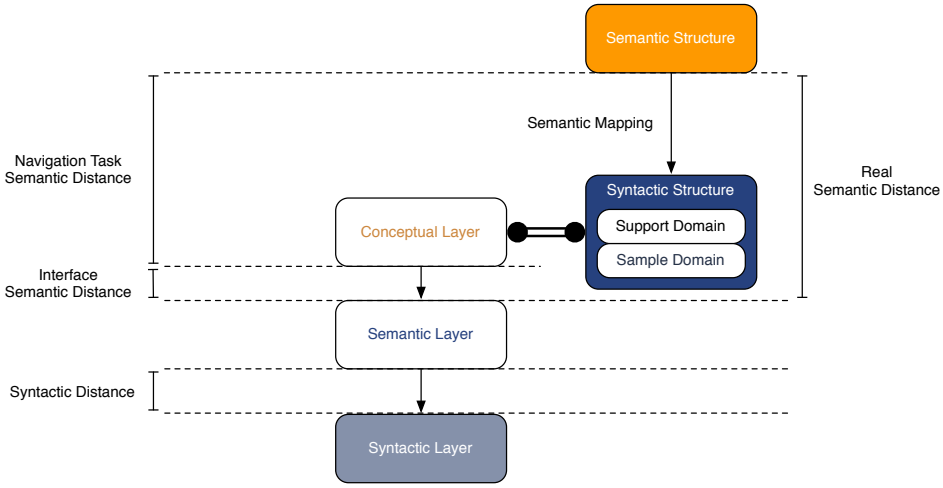

Figure 2.16: Combined media and interface model where the conceptual layer of the interface is associated with the *syntactic* structure of the medium. In this case, the semantic layer of the interface is concerned with the support domain of the syntactic structure in a mostly content-agnostic way. The semantic distance includes not only the semantic distance between the conceptual model of the syntactic structure and its representation in the UI but also the cognitive effort to transform the navigation goal into a location in the syntactic structure in the first place.

or not. If, furthermore, the mapping $I \circ \varphi_{\text{syn}}$ between the input and the effect on the semantic structure of the medium is predictable—ideally, this compound mapping is close to the identity map—, true direct manipulation with all its benefits can be achieved. An optimal point is reached when most navigation tasks directly and without conscious effort transform to very simple directed control gestures similar to a typical Fitts task. Then, the largest part of the navigation can be quickly covered in a quick, open-loop control action—the *ballistic phase*—, and only the last phase of homing onto the navigation goal—the *homing phase*—requires a continuous closed-loop execution and evaluation behavior (cf. [Woodworth, 1899] and [Nieuwenhuizen et al., 2010]).

For navigation interfaces that have their conceptual layer associated with the syntactic structure of the medium, such directness of control is not possible. In contrast, navigation must either follow a trial-and-error pattern where the outcome of each control input has to be newly interpreted and

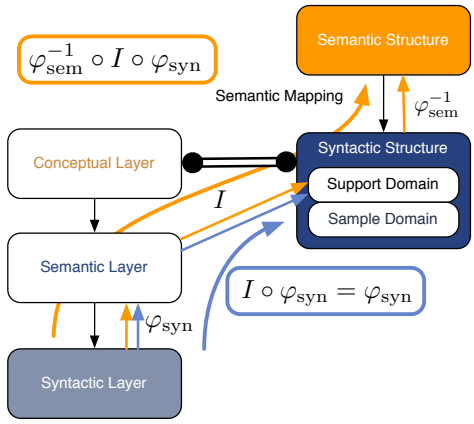

Figure 2.17: Implementation (blue) and mental model (orange) of a navigation technique where the conceptual model of the interface is associated with the syntactic structure of the medium. The implementation is relatively straightforward, because every user action on the syntactic layer is first mapped to the semantic layer and then to the syntactic structure. It is left to the user to figure out how this influences the semantic structure.

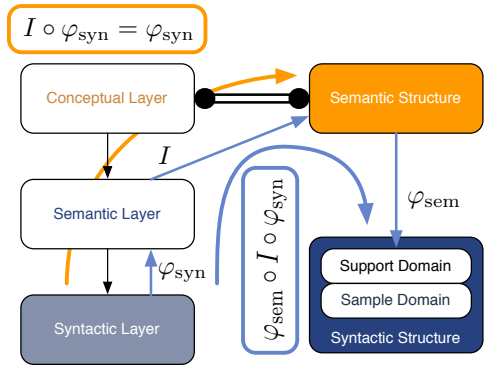

Figure 2.18: Implementation (blue) and mental model (orange) of a navigation technique where the conceptual model of the interface is associated with the semantic structure of the medium. The implementation is more complicated, because it requires a programmatic implementation of the semantic mapping, but the mapping in the user's mental model is much simpler.

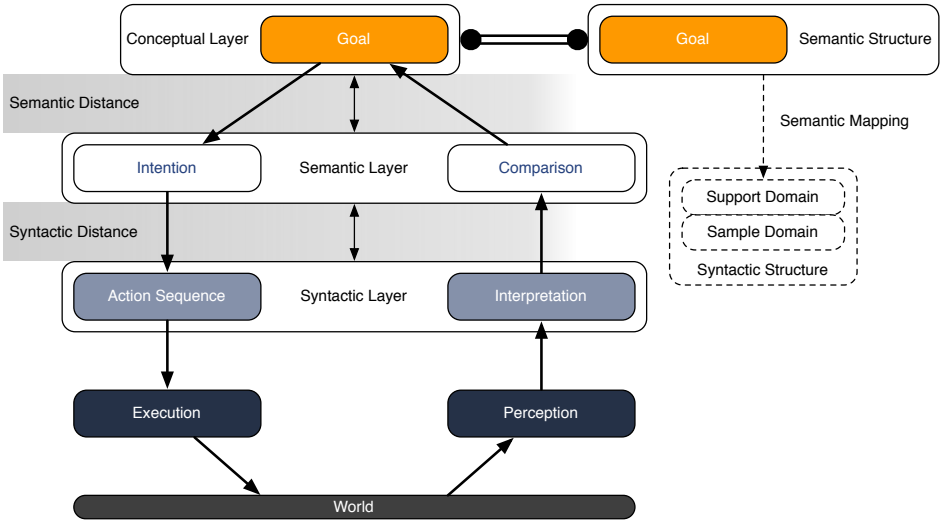

Figure 2.19: If the users' goal can be directly represented in the conceptual layer of a navigation interface, the semantic distance—and consequently the effort for a cycle through the seven stages—can be made small. The actual mechanics of accessing the medium at the right location are transparent to the users and taken care of by the application, which has access to an implementation of the semantic mapping.

evaluated with respect to the goal, and the next control action has to be intelligently guessed. Or, in cases where at least the direction towards the goal is visible, execution and evaluation of the control actions have to be run in a closed-loop fashion for the whole duration of the navigation.

The combined model can be combined with the seven stages of action.

This argument becomes clearer if we overlay the model with the seven stages of action as they are looped through during the interaction with the navigation interface as shown in Figures 2.19 and 2.20. With interfaces for syntactic media navigation, we can see that for every cycle through the seven stages the users have to transform their goal via the semantic mapping into a conceptual model that was derived from the syntactic structure of the medium. If we can design interfaces for semantic media navigation, this step becomes unnecessary, thereby reducing the semantic distance.

As the semantic and syntactic distances in the interface get smaller, so do the gulfs of execution and evaluation. Con-

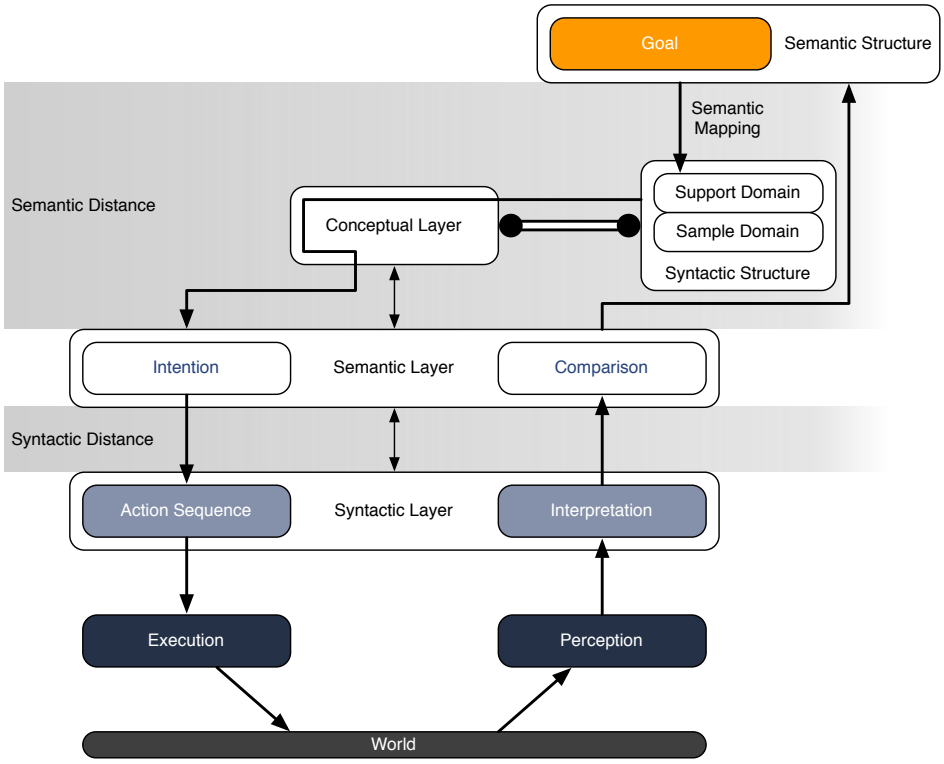

Figure 2.20: If the users can only interact with the syntactic structure of the medium, they have to perform the semantic mapping in the head for every cycle through the seven stages. This widens the semantic distance, thus potentially reducing the frequency of iterations and hurting navigation efficiency.

sequently, we can expect users to be able to predict the coupling between their actions and the resulting progress towards the navigation goal, resulting in a different, more efficient chunking of control gestures (cf. [Buxton, 1995]) like we observe in well-designed direct manipulation interfaces.

With this combined model for media navigation, we can now re-evaluate existing navigation interfaces and refine their position in our interface design space. Different navigation interfaces can thus be compared for different media and navigation tasks. At the same time, the model can help us to generate ideas for navigation interfaces that in many situations are closer to the 'ideal' spot near the origin of the design space.

2.2.4 Descriptive, Comparative, and Generative Power of the Combined Model

We propose our combined model as an *interaction model* for navigation in digital media following the definition for interaction models by Beaudouin-Lafon [2000]:

> "An interaction model is a set of principles, rules and properties that guide the design of an interface. It describes how to combine interaction techniques in a meaningful and consistent way and defines the 'look and feel' of the interaction from the user's perspective. Properties of the interaction model can be used to evaluate specific interaction designs."

Interaction models must have the power to describe both existing and new interfaces, to compare two such descriptions in a qualitative or quantitative way, and to facilitate creating new interfaces or interaction techniques [Beaudouin-Lafon, 2000].

We can use our model to *describe*, *evaluate*, and *create* media navigation interfaces.

For our model, we will first describe the generative aspect, offering a process for designing navigation interfaces that follow our envisioned approach of navigating digital media through their semantic structure and by means of direct manipulation techniques. The following four chapters will then cover the descriptive and comparative aspects of the model: Each will be discussing a different type of medium with different navigation goals, in the context of which we will describe the dominant design of existing interfaces. Then we will use the generative aspect of the model to propose alternative interfaces and compare them to the existing ones. The results—and thus the comparative power of the model—will be verified by extensive experiments and user studies.

Generating New Interfaces Using the Combined Model

In our combined model, generating a navigation interface is divided into four steps. These cover all areas that are im-

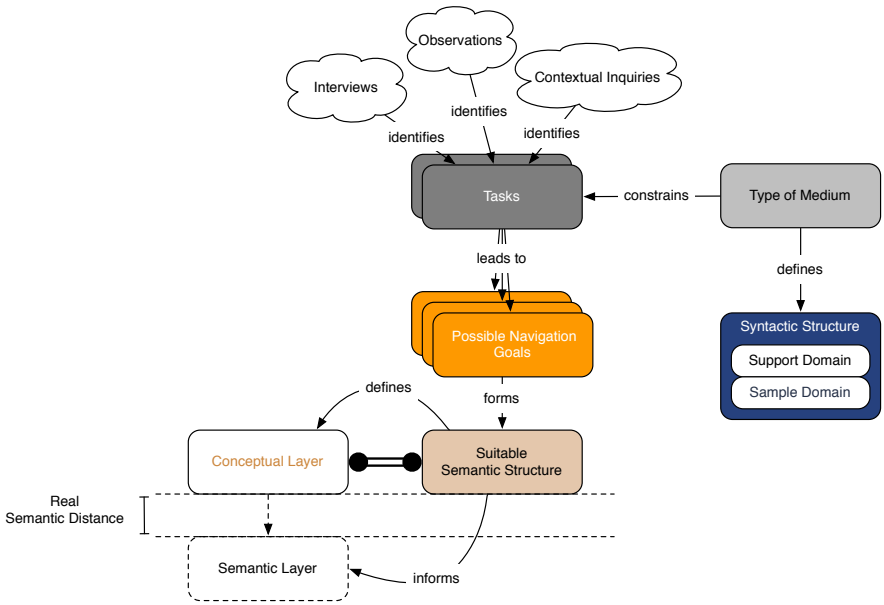

Figure 2.21: First step in creating interfaces for semantic navigation. After the space of possible navigation goals for a class of tasks and a media type has been determined, this space serves as a basis for the semantic structure. This also defines the conceptual layer and informs the design of the semantic layer.

portant for creating an interface, including design, implementation, and analysis related sub-steps.

1. *Find semantic structure and define conceptual layer*
   To find a good conceptual model for the interface that will allow us to achieve a small semantic distance, we first need to know how and if the navigation goals that the interface should support are related to the content of the medium. This leads directly to a definition of the medium's semantic structure for that navigation task; as detailed above, this structure is determined by the combination of all subspaces that contain the possible navigation goals. After the semantic structure has been identified, we derive from it the conceptual layer of the navigation interface and decide which objects of interest will be represented by the semantic layer (Figure 2.21).

The semantic structure is the product of all subspaces that contain possible navigation goals for a task.

This step may be easy if the classes of navigation goals that have to be supported are known and well-understood; it also may even be pre-defined by the character of the application, in which the navigation interface is to be integrated. Examples for such cases are the PERSONAL ORCHESTRA and DRAGON interfaces, which will be described in the next chapters (3 "Time-based Media: Orchestral Music" (p. 65) and 4 "Time-based Media: Video Scenes" (p. 89)).

In other cases, however, a considerable amount of research, user interviews, or contextual inquiries may be necessary to identify established navigation patterns and classify them into a small number of abstract goals from which the semantic structure can be determined. This approach has been taken with the FLY and STACKSPLORER interfaces described in chapters 5 "Hybrid Media: Presentation Visuals" (p. 185) and 6 "Non-time-based Media: Source Code" (p. 271).

2. *Determine and implement the semantic mapping*

The semantic mapping often only takes the form of a relation and is derived from its inverse.

For the implementation of the interface, it is necessary to define the mapping φ_{sem} between the semantic structure and the syntactic structure of the medium. This task is non-trivial, because the semantic structure is not necessarily a connected definition domain, and the mapping cannot be expressed in a closed form. The mapping can still be approximated by first constructing the reverse mapping $\varphi_{\mathrm{sem}}^{-1}$ from syntactic space—which has all the properties necessary for a domain of definition—by finding in the syntactic space the non-linear, potentially discontinuous subset of the navigation goals identified in the first step (Figure 2.22). Reversing this mapping does not always yield a mathematically well-defined function, because one point in the semantic space may be assigned multiple points in the syntactic space. Still, a reverse lookup together with a set of heuristic rules can act as a proxy for the semantic mapping as we will see in the following chapters.

We demonstrate four ways for arriving at the inverse mapping.

Constructing the initial mapping from the syntactic to the semantic structure, unfortunately, may also be difficult. Although it can be generated from the result of a semantic search over the syntactic structure, this requires choosing a suitable semantic query language and implementing the search algorithm. In this thesis,

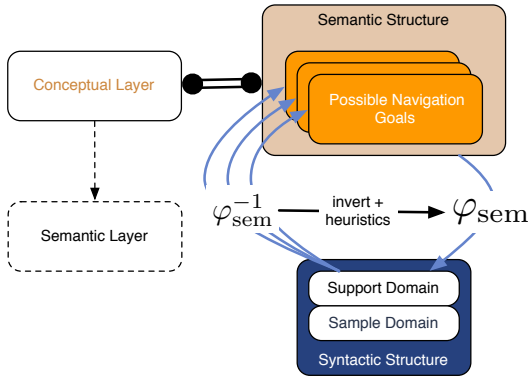

Figure 2.22: Second step in creating interfaces for semantic navigation. Because the semantic mapping can often not be determined directly, we initially define its inverse by finding the syntactic locations of potential semantic navigation targets. This can be done manually or automatically. Although the resulting map is generally singular, an inverse relation can usually be defined through heuristics and interpolation.

we will therefore demonstrate four alternatives for acquiring the initial mapping in different ways:

(a) *Manual definition of the initial mapping as external part of the media authoring process*
If the media content is not authored by the user, it may be feasible to have the initial mapping constructed manually by and expert and added as metadata to the medium in an extra step. Especially for prototypes or for systems tailored to specific content, such as kiosk systems or interactive exhibits, this can be an efficient approach. In our PERSONAL ORCHESTRA conducting simulator (cf. 3 "Time-based Media: Orchestral Music"), the mapping from the syntactic to the semantic structure of orchestras music is manually defined in this way.

(b) *Manual definition of the initial mapping as integral part of the media authoring process*
If the content is authored by the user, and compatibility to legacy formats is not required, the

mapping between the syntactic structure and the semantic structure of a medium can be explicitly made part of or can be derived transparently from the authoring process. The order, timing, and chunking of content during authoring provides strong cues about the semantic structure, which can later be exploited for navigation. As an example, our FLY presentation system (cf. 5 "Hybrid Media: Presentation Visuals") affords presentation authors to semantically cluster the visual representation of the content; this leads to a simple mapping between syntactic and semantic structure, which become easily navigable.

(c) *Automatic generation of the initial mapping through approximate extraction of the semantic structure*
In cases where manual identification of the semantic structure is not possible or feasible, the initial mapping has to be estimated automatically. For this, we have to rely on algorithmic approaches to extract the relevant parts of the semantic structure; such algorithms are the focus of related fields, for example, computer vision or linguistic modeling. Our video navigation interface, DRAGON (cf. 4 "Time-based Media: Video Scenes"), uses computer vision methods to provide ad-hoc estimates of the semantic mapping according to different navigation goals of the user.

(d) *Automatic generation of the initial mapping in semantically structured media*
For some types of media, the syntactic structure is built up in a way that allows to directly derive different semantic structures. This is often the case for media in which the content acts as linguistic descriptions of other higher-level content or computational processes: examples are layout description documents, source code, or certain types of structured vector graphics. Semantic mappings for these media can be generated automatically and with higher fidelity than with the approximation approach described above. We demonstrate this method in our source code navigation interface, STACKSPLORER (cf. 6 "Non-time-based Media: Source Code").

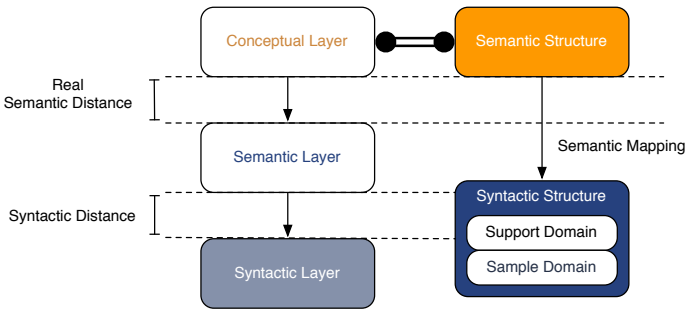

Figure 2.23: Third step in creating interfaces for semantic navigation. The semantic layer is fleshed out, and a suitable syntactic representation of the interface is developed. This step is mostly 'classic' interface and interaction design and has influence on both the semantic and syntactic distances of the navigation interface.

3. *Design the user interface*
   This step comprises the usual design process of the visible part of the interface. While the semantic layer has already been partly determined by the steps above, one still has to find a way to create interface elements and interaction techniques on the syntactic layer that have a small syntactic distance. Other important points are guaranteeing a high feedback rate, finding good representations for the semantic objects of interest, and presenting obvious affordances for the navigation (Figure 2.23).

   For semantic navigation, the interface design is a central component.

4. *Evaluate the interface*
   While our interaction model allows a comparison—in terms of the location in the design space—between different navigation interfaces in the context of the medium and the navigation goals, an evaluation of the interfaces is still necessary. It not only serves as a verification of the advantages of a new interface but may also reveal new navigation strategies that emerge under the influence of the changed interaction. These emergent user behaviors then, in turn,

   Semantic navigation interfaces sometimes give rise to different user behavior.

sometimes give rise to changes in the way the medium is used (cf. 5.3.5 “Fly Case Study—Analyzing Canvas Presentations in the Wild” (p. 249) and 6.2.3 “STACKSPLORER ” (p. 305)).

After proposing our interaction model for navigation in digital media and describing its generative aspect for the creation of navigation interfaces, we will now demonstrate the descriptive and comparative aspects in the context of four different types of digital media. The following two chapters are concerned with *orchestral music* and *video scenes*, which are both time-based media that are commonly navigated either by rate-based controls or a linear timeline slider. The remaining two chapters describe the application of the model to *visual presentation support documents* and *Objective-C source code* where the established navigation paradigms are rooted in the technical metaphors of slides for the former and files and lines of text for the latter type of medium. Each of the four examples is an example not only for a different way to acquire the semantic mapping as described above, but each also lays emphasis on one of the four steps needed to create these interfaces (Figure 2.24).

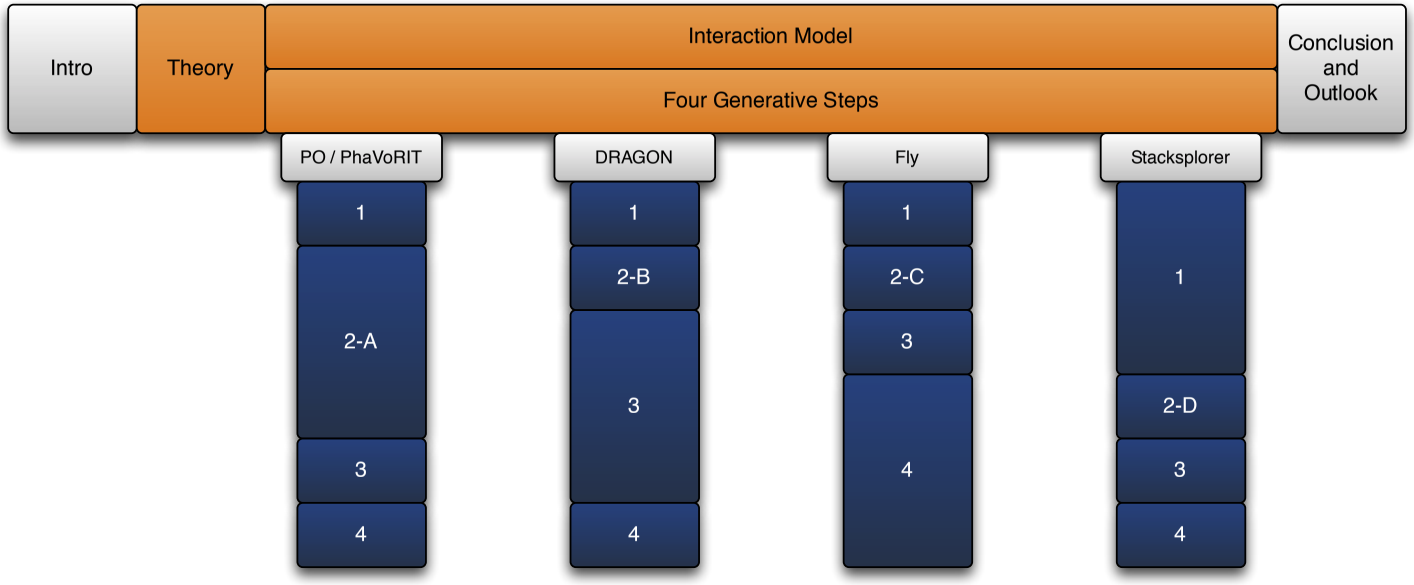

Figure 2.24: Structure of the thesis. Our proposed interaction model for semantic media navigation is supported by four different projects that cover four different types of digital media: PERSONAL ORCHESTRA—music, DRAGON—video, FLY—presentation visuals, and STACKSPLORER—source code. These projects are discussed each in one chapter, and each discussion is focused on one of the four steps of our model for generating semantic navigation interfaces.

Chapter 3

Time-based Media: Orchestral Music

A beat-based semantic structure for music is isomorphic to the syntactic structure.

The first example for a semantic navigation interface for digital media is concerned with music and one of the simplest semantic structures in this type of medium: the beat or *meter* of a musical piece. We will see that even though there exists an isomorphic mapping between the temporal syntactic structure of digital music and the continuous representation of the musical timing of the meter, both technical and design challenges arise when developing such semantic music navigation interfaces.

This chapter will be shorter than the ones that follow, serving primarily as a demonstration of the applicability of our model presented in the last chapter (2.2.3 "Combined Model"). As part of the research that led to this particular kind of audio interface has already been published in the context of the author's Diploma Thesis [Karrer, 2005], several publications [Karrer et al., 2006; Lee et al., 2006a, 2007], and Eric Lee's dissertation [2007], some aspects of the interface and the underlying system will only be presented in summarized form.

After a short excursus on the properties and the structure of digital music, we describe the navigation task the system was developed for. We then describe two existing nav-

igation interfaces in the framework of our model before showing how the development of a new interface fits into the four steps proposed above 2.2.4 "Generating New Interfaces Using the Combined Model". At the end of the chapter, we compare the models for the existing interfaces and the new one.

EXCURSUS: DIGITAL AUDIO:
Technical Representation
Sans any encoding for compression or error correction, digital music usually is represented as a file containing one of multiple audio channels. Each of these channels contains an ordered list of samples, scalar values interpreted as amplitudes. Also, the representation includes the sample rate, a scalar value defining the linear mapping from time to sample index.

Syntactic Structure
The support domain of the syntactic structure for any kind of digital audio is time, which is why audio is a time-based medium. As such, each point in time is directly linked to a sample number with the sample rate as the connecting factor. In contrast to some other time-based media, however, a single audio sample cannot be meaningfully perceived; audio feedback can therefore only be given by playing a continuous range of samples.

Semantic Structures
Possible semantic structure for audio whose content is music are, for example, the musical structure of the piece, the meter, the turn-taking of different instruments, changes in key, or even the mood conveyed by the music. Other kinds of digital audio, for example, recorded speech could have semantic structures including the spoken words, roles in a dialogue, or even higher level structures, such as the literary and argumentative structures of a read out essay or report.

3.1 Describing Audio Navigation in Standard Interfaces

Apart from the timeline slider, navigation in music still works like it did with tape recorders.

Digital music is a medium that, in terms of navigation, has not changed very much in the last decades. As the dominant interaction context is linear consumption of a piece of music, we are still using the stereotypical 'tape recorder' controls (cf. Figure 1.1): a *play* button to start playback, either a *stop* or a *pause* button (or both) to suspend playback

completely or temporarily, and buttons that manipulate the temporal position in the audio by moving forward or backward in time—usually at a higher rate than the standard playback rate. The only new element commonly found is a *timeline slider*, which is an abstract representation of moving the tape manually to a different position. Of course, interfaces for non-linear editing (NLE) of audio offer more functionality and often employ advanced navigation facilities, but we will concentrate on playing back music for now.

Simple play and pause navigation is good for passive listening but otherwise limited.

An interesting observation is that once the *play* function of an audio player software is invoked, navigation through the music is taken out of the user's hands and handled automatically: A fixed playback rate is being set, and the audio samples are read, processed, and sent to the speakers at that rate. On a *stop* or *pause* command, this rate is simply set to zero. This is being done in a universal way for all pieces of digital audio, independent of the content. For the task of simply listening to a piece of music, such an interface is completely adequate and easy to understand and use; for other tasks, an interface that is so closely linked to the technical representation of the medium may pose problems.

Let us first analyze the standard task of listening to a piece of music. The user's semantic understanding of the medium in the context of this task may well be described as the audio just being played back or not. Therefore, the model of the interface could look roughly like shown in Figure 3.1, and the resulting spot of a standard media player interface would be as shown in Figure 3.2.

More expressive control over how we move through a piece of music would require an interface that is less tied to the syntactic structure.

If the user's task, however, involves having more control over the playback of the music, this interface is unsuited. Actively controlling the temporal aspects of the expression of classical music, for example, like a conductor does, fundamentally changes the user's conceptual model and thus the semantic structure of the medium. The content, how the notes in the music are arranged temporally, becomes important—a structure that is much less rigidly connected to the list of samples than could be expressed by a single scalar factor. A conductor basically *navigates* through a piece of orchestral music by deliberately placing the individual beats of the meter in time; the music temporally evolves according to this series of navigation commands.

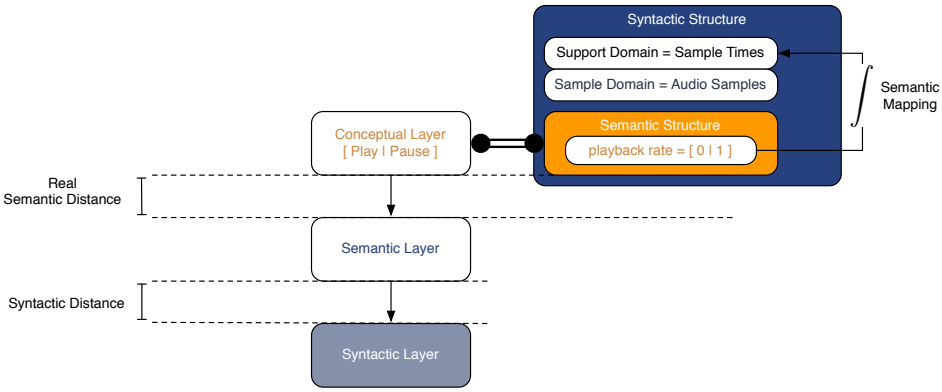

Figure 3.1: Combined navigation interface model of a minimal audio player interface in the context of a standard listening task. The conceptual layer of the interface is associated with the semantic structure, which is part of the syntactic structure. Thus, the semantic mapping is a simple fixed function—in this case an integration that constantly increases the sample time using the current playback rate.

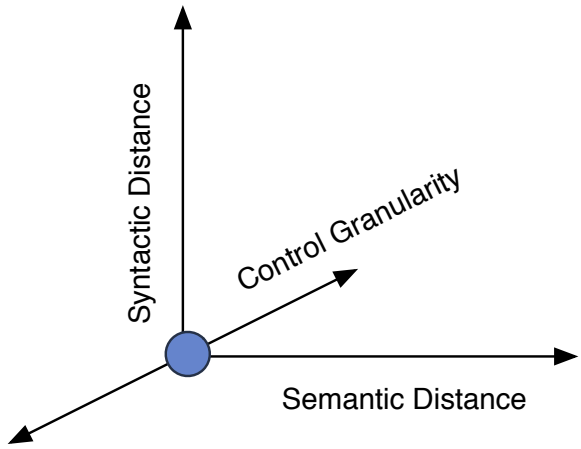

Figure 3.2: Position in the design space of media navigation interfaces of a minimal audio player interface in the context of a standard listening task. For this task, the interface is close to the sweet spot.

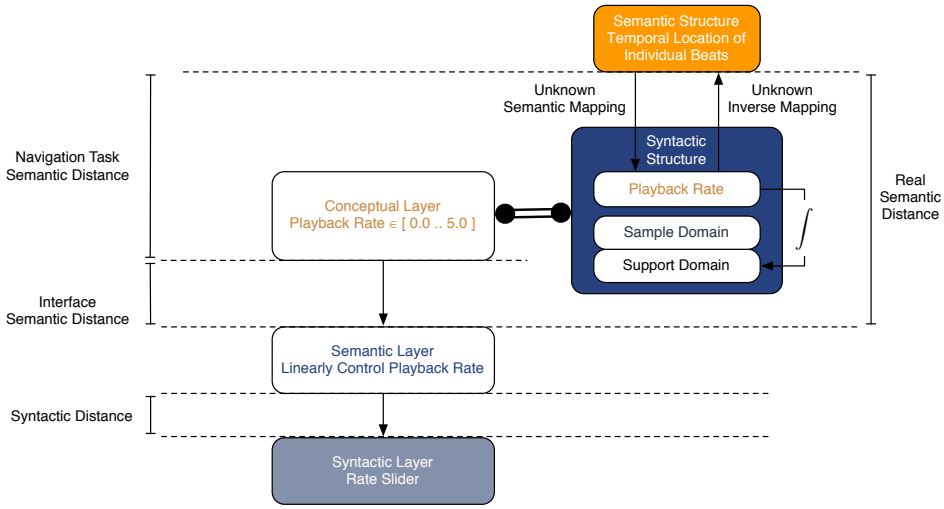

Figure 3.3: Combined navigation interface model of an audio player interface with rate control in the context of a conducting task. To conduct individual beats of the music, a user would have to control the playback rate in a way that requires knowledge of the inverse of the semantic mapping. Even if the syntactic location of every beat in the piece and, therefore, the semantic mapping was known—which is usually not the case—, it would have to be inverted on-the-fly in the user's head, creating an immense semantic distance. Also manually controlling a rate slider in this way would be challenging.

This task is impossible to solve using just play and pause buttons. Even if we designed an interface where the playback rate could be adjusted on-the-fly, the underlying problem would remain: the interface operates on the linear temporal support domain of the syntactic structure, but the object of interest for the task is the semantic time base of the music. This fact makes the combined model of any interface that operates only on the syntactic time—even if we allow higher degrees of control, such as changing the playback rate on-the-fly or using the timeline slider—look quite different for such a task (Figure 3.3).

Directly controlling the meter of the music using rate-based controls is difficult.

On the conceptual layer of the interface, only the playback rate of the medium and thus part of its syntactic structure is represented; the extra transformations between the syntactic and semantic structures of the medium have to be performed by the user both for execution and for interpretation of navigation commands, resulting in an increase in seman-

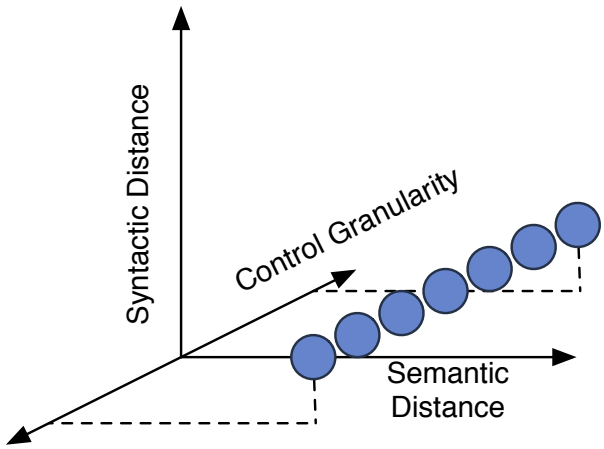

Figure 3.4: Position in the design space of media navigation interfaces of an audio player interface with playback rate control in the context of a semantic navigation task, e.g., conducting.

tic distance (Figure 3.4). Even if the syntactic distance of the interface was really small, because we found an excellent way of representing and manipulating the playback rate, the combined semantic and syntactic distances would still be large and the interface difficult to use for the task.

We can argue that interfaces that operate on the syntactic structure of digital media are therefore not really direct manipulation interfaces (cf. 2.1.4 "Edge Cases and Exceptions") in the context of semantic tasks like the conducting example, because they do not allow users to act on the objects of interest. A thorough analysis of these differences between the semantic and syntactic concepts of time in digital audio and how they influence the interaction can also be found in [Lee, 2007].

3.2 Creating an Interface for Expressive Playback of Orchestral Music

Our four-step approach guides us through the generation of a semantic navigation interface.

Following the four-step-approach we have proposed in the last chapter (2.2.4 "Generating New Interfaces Using the Combined Model"), we can walk through the development of an interface for the expressive playback of music that allows interactive navigation through the semantic structure of a musical piece. Our *Personal Orchestra* and *PhaVoRIT* projects—an interactive conducting exhibit, which has been installed at several museums around the world, and an efficient high-quality time-stretching algorithm—will serve as the illustrative examples in this chapter.

3.2.1 Finding a Conceptual Model

The conceptual model for orchestral music is concerned with beats instead of sample timings.

When people think of expressive playback of orchestral music, very few would describe it in terms of absolute time, and even less would talk about the technical abstraction of sample time in a digitally stored piece of audio: "At two minutes and forty seconds into the piece, the next one and a half seconds should actually be played gradually slower as to take one point nine seconds," or "The samples at indices 7056000 through 771750 should have their sample rate linearly decreased from 44100 to 34816." Of course, they both just mean to express the ritardando at the end of the first movement.

A better conceptual model—and therefore a suitable semantic structure of orchestral music given the task—is to represent music as a succession of *beats* [Lee and Borchers, 2005]. This representation transforms the necessary navigation steps into the well-known task of a conductor, the very model of explicit control over how orchestral music is being played expressively. This conceptual model and semantic structure are the basis for the PERSONAL ORCHESTRA [Borchers et al., 2004; Lee et al., 2006a] project, which allows users to interact with recorded orchestral music through the act of conducting instead of manipulating the syntactic temporal structures of time or samples.

3.2.2 Determine the Semantic Mapping

The semantic mapping from beats to sample times is generated by inverting a manually created mapping from sample times to beats.

Since digital music is not directly represented as a series of musical beats and can thus not be controlled by means of this semantic measure, we need to provide a way of mapping beats to sample times. In the theory chapter (2 "Generating New Interfaces Using the Combined Model"), we have discussed four ways to create an initial mapping between the syntactic and the semantic structure in a medium, which can then be inverted to yield the desired semantic mapping. For *Personal Orchestra*, we followed the approach of manually creating this initial mapping (2a "Generating New Interfaces Using the Combined Model"). This was done for several reasons: On the one hand, although algorithmic detection of the meter in a musical piece had already been possible [Goto, 2001; Jensen and Andersen, 2003], these methods were still not perfectly reliable at the time PERSONAL ORCHESTRA was developed—especially in the presence of local temporal changes such as fermatae or ritardandi. These and other variations in the timing or microtiming can, on the other hand, be easily detected and correctly reproduced or annotated by people with musical training. We therefore created a simple software, *BeatTapper* (Figure 3.5), which allows to annotate an audio file with beat markers just by tapping a key alongside the music as it is being played back.

The semantic mapping can, in this case, be represented as a simple table.

The temporal position of each beat marker (green triangles in Figure 3.5) can be individually adjusted before the software inverts this initial mapping and exports a *beat file* containing the semantic mapping—each beat is assigned to a time value or sample index—in tabular form. From this discrete semantic mapping, a dense mapping can be derived by linear interpolation or other low-pass filtering techniques.

Audio is an inherently transient medium, which makes its presentation during navigation more difficult.

Implementing a conducting interface such as PERSONAL ORCHESTRA faces another difficulty that is inherent in the way humans process auditory signals: music and other audio have to be perceived over time—any single instant of the signal is meaningless to us. Because of this, we cannot just provide access to the samples of the music at the times of the beats, but we must always present a succession of samples over time in order to generate observable feedback.

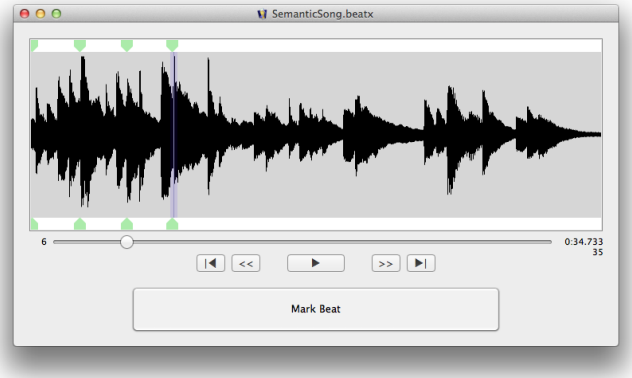

Figure 3.5: Interface of the *Beat Tapper Extreme* software for manually creating semantic mappings between the musical beats and the syntactic time stamps in digital audio. *Beat Tapper Extreme* was written by Eric Lee [2006] and is the successor of the original *Beat Tapper* software created by the author.

In other words, we have to control the rate of the playback in a way that makes sure that beats occur precisely at the points in time where the user wants them to.

Unintended pitch-shifting and error cumulation are challenges when adjusting playback rates.

Interactively adapting the playback rate in this way is, unfortunately, non-trivial because of two problems that occur: First, the playback rate has to be constantly controlled in a closed-loop fashion. Otherwise, prediction errors can accumulate, and the phase difference between the beats in the music and the beats conducted by the user can grow unbounded, defeating the purpose of the interface. Second, the content of audio files is tightly interwoven with the syntactic structure in a way that changing the rate of the playback also changes the pitch of the music. Such implicit modification of the content is, of course, unwanted, and we therefore need a way to properly de-couple the temporal progression of the content from the time base it was recorded in.

A closed-loop control avoids accumulating errors over time.

The first problem was solved by Lee [2006a; 2007], who developed a closed-loop control mechanism for conducting input. In contrast to his original method, however, we

extended the technique to avoid two phenomena that had been observed with early PERSONAL ORCHESTRA prototypes:

- At the beginning of a piece, a 'spiral of death' may occur where the user waits for the music to pick up the tempo and vice versa.
- Until enough beats have been conducted to reliably estimate the tempo, the original control loop exhibits strong ringing.

For the new control, we added a second, open-loop filter on top of the inner closed-loop control that determines the overall tempo v as a weighted affine combination of a predefined standard tempo v_c and the estimate $\hat{v}$ of the closed-loop.

$$v = \alpha \cdot v_c + (1 - \alpha) \cdot \hat{v}$$

Through the parameter α, this mechanism weighs the predefined tempo stronger until the point in time t_{stable} when enough input beats have occurred to stably estimate a tempo. After t_{stable}, it gradually starts phase locking the input and output signals over a timespan τ and finally hands control over to the inner control.

$$\alpha = \begin{cases} 1 & t < t_{\text{stable}} \\ 1 - \frac{1}{\tau}(t - t_{\text{stable}}) & t_{\text{stable}} \leq t < t_{\text{stable}} + \tau \\ 0 & \text{else} \end{cases}$$

Time-stretching algorithms can change the playback rate of audio without altering its pitch.

The second problem of unintended pitch-shifting is more difficult to solve. Separating the user control over the content of digital audio from its syntactic structure requires the use of a *time-stretching algorithm*. The goal of such an algorithm is to re-arrange time-based data on the timeline, thereby changing the speed of the contents' evolution without changing the content in any other way [Karrer, 2005]. Furthermore, for the envisioned task we need a time-stretcher that can produce dynamic changes to the playback rate over a large range of rate factors with high fidelity. Only then can the users place the beats arbitrarily closely or wide apart in time, and the syntactic timeline of the music can be warped to make the user's beats and the beats of the musical piece coincide (Figure 3.6). We developed our own time-stretcher, PHAVORIT, with exactly these goals in mind.

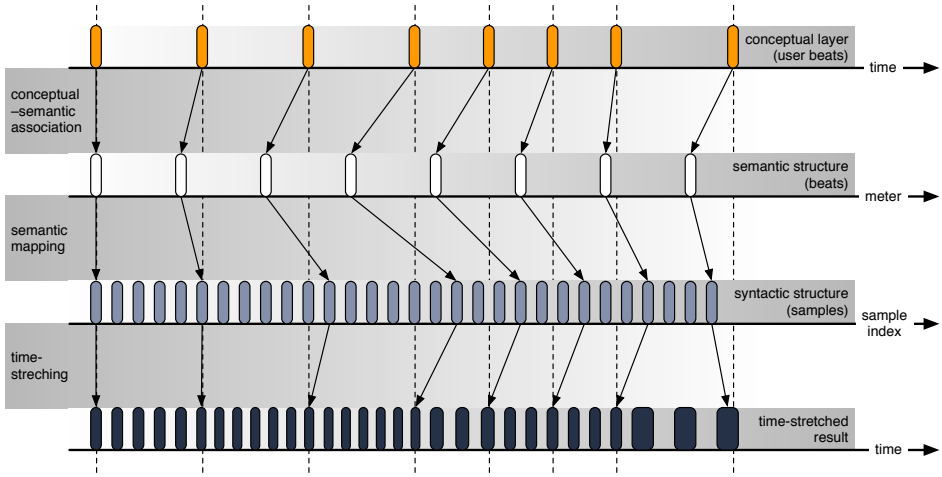

Figure 3.6: Sequence of mapped structures from the conceptual layer down to the warped syntactic structure. Users can place musical beats at arbitrary positions in time through conducting, expressing their goal on the conceptual layer (the actual conducting takes place in the syntactic and semantic layers of the interface and has been omitted here for clarity). The beats are associated with the actual beats of the music in the semantic structure of the medium; the semantic mapping points to the respective samples in the syntactic structure. A time-stretcher can control the playback rate of the music in a way that these samples get played at the intended points in time.

PhaVoRIT: a Phase Vocoder for Real-time Interactive Time-stretching

Phase vocoders produce phase adjusted blocks for each frequency band separately.

PHAVORIT is an extended variant of the well-known *phase vocoder* algorithm [Flanagan and Golden, 1966] and was developed by the author as part of his Diploma Thesis [Karrer, 2005]. A phase vocoder is a time-stretcher that works in the following way: It first divides the sampled audio signal into short overlapping blocks. Then, the partials of each block are extracted by means of a fast Fourier transform (FFT), and the instantaneous frequency of each partial is determined. The knowledge of the instantaneous frequency allows the calculation of each partial's phase at any point during the length of the block. Since the initial phase of each block is also known—and can be modified—, the overlapping blocks can be re-assembled at a different spacing with their initial phases matching the calculated phase of the preceding block at the pasting point (Figure 3.8). The

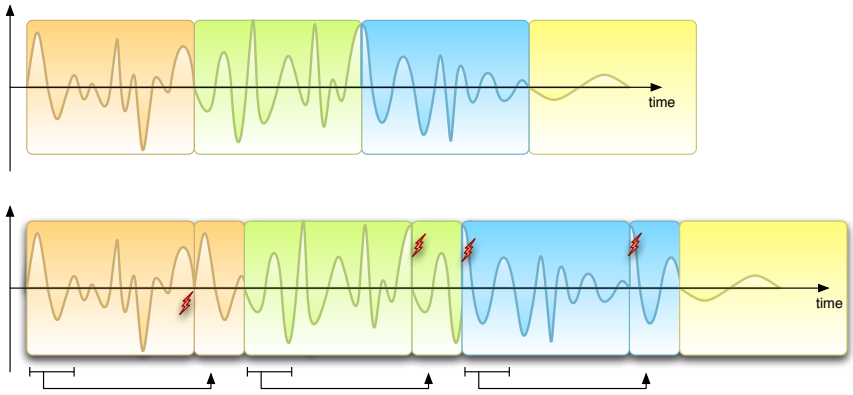

Figure 3.7: Time-stretching by granular synthesis: First, the audio signal is segmented into blocks. The duration of each of these blocks is then shortened or extended by truncation or repetition. Finally, the blocks are reassembled to form the time-stretched signal. This technique is simple and efficient but produces clearly audible artifacts caused by the phase mis-alignments at the block boundaries. Source: [Karrer, 2005].

phase modification eliminates the audible phase jump that would otherwise occur (Figure 3.7), and the altered spacing changes the duration of the two block compound. The latter is equivalent to a change in speed or playback rate, and, since inside a block the frequency spectrum is left unaltered except for the phase, the pitch of the signal stays the same as in the original, non-time-stretched version.

PHAVORIT mitigates the three problems of the traditional phase vocoder algorithm.

For PHAVORIT, we introduced a number of improvements to the basic version of the phase vocoder. They were meant to counter the typical audible artifacts that the algorithm is known to produce:

- *'Transient smearing'* occurs at transient sound events; typical examples include drum shots, note onsets, or burst sounds in speech like hard consonants. These acoustic features are rendered much softer than they should by a phase vocoder, so that drum shots, e.g., become short, slurred noises and miss the directness and sharpness that originally characterized the sound.
- *'Reverberation'* or *'phasiness'* is the undesired effect of a time-stretched signal seeming to be recorded in a very

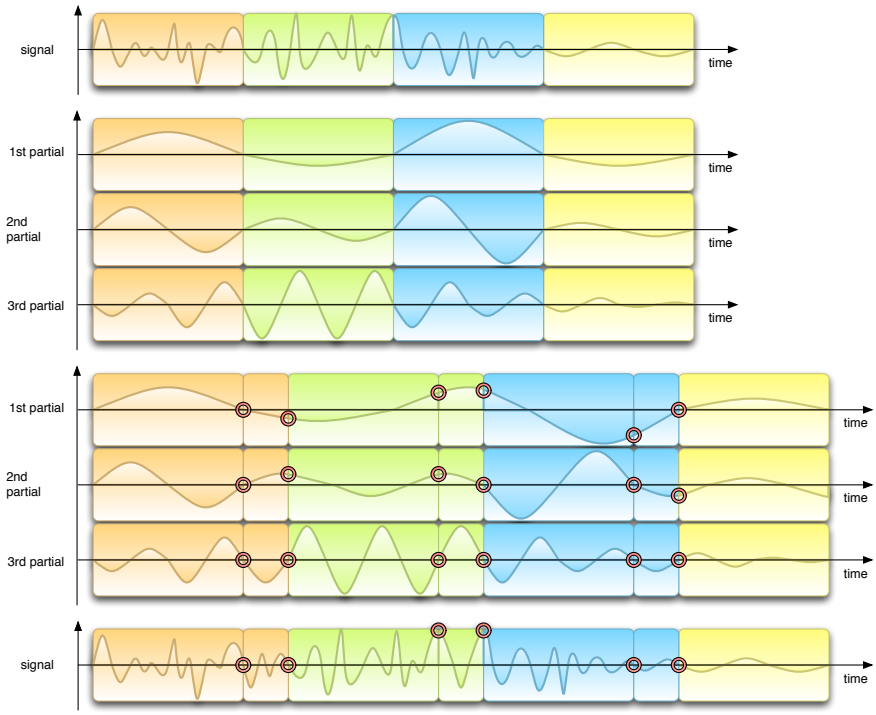

Figure 3.8: Time-stretching using a phase vocoder: In contrast to the simple approach shown in Figure 3.7, a phase vocoder splits the signal into its partials. For each partial, it then re-aligns the starting phases of each block to match the ending phases of the preceding block before summing up the individually time-stretched partials to form the output signal. As a result of this phase alignment, the audible artifacts are greatly reduced. Image from [Karrer, 2005].

small room but, at the same time, at a large distance to the microphone. This 'loss of presence' also causes a spatial recording, i.e., a stereo or surround sound signal, to lose spatial coherence, thus changing its spatial characteristics erratically over time.

- *'Warbling'* artifacts are melodic overtones primarily present in passages where the signal is dominated by a noise component. Examples are recordings of applause at the end of a musical performance or cymbal shots on a drum set.

Phase vocoders assume a signal model that is violated by noisy or transient components.

The reason for these artifacts lies within the fact that the phase vocoder algorithm implicitly assumes a signal model, in which the input is composed of a finite number of sinusoidal, spectrally well separated components (cf. [Flanagan and Golden, 1966]). In reality this is, of course, not the case for any input signal of interest: For example, in orchestral music, the harmonics of dozens of instruments may fall into the same spectral area and overlap. Transients consist of an arbitrarily high, not even necessarily finite, number of sinusoidal components. And noise signals do not satisfy either of the two assumptions. Also, when performing the FFT, one has to carefully choose a point on a tradeoff axis between temporal and spectral resolution: A poor temporal resolution leads to a mis-treatment of transients; their temporal location inside the analysis window becomes difficult to determine, and they may become smeared over several overlapping synthesis windows. A poor spectral resolution leads to a mistreatment of harmonics; the harmonics and center frequencies of distinct signal components may fall into the same frequency 'bin', and the resulting synthesis phase will be incoherent with the phases of the respective components' remaining harmonics.

PHAVORIT limits the influence of noise on the phase of locked frequency band groups by limiting the locking range.

Treating noise dominated signals as a sum of stationary sinusoids in the phase vocoder leads to a systematic phase propagation over time, which destroys the phase randomness that is characteristic for these signals. This is the main cause of the 'warbling' mentioned above. One possible solution to this problem is separating the sinusoidal components of the signal from the noise-dominated ones and treat both signal path ways differently: the stationary sinusoids can be processed by a phase vocoder, the noise can be assigned random phases or have their phases left untouched. Another method is to limit the spectral ranges for phase continuation and phase locking of the phase vocoder to the spectral neighborhood of confirmed sinusoids [Karrer, 2005]. Such solutions, unfortunately, are either very runtime expensive, because they require additional FFTs for multibank filtering, or they are not robust enough to reliably separate noise from steady state sinusoids.

PHAVORIT follows Röbel's approach for treating transient signal components.

Time-stretching transient events to a lower playback rate is already destined to fail from a conceptual point of view. A transient, by its very nature, cannot be slowed down itself—it has to be re-positioned in time and played back

at the original speed to retain its transient quality. The basic phase vocoder algorithm treats a transient event like a number of unrelated sinusoidals, stretching and phase aligning each of them individually. This not only distorts the transient by trying to extend its duration, but it also disperses the transient's location in time over its frequencies. Again, this can be mitigated or even theoretically avoided by detecting transient events in the audio signal, locating them in both the time and frequency domain, and separating these components from the rest of the signal for special processing. Different ways to detect transients have been proposed, among others, in [Levine and III, 1998; Masri and Bateman, 1996; Bonada, 2000; Duxbury et al., 2001; Brossier et al., 2004; Röbel, 2003a]. Methods for transient processing can be found in [Masri and Bateman, 1996; Levine and III, 1998; Bonada, 2000; Hammer, 2001; Röbel, 2003a,b]. We give an extensive overview together with added explanations in [Karrer, 2005]. For PHAVORIT, we attack the problem of transient detection and reconstruction in the time-stretched signal by employing a modified version of Röbel's [2003a] approach. For details we, again, refer to [Karrer, 2005] and to [Karrer et al., 2006].

PHAVORIT uses psycho-acoustics and heuristics to model how tomes move through the spectrum over time.

Polyphonic sounds, even if they are not transient or noise-like, have their timbre and spatial characteristics determined by the relations between the phases of the main frequency and its harmonics. Since the basic phase vocoder has no notion of spectral areas belonging together, these relations are quickly lost over time, causing reverberation and phasiness. Re-establishing vertical phase coherence over parts of the spectrum greatly mitigates this problem, with different approaches proposed by Puckette [1995] and Laroche and Dolson [1997; 1999]. PhaVoRIT extends Laroche's and Dolson's [1999] technique by refining the peak-picking stage of their algorithm and introducing an explicit model for sinusoidal trajectories across the time-frequency domain [Karrer, 2005; Karrer et al., 2006]: *Multi-resolution peak picking* accounts for the non-uniform frequency resolution of the human auditory system. *Sinusoidal trajectory heuristics* constrain the set of spectral peaks that are candidates for sinusoidal trajectory continuation. This allows for a more informed decision in the algorithm as to which frequency areas should be subject to vertical phase coherence re-establishing, thus making synthesis phase propagation along non-meaningful trajectories much

less probable. Also, PHAVORIT detects silent passages, such as pauses in music or speech, to hard-reset the phase accumulators. This automatically re-instantiates the correct phase alignment across the frequency spectrum periodically and effectively reduces phase accumulation errors.

The quality of PHAVORIT over a wide range of music genres was confirmed in a user study.

We performed a controlled study comparing PHAVORIT to five other state-of-the-art time-stretching algorithms: two phase vocoder algorithms and three commercial time-stretchers that rely on different algorithmic approaches. A group of 60 users was recruited to rank the quality of the six different time-stretchers with regard to audio fidelity across six different music genres and six different stretching factors. Although different algorithms performed best on different music genres, only PHAVORIT was consistently ranked among the top three. PHAVORIT thus was the algorithm of choice for all newer versions of the PERSONAL ORCHESTRA interactive conducting system. A detailed description of the time-stretching problem, the phase vocoder algorithm, the theory and implementation of PHAVORIT, and the user study can be found in [Karrer, 2005; Karrer et al., 2006; Lee, 2007].

The PHAVORIT time-stretcher is an example of how the semantic mapping that we determined earlier can be reified in a working implementation. It takes the information from the beat file and uses it to warp the syntactic time on-the-fly, thus allowing direct semantic control over the playback and providing semantic navigation through the medium for our use case. With the media model in place both theoretically and functionally (Figure 3.9), we can now move on to designing a suitable interface.

3.2.3 Designing the User Interface

For the semantic layer we must find a way to interact with musical beats.

The last layers of the user interface language remaining to be designed are the semantic and syntactic layers. As the media model already allows us to interact with the medium through the semantic concept of beats, it seems only natural to choose the objects of interest in the interface accordingly: we can design the interface in a way that it is concerned with musical beats and then only have to find a syntactic representation for this concept that can be easily manipu-

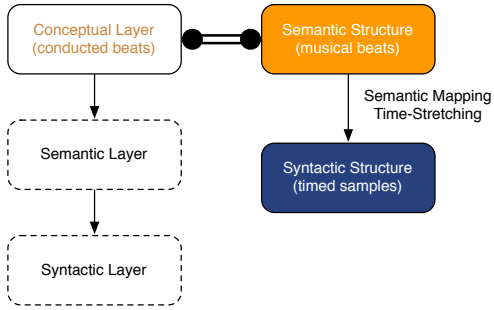

Figure 3.9: Combined interface and media model for semantic navigation in orchestral music up to the second step. At this point, we are still missing the functionality and the design of the syntactic and semantic interface layers.

lated by the user. Just as the task-dependent choice of semantic structure in the medium helps to keep the overall semantic distance in the interface small, a well-chosen syntactic representation of the semantic layer will result in a small syntactic distance.

Conducting is a natural way of specifying beats in orchestral music.

There are, of course, many ways of interacting with music on the level of beats, with different cultures having established different ways that seem natural to them. One mode of interaction with beats in the context of orchestral music that is particularly common in the western cultures is that of *conducting* a piece of music. Although most people are not proficient in the correct procedure of leading an orchestra by means of this language of expressive hand or baton gestures, there is a certain laymen's understanding of how the up-and-down movements should correspond to certain qualities of the music. Intuitively, most people will be able to use this tool to express the tempo, placement of beats, overall volume, and emphasis on a certain group of instruments within the orchestra [Lee et al., 2006b]—not in accordance with the formal rules of conducting (cf. [Rudolf, 1980]), most certainly, but in a way that they feel confident to have communicated their intention.

Following this argument, conducting has been taken up as a mode of interacting with music in audio and audiovisual

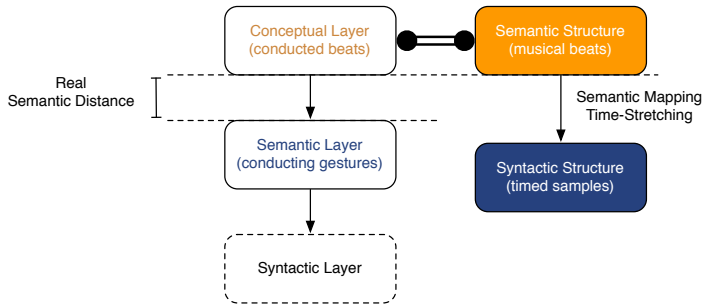

Figure 3.10: Combined interface and media model for conducting interfaces. The actual gesture set used for conducting will be defined in the syntactic layer.

form in a number of interactive systems, for example, [Marrin and Picard, 1998; Borchers, 2001; Lee et al., 2004; Argueta et al., 2009]. These systems take a number of slightly different approaches in lowering the syntactic distance between a musical quality and how it is expressed through conducting gestures. The common ground, however, is that the interface model has the conceptual layer associated with the semantic structure of the medium and that the semantic layer of the interface deals with conducting gestures to represent beats and other expressive qualities of the music to be played (Figure 3.10). For the discussion at hand, we will focus on the PERSONAL ORCHESTRA systems, most of which have been developed at our group.

Personal Orchestra: A Conducting Interface for Semantic Control Over the Temporal Progression of Audio

PO is installed in the HOUSE OF MUSIC in Vienna.

The PERSONAL ORCHESTRA (PO) systems are a series of conducting exhibits that had begun with the installation of PO1 (Figure 3.13) at the house of music in vienna [Borchers et al., 2002]. These exhibits allow visitors to 'conduct' one of a number of world famous symphonic orchestras: Holding a baton, the visitor stands on a conductor's pedestal in front of a large digital video projection showing the orchestra. When she lifts the baton, the musicians in the video take up their instruments and prepare to play. The visitor can then move the baton in an up-and-down conduct-

Figure 3.11: Piece selection menu screen in PERSONAL ORCHESTRA 6. The user touches one of the six partitures to start conducting that piece.

ing gesture, dictating the tempo and volume of the piece or emphasize certain groups of instruments in the ensemble. The digital video and audio streams will be played back and time-stretched to match those gestures, thereby ensuring that the visitor always feels like she really is conducting the orchestra. Additionally, in the current version, the current progression of the piece in the partiture is displayed and continuously updated on a digital note stand in front of the conductor (Figures 3.11 and 3.12).

The system emphasizes discoverability and graceful degradation.

In the design of the original system [Borchers et al., 2002], much care had been taken to ensure easy discoverability of the interface by creating strong conducting affordances. The digital baton, conductor's pedestal, large screen projection of the interactive video showing the orchestra, and the digital note stand with the interactive partiture all contribute to conveying the conceptual model of the interface and thereby further reduce the semantic distance. The syntactic distance is kept low by relaxing the requirements for 'correct' conducting gestures (Figure 3.14); a set of different gesture recognition engines [Gruell, 2005] allows the system to be used with a range of conducting styles, from the classic 4-beat-gesture to simple up-and-down movements or just wiggling the baton in the air at different velocities. As a result, everybody can walk up to the exhibit and start

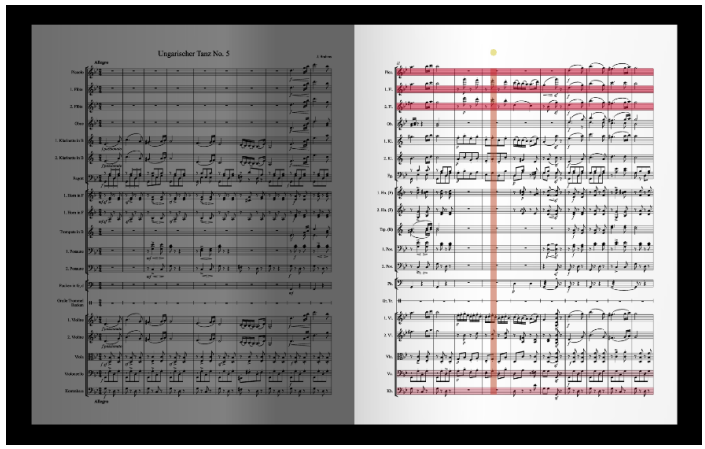

Figure 3.12: Live partiture screen in PERSONAL ORCHESTRA 6. The position and instrument emphasis as it is currently conducted by the user is shown on an interactive note stand.

conducting a virtual orchestra. No training is required to play back the video and audio recordings according to one's individual preferences in musical expression.

To deliver this interactive experience, a number of different technologies must work in concert. The conducting gestures must be captured, processed, and transformed into a series of input vectors consisting of tempo, phase, amplitude, and spatial direction. From this data, the desired point in time for the video and the audio tracks has to be determined. The closed-loop control explained above calculates the best playback rates for both video and audio, using the desired point in time and the actual point in time. These rates are then fed into the respective video and audio time-stretching algorithms, and the resulting streams have to be rendered to the projection screen and to the speakers. Of course, all of the steps must be integrated into a software package that can be operated without difficulties by non experts in the context of a museum or exhibition.

PERSONAL ORCHESTRA has been refined over many iterations.

The implementation of the PERSONAL ORCHESTRA systems has changed greatly over the different existing versions. A schematic of the sixth and current version of the system, dubbed PO6, is shown in Figure 3.15. It includes the ges-

Figure 3.13: PERSONAL ORCHESTRA lets laymen conduct the famous Vienna Philharmonics at the House of Music in Vienna. Visitors can interactively navigate through audio-video recordings of a selection of classical pieces. The temporal position of each beat is indicated by moving a digital baton through a simplified conducting pattern; the size and direction of the gesture influences volume and instrument emphasis as well.

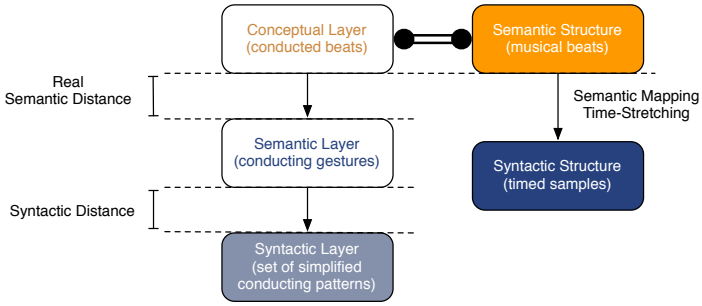

Figure 3.14: PERSONAL ORCHESTRA does not require the users to perform the same conducting patterns as professional conductors do. A simplified set of gestures is recognized just as well, reducing the syntactic distance.

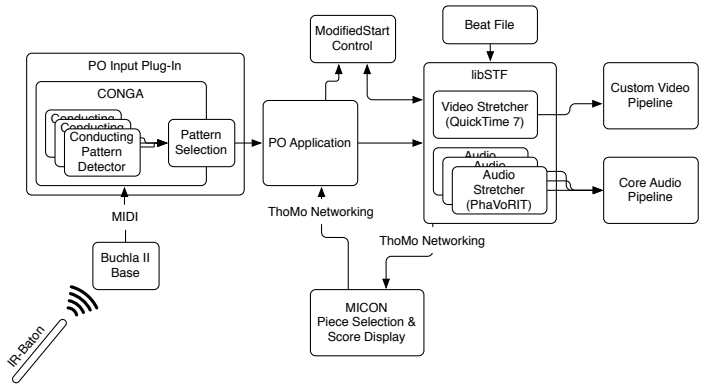

Figure 3.15: Schematic overview of the individual components of PERSONAL ORCHESTRA version 6. The conducting gestures performed by the user are detected, interpreted, and transformed into a sequence of beat timestamps with volume and instrument emphasis. The audio and video streams are then time-stretched so that the temporal progression of the content matches the user input.

ture recognition framework, CONGA, developed by Ingo Grüll [2005] to process the MIDI-encoded position data of the baton, which comes from a Buchla Lightning II unit [Buchla]. The beat propagation and closed-loop control was devised by Eric Lee [2006a] and uses an early version of his libSTF framework [2007] developed for PO3, an earlier PERSONAL ORCHESTRA system. The PO6 version includes modifications to the closed loop by Moritz Wittenhagen, Leonhard Lichtschlag, and the author to allow faster phase locking to the conductor's gestures at the beginning of the performance. High definition video and multi-channel audio can be used with PO6; for this purpose, we employ a number of parallelized PHAVORIT time-stretchers—one for each instrument group—and a custom built video rendering pipeline on top of QuickTime 7. To provide additional feedback to the conductor, PO6 also features MICON, a digital note stand [Borchers et al., 2006] that dynamically displays the orchestral score for the current part of the musical piece (Figures 3.11 and 3.12). The software architecture was developed by the author together with Moritz Wittenhagen and Leonhard Lichtschlag.

In 2009, PO6 has been installed at the House of Music in Vienna, replacing the original PO1 system. Thousands of visitors every year enjoy the opportunity to 'lead the Vienna Philharmonics' and to explore different points in the space of possible ways to conduct a symphonic orchestra.

3.2.4 Evaluate the Interface

The system offers a kind of navigation that timeline or rate controls cannot.

If we compare the standard interfaces for audio navigation from the introduction of this chapter with the conducting interfaces like Personal Orchestra (Figure 3.3 vs. Figure 3.14), it becomes clear why achieving expressive playback of orchestral music with the standard UIs is extremely difficult. Users would have to construct the semantic mapping in their head beforehand to conclude how to interact with the semantic structure of the medium through its syntactic structure. Even if the necessary information would be available, for example in the form of a PO-style beat table, evaluating the mapping in the head would be demanding. Furthermore, the motor control required to either adjust the rate according to this mapping or use a timeline slider directly would be above most people's capabilities [Walther-Franks et al., 2012]. These problems increase the semantic distance using the standard UI up to a point where the task cannot realistically be solved, even given the relatively small syntactic distance between a timeline slider and the concept of syntactic or sample time.

3.2.5 Conclusion

We have shown that for music we can navigate in a semantically more meaningful way by mapping the syntactic domain onto a semantic domain defined in terms of content features—in our case the musical beats of orchestral performances. By following the four steps proposed in 2.2.4 "Generating New Interfaces Using the Combined Model", we explained the rationale behind the design of our conducting interface, PERSONAL ORCHESTRA. The focus of our discussion lay on the second step, the definition of the semantic mapping, and its ramifications regarding the implementation of such systems (cf. Figure 2.24). In this context, we

briefly summarized the theoretical and technological difficulties of time-stretching and our contributions to that field in the form of the PHAVORIT algorithm.

We also compared PERSONAL ORCHESTRA to two established interfaces for digital audio navigation and playback, which constitute the dominant design approach in this domain, and argued that our approach of semantic navigation facilitates navigation tasks that are extremely difficult or even impossible to solve with the current interfaces that are based on content-agnostic syntactic navigation paradigms. This is true even in the case at hand where the semantic and syntactic structures of the medium, both being representations of time, are still isomorphic; we will see examples where the relationship between the structures of the medium are not so well-defined in the following chapters.

PERSONAL ORCHESTRA is only a first step in the direction of semantic media navigation interfaces.

PERSONAL ORCHESTRA, of course, remains a conceptually simple interface in the context of the discussion of media navigation, because the kind of interaction that PO supports is only an edge case: On the one hand, it enables a semantic playback control and thus allows a user to progress through the media in a meaningful way that is related to the contents. On the other hand, the user can only manipulate the linear progression of the contents in a very limited way. Still, this is all that is needed to make a recording of the Vienna Philharmonics follow your lead, and it fits our overall concept of media navigation along semantic domains (the domain here is the axis of all possible ways to conduct a certain orchestral piece). In this regard, PO was our first system to deliberately shift access to the medium away from an existing dominant, syntactic domain.

Chapter 4

Time-based Media: Video Scenes

Our approach to semantic video navigation allows direct manipulation interaction with the actual contents of the scene.

In contrast to our audio use case from the last section, where we re-mapped the syntactic structure of a piece of digital music to the semantic domain of musical beats, our focus in this chapter lies in enhancing video navigation by making a whole class of local semantic subdomains of the medium accessible to the user. This time, we will not expose another form of temporal control over the progression of the video but utilize the video's connection between temporal and spatial information to allow fast, frame accurate navigation by a positional direct manipulation interface: as a result, users can move through the syntactic temporal domain of the video by grabbing objects in the video and drag them along their movement trajectory in the scene. We will see that, even though there are a large number of semantic mappings to establish at any time—one for each object visible in the video scene—and these mappings are much less well-defined in this example, such a semantic navigation interface can make a number of video navigation tasks much more efficient.

After listing the most important characteristics of video as a time-based medium, we describe the navigation task and have a look at a number of existing interfaces, their problems, and solutions proposed in related work. We then

introduce DRAGON, our video navigation system, and explain its design along the lines of our four-step approach (cf. 2.2.4 "Generating New Interfaces Using the Combined Model") for generating semantic navigation systems for digital media. With DRAGON as an example, we define the broader concept of direct manipulation video navigation (DMVN) and discuss its applications and limitations.

In this chapter, we focus on the third step: designing the interface.

This chapter focuses on the third generative step of designing the interface and the interaction for semantic navigation. With less stringently defined semantic mappings, as we will encounter in this case, situations can arise where the mapping offers the wrong semantic granularity for efficient navigation or where it becomes singular, and the navigation concept can break down. We will show how such problems can be mitigated by careful UI design in 4.2.3 "Interactive Scoping and Trajectory Filtering" and dynamic reinterpretation of the semantic context in the interface language in 4.2.3 "Object Pauses".

4.1 Describing Video Navigation in Standard Interfaces

Video navigation interfaces are also often modeled after tape interfaces.

Similar to the audio navigation interfaces described in the last chapter (3.1 "Describing Audio Navigation in Standard Interfaces"), most video navigation interfaces are a direct successor to tape control interfaces. Like mentioned before, these interfaces usually consist of a number of fixed-gear playback rate controls (play, ffwd, rwd, pause) and a timeline slider that simulates directly positioning the playhead over a specified position of the tape.

Existing semantic navigation techniques mostly rely on timeline annotations.

In the context of video editing in general, and today's nonlinear editing (NLE) video software packages in particular, there exist, of course, a multitude of specialized navigation tools that allow to manage collections of multiple clips, navigate between them, and jump to cue points or marked regions in the timeline (Figures 4.1 and 4.2). Such tools, however, all rely on metadata or annotations that have to be added to the video first—if one wants to navigate in a video scene that has not been processed (mostly manually) in such a way, they do not help. Moreover, they behave rather like

EXCURSUS: DIGITAL VIDEO:
Technical Representation
If we strip away any encoding, digital video is usually represented as an ordered list of samples, the video frames. Each frame is an image, and usually all frames in a video share the same size, pixel count, and color depth. Frames may be stored in the form of differential images from neighboring frames for more efficient encoding, but for the discussion at hand, we treat each frame as an individual human readable picture. Like most time-based media, a digital video needs a sample rate value, which is stored together with the list of frames and determines the temporal succession of how the frames are being displayed.
Many container formats for digital video allow the storage of additional video and audio tracks that are mixed down and played back simultaneously in the player software. Here, we are only interested in video as a medium in itself.

Syntactic Structure
The support domain of the syntactic structure for any kind of digital audio is time, which is why video is a time-based medium. As such, each point in time is directly linked to a sample number with the sample rate as the connecting factor. The spatial dimensions of the video frame act as a second syntactic structure, but time is the one usually used for navigation.

Semantic Structures
Video is a rich medium in terms of possible semantic structures: these structures can range from simple constructs like the presence of unknown objects in a scene of a surveillance video to abstract concepts like the social connections between characters in a movie. In this spectrum also live movie plots, positional constellation of objects, color palettes, atmosphere and emotion conveyed through the content, information spaces, and many others depending on the type and genre of the video.

the index in a book in that they can only represent labeled single points in time or time ranges as objects of interest; the time-based nature of the content itself is not reflected. These tools are therefore less suited for ad hoc navigation, and they need other good navigation tools to do the annotations in the first place.

Event-based DOI navigation systems are still a research topic in computer vision.

Other interfaces require no such annotations and use automated approaches, for example computer vision, to generate sets of possible navigation targets according to some degree-of-interest (DOI) function. These methods include domain specific event detection algorithms (e.g., [Ekin et al., 2003; Lv et al., 2006]), detection of cuts and scene changes,

Figure 4.1: Apple's iMovie is a modern non-linear video editing software package. Individual shots or scenes of a longer video clip can be accessed directly, and markers can be assigned to special points in time. Navigation inside a single scene, however, still relies on a scrubbing technique using the playhead in the timeline.

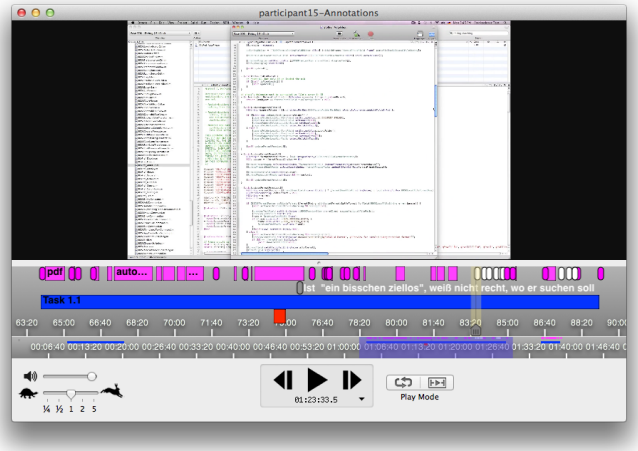

Figure 4.2: ChronoViz [Fouse et al., 2011] is a tool for the annotation of events in time-based data, e.g., videos. Once interesting points have been marked and annotated, they can be easily accessed.

recognition of certain movement patterns [Chen et al., 2004], or even more generic search queries. While such approaches are close to our goal of semantic navigation, very few of these systems have grown past the prototype stage. Apart from cut and scene boundary detection algorithms, most of them are not yet robust enough to be implemented in standard navigation interfaces, and they often require training in specifying the DOI function through query languages or user supplied target patterns.

Even for specialized tasks, the timeline slider is the dominant interface.

For these reasons, most video navigation interfaces in media players or even in NLE software packages are still limited to the standard controls mentioned above. If one is to find the perfect frame for a cut, for the extraction of a still frame at the right moment in the video, or for placing an annotation, the timeline slider is still the way to go. This is true, unfortunately, for most areas where frame-accurate navigation is needed: video forensics, traffic or security surveillance, live analysis of sport events, market research, video editing, or video ethnography.

We will therefore first take a closer look at the timeline slider as the dominant tool for such in-scene video navigation. It is immediately clear that, in our model, this interface must be structurally identical to the timeline slider interface for audio navigation but with a different semantic structure of the medium and thus a different overall semantic distance.

For tasks with syntactic goals, current interfaces seem suitable at first glance.

Again, if the navigation task is defined in terms of the syntactic temporal structure of the medium, the semantic and syntactic structures coincide, and the semantic distance for the timeline slider becomes small as a result: Navigating to a frame that is, for example, four minutes and six seconds into the video is a direct manipulation task with the timeline. This is because what is modified with the slider—the time—is the object of interest in this task (Figure 4.3).

For tasks with semantic goals, current interfaces are less helpful.

Unfortunately, we can easily imagine that most relevant navigation tasks for the application areas mentioned above are seldom formulated in terms of the syntactic structure. If the goal is to analyze what is happening in the video, to demonstrate a process, or to find a specific frame for an event to be annotated, it is normally not possible to directly express this in the abstraction of time and without referencing the content. We will come back to this problem and its

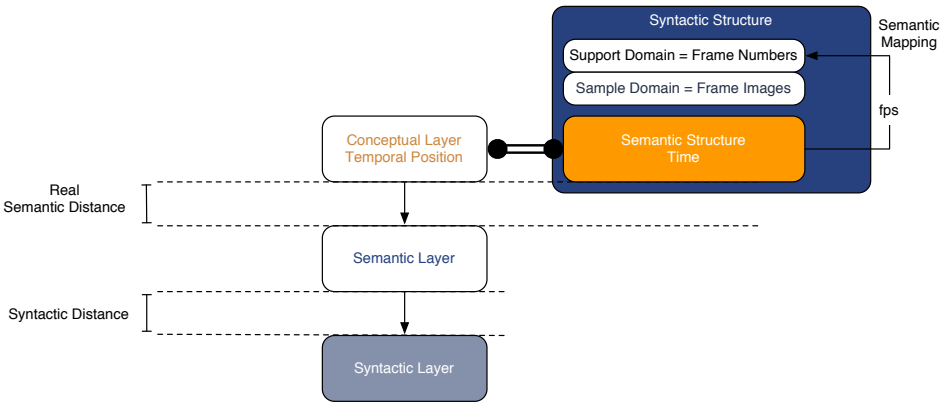

Figure 4.3: Combined navigation interface model of a standard video timeline interface in the context of a temporal navigation task. The conceptual layer of the interface is associated with the semantic structure, which is part of the syntactic structure. Thus, the semantic mapping is a simple fixed function that linearly converts a time description to a frame number.

implications on the semantic distance of timeline slider interfaces later in this section.

4.1.1 Analyzing the Syntactic Distance of Slider-based Navigation and Selection Interfaces

The syntactic distance of timeline slider interfaces can be large because of granularity mismatches.

Before we discuss the semantic distance, it should be noted that even the syntactic distance in slider interfaces as mentioned above can be surprisingly large. Although they are sometimes lauded as the archetype of a direct manipulation interface—which they really only are if the conceptual layer deals with linear values as the objects of interest—timelines and other sliders have several shortcomings. These become evident every time a slider is used for any kind of *selection* task, i.e., a task where a slider is used to pick one element out of an ordered list of unknown length where only the currently selected element or a local neighborhood of this element is visible at the same time. Such tasks comprise navigation in video and audio timelines, where the slider selects one sample over the temporal sample domain, operating scrollbars, where the slider selects one document po-

sition over a spatial domain or a discrete syntactic structure (like pages in a multi-page PDF document), or numeric selection like using a slider for volume control.

The first and most obvious problem is one of the *control granularity* of the interface, which we have already discussed earlier (cf. 2.2.1 "Interface Model and Design Space" (p. 42)). It is obvious that the control granularity of a selection slider depends on the cardinality of the set of elements that the slider selects from: if, for example, the set of elements is larger than the set of positions the slider can assume, the slider is not an accurate selection or navigation interface. Since we are mainly concerned with on-screen representations of sliders, this number is, given a sufficiently expressive input device (cf. [Ballagas, 2007]), equal to the number of pixels the slider occupies. Thus, if the number of frames in a video, for example, surpasses the number of pixels on the slider, not every frame in that video will be accessible via the timeline interface (Figure 4.4, bottom). But the control granularity can not only be too low but also too high; using a slider that is several hundred pixels in size to select an item from a set of small cardinality loses much of the feeling of directness that direct manipulation interfaces are supposed to convey (Figure 4.4, top). It starts feeling like a 'wasteful' interface suffering from an expressive mismatch between the large control space of the UI and the small target space. This problem is not new, of course, and has been acknowledged for decades. Consequently, there exists a large body of related work that deals with the mismatch of the control and syntactic spaces.

The Alphaslider explores different ways of adjusting the control granularity of sliders.

The *Alphaslider* [Ahlberg and Shneiderman, 1994] is an early attempt to solve the closely related problem of selecting elements in long lists using a slider-based interface. In an experiment, Ahlberg and Shneiderman explore four different designs of giving the user control over the granularity of the syntactic mapping:

Their *position interface* allows users to select between one of three different granularities by dividing the thumb of the slider into three distinct click targets, each of which stands for one of the three granularity modes (Figure 4.5). The multi-scale timeline slider interfaces by Richter et al. [1999] are a continuation of this idea.

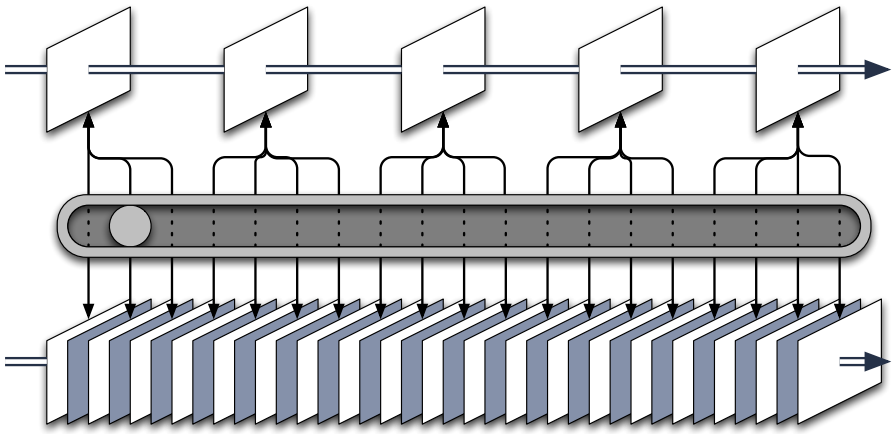

Figure 4.4: Different control granularities of a timeline slider for video navigation. *Top:* If there are less frames in a video than possible positions of the slider, multiple positions navigate to the same frame; the control granularity is too high. *Bottom:* If there are more frames in a video than possible positions on the slider, some frames (blue) do not correspond to any position; the control granularity is too low.

The *scrollbar interface* resembles the current implementation of scrollbars in many window systems; the thumb is dragged for coarse navigation—fine grained navigation is available by advancing through the list, one item at a time, by repeatedly clicking on arrow buttons on either side of the slider (Figure 4.6).

The *acceleration interface* couples the syntactic granularity of the slider to the velocity at which the user drags the slider thumb (Figure 4.7). This method is related to current adaptive *C:D ratio* approaches for controlling mouse cursors on large screens (cf. [Casiez and Roussel, 2011]). We also use a variant of this technique for an adaptive syntactic granularity approach of our own video navigation interface 4.2.3 "Interactive Scoping and Trajectory Filtering".

The *micrometer interface* extends the essentially one-dimensional control space of the slider along a second spatial axis: moving the pointer perpendicularly to the slider while dragging its thumb changes the syntactic granularity on a continuous scale (Figure 4.8). This technique has since been refined by Hürst et al. [2004a; 2005; 2008] and is part

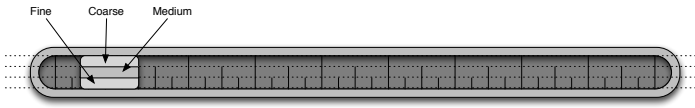

Figure 4.5: *Position interface* of the Alphaslider by Ahlberg et al. [1994]. Different click targets on the slider thumb explicitly select different control granularities.

Figure 4.6: *Scrollbar interface* of the Alphaslider by Ahlberg et al. [1994]. The slider is a regular slider; the control granularity depends on the cardinality of the set that is navigated via the slider. For fine grained navigation (one item at a time), the two buttons on the side can be used.

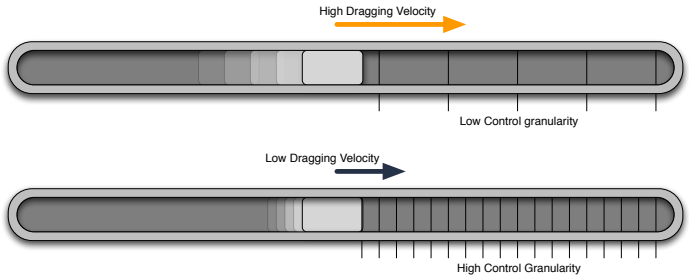

Figure 4.7: *Acceleration interface* of the Alphaslider by Ahlberg et al. [1994]. The control granularity is proportional to the dragging velocity of the slider thumb.

of a number of current standard video and audio timeline slider navigation interfaces, for example, on Apple's iOS[1] mobile operating system.

Popup Vernier [Ayatsuka et al., 1998] is an interface that uses a quasimode (cf. [Raskin, 2000]) to change the syntactic granularity of spatial direct manipulation—linear slider control as well as two-dimensional canvas panning—in a slightly different way: Not only is the 'gain' of the syntactic mapping changed for the input control but also for a lo-

Popup Vernier allows users to enter a mode where the effector space is temporarily zoomed in.

[1] http://www.apple.com/iOS

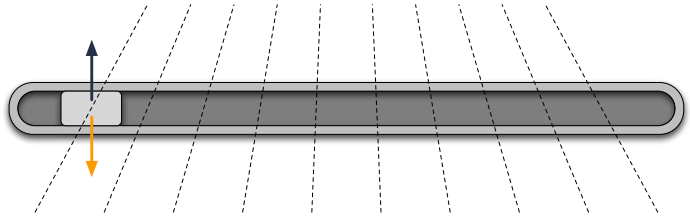

Figure 4.8: *Micrometer interface* of the Alphaslider by Ahlberg et al. [1994]. The control granularity is proportional to the y-offset of the cursor from the slider axis while dragging.

cal neighborhood of the visual feedback. The method can be compared to the vernier scale on common caliper tools, hence its name. The advantage of the *popup vernier* is that both input and output of the space to be manipulated are effectively zoomed to achieve the desired effect while still showing the context at its original scale. A related technique for video navigation based on pen pressure as the quasimode has been proposed by Ramos et al. [2003]. We have used a variant of this to allow users to control the combined syntactic and semantic granularity in one of our own video navigation interfaces (see 4.2.3 "Interactive Scoping and Trajectory Filtering" (p. 162)).

Other approaches fill the gaps in the control granularity of the slider by temporarily switching to rate-based controls.

Another approach to solve the control granularity mismatch problem of timeline sliders is to approach the positional direct manipulation concept of the slider in a more flexible way. Combined rate and position controls, for example, use the positional slider thumb as a visual indicator of the current location but explicitly de-couple the manipulation aspect by only allowing rate-based control. The rate is either directly set by the user, as in the simple classic interfaces for time-based media that commonly feature *fast forward* and *rewind* buttons. Or the rate is adjusted by an automatic closed-loop control to make the position in the target space asymptotically approach a target position in the control space. One early example of this technique is the *FineSlider* [Masui et al., 1995]; similar interfaces for time-based media such as video and audio have been proposed, among others, by Hürst et al. [2004b; 2006; 2008].

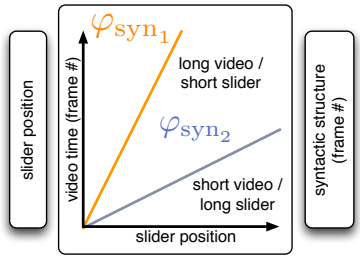

Figure 4.9: Possible mappings φ_{syn} between a timeline slider and the syntactic structure of a video. The exact mapping is difficult to predict for the user, because it depends on the ratio between the length of the slider and the duration of the video.

The control granularity of slider interfaces can change unpredictably depending on the size of the UI and the duration of the medium.

A second problem with timeline sliders for time-based media navigation arises when we look at how these interfaces change when the navigable space (the duration of the medium) changes: in most implementations, the slider does not change at all to reflect the change in cardinality of the navigation space! This behavior is inconsistent and unpredictable; if, for example, for one video, moving the slider to the right for 100 pixels means advancing by one minute in the content, the same control gesture can mean advancing by any other arbitrary amount of time, depending on the total length of this video. Moreover, most media navigation interfaces are resizable on the screen; changing the length of the slider again modifies the mapping φ_{syn} between the position of the playhead and the syntactic structure of the medium (Figure 4.9).

Even for syntactic tasks, the mapping can be unknown or change frequently.

In our model, this means that even if we assume a syntactic navigation goal along the lines of "I want to navigate to the timestamp of 00:02:40:23 in the video.", thus assuming a small semantic distance and an identity semantic mapping, the syntactic distance is still non-negligible (cf. Figure 4.3). The reason for this is that even such a syntactic navigation goal is not directly visible—unless the slider is labeled with the exact navigation target, which is unlikely. Thus, a user cannot know to which position on the slider their navigation goal corresponds. This position can only be determined if the combined mapping from the control space of the slider

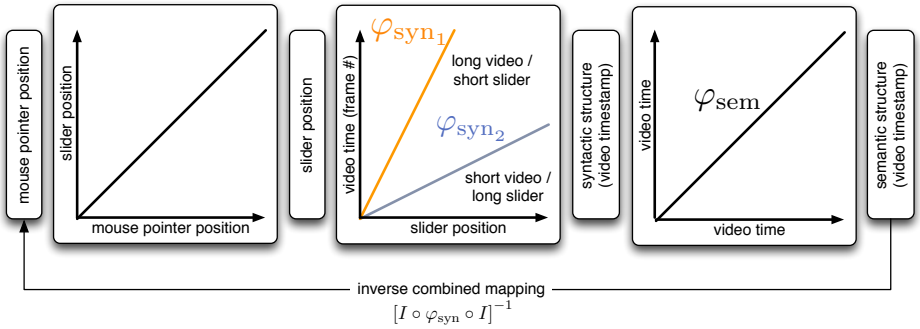

Figure 4.10: Combined mappings from the control space to the semantic structure. Even in the case of a syntactic navigation goal and thus an identity semantic mapping ($\varphi_{\text{sem}} = I$), the combined mapping is not easily reversible because of the variable control granularity of the slider. Users need to know the inverse mapping in order to project their semantic navigation target into the control space.

via the syntactic structure of the medium (φ_{syn}) to the semantic structure (φ_{sem}) is known and can be easily inverted (Figure 4.10).

The mapping between slider and medium has to be learned by the user.

In other words, even for a task where the semantic and syntactic structures coincide, the combined syntactic and semantic mapping of a timeline slider—the mapping that holds between the effector space of the user's control gestures and their impact on the target space of the domain of interest—is not generally discernible for the user but has to be guessed or learned. This hurts both the feeling directness of the interface [Hutchins et al., 1985] and the efficiency with which it can be operated.

This learning process is costly in terms of iterations through the seven stages of action.

Interpreting this process of navigating with a timeline slider using the model of Norman's *Seven Stages of Action* (cf. 2.1.2 "Activity and Interaction Models") reveals that, because of the unknown combined mapping, the 'correct' action sequence cannot be initially formed directly from the intention—instead an 'estimate' action sequence has to be devised and executed first. From the result of that initial action the users can then, hopefully, deduce or learn the combined mapping—usually through a number of iterations through the seven stages—before they have a chance to formulate an action sequence that finally allows them to navigate to their goal.

In his Diploma Thesis, Andreas Nett [2012] found that this learning process has a significant impact on task performance times even for very simple targeting tasks. He conducted a Fitts' law experiment where users were asked to perform a number of such 1D-targeting tasks using a physical slider. Even for an identity mapping between effector space and target space and with the tasks thus being *identical in motor space*, the movement times of participants were significantly slower when the mapping function was unknown in the beginning.

Experimental evidence suggests that learning the mapping is slow and that some mappings may even not be learnable.

For more complicated, non-linear mappings Nett observed that these mappings would not be learned at all. In such cases, the navigation strategy changed and was completely changed to a linear search using a fine-grained iterative process of trial and error where the result of every control gesture had to be evaluated and compared to the goal.

Generally, if the mapping is linear and known, users perform an initial ballistic control gesture, similar to what we see with Fitts's Law tasks, followed by a short homing phase of closed-loop control behavior [Woodworth, 1899]. With an unknown mapping, users seem to formulate a series of linear mapping hypotheses that are refined over multiple iterations. If the real mapping is not linear, these hypotheses are always wrong, causing the iteration deltas to become smaller, thus making the whole navigation process inefficient and tedious.

What we can see from this, is that the complexity and discoverability of the combined mapping directly influences the users' navigation strategy in terms of the seven stages of action. What could be a single, directed control gesture in cases where the mapping is known and simple enough to invert, becomes an iterative dead-reckoning task where no absolute but only relative comparisons between the current result and the goal are possible. Because of this problem, the timeline slider—and any interface that does not directly reveal its combined mapping—misses the full potential of direct manipulation.

If the combined mapping is unclear the benefits of direct manipulation are lost.

It should be noted that, so far, we have assumed a navigation task that is concerned with the syntactic structure of the medium only, causing the semantic mapping to be the identity. Even in this special situation where the semantic dis-

The situation may become even more difficult for semantic navigation targets.

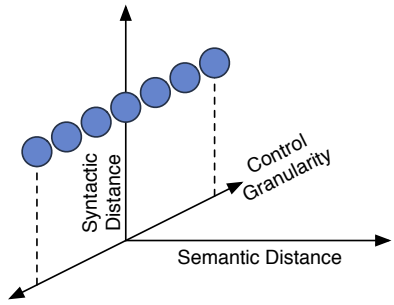

Figure 4.11: Timeline slider video navigation interface in the context of a purely syntactic task. Because of the variable syntactic mapping, the control granularity is unknown, and the syntactic distance in the interface remains large.

tance must be small, we have seen that the task is difficult when using a timeline slider because of its hidden syntactic mapping and the resulting large syntactic distance (Figure 4.11). If we now look at navigation tasks that include content-defined navigation goals, we will find that these problems get only more severe.

4.1.2 Analyzing the Semantic Distance of Slider-based Navigation in Videos

For semantic navigation tasks, the semantic mapping has to be learned as well.

Most navigation goals are not formulated in a syntactic way, as discussed above: when we navigate in a video, we are rather interested in finding specific content than numbers on a timeline. Timeline interfaces, of course, do not directly support semantic navigation; it is thus required that, in addition to the syntactic mapping of the slider, a user knows—or discovers on-the-fly—the semantic mapping from the video's semantic structure into its syntactic structure to successfully navigate the video (Figure 4.12).

Since the semantic mapping depends on the content, it cannot generally be known or predicted.

The problem is that due to its very nature, the semantic mapping is entirely dependent on the content of the video. It therefore can only be learned through rote memorization of the content and, generally, not be predicted at all if the content is unknown (Figure 4.13). For many videos that

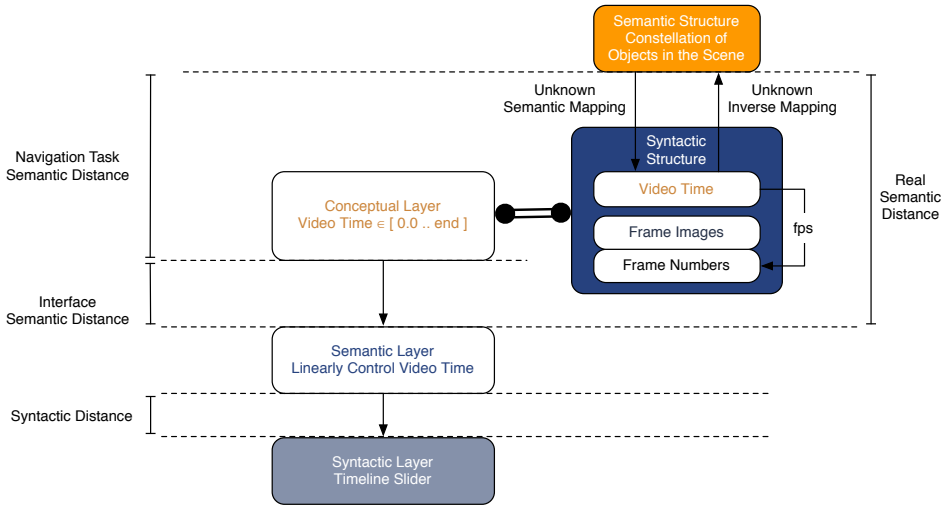

Figure 4.12: Combined navigation interface model of a standard video timeline interface in the context of a semantic navigation task concerned with the location of objects in the scene. The conceptual layer of the interface is associated with the syntactic structure. To plan their input for reaching a navigation goal, users must know the semantic mapping.

show 'real world' content, a limited prediction in a small local temporal neighborhood of the current position is often possible. If the current frame shows a person running from the left side towards the center of the scene, we can predict the semantic structure regarding the position of that person in the scene relatively reliably for a small number of subsequent frames. There is no way, however, to predict the complete semantic mapping that would allow us to know where that person would be located—or if it would be off the screen—for every frame in the video.

With the mapping being unpredictable, so is the outcome of any control gesture on the timeline slider.

One result of this problem is that the same control gesture performed on the timeline slider can have entirely different semantic meanings, depending on the content of and the position in a video. The objects or people visible in a scene at a certain point may exhibit different behaviors if the slider is moved, say, one centimeter to the right: they may move a short or long distance to the right, they may move to the left, or not at all. The mapping between the slider and the semantic structure thus may be the complete opposite of a natural mapping.

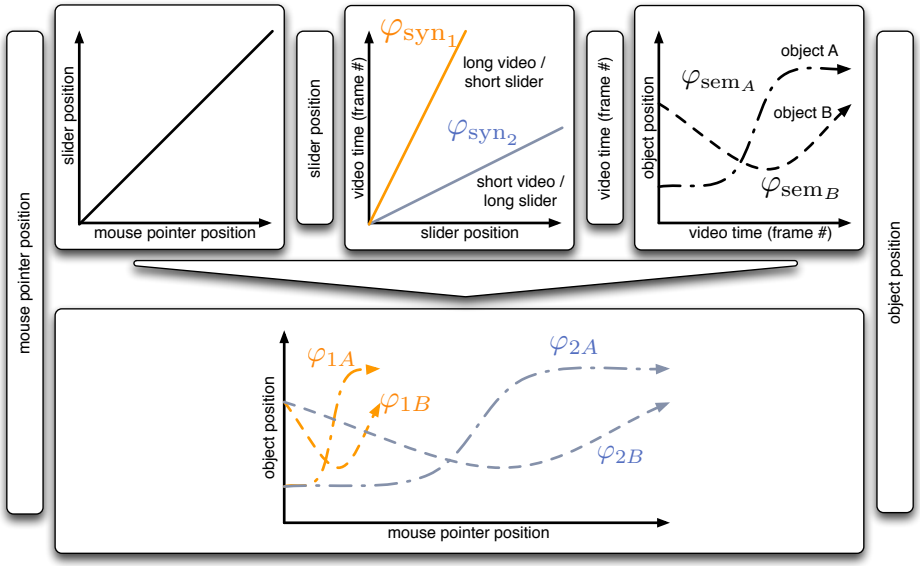

Figure 4.13: Combined mappings from the control space to the semantic structure for a navigation task concerned with object position. Possible mappings are influenced by the syntactic ambiguities of the timeline slider and by the content defined semantic structure. Users cannot know the combined mapping but through rote memorization.

In this situation, semantic navigation with the timeline slider is more of a search task than a selection task.

Reiterating our argument about the *seven stages of action* from above, we can therefore state that under a combined mapping that is a concatenation of the (usually linear) syntactic mapping of the timeline slider and the highly nonlinear semantic mapping induced by the content of the video, the whole navigation process easily degenerates to an exhaustive search over the length of the video. Every control input on the timeline slider maps to a change of position in the semantic structure with arbitrary granularity (Figure 4.14) and possibly even in an unpredictable direction. Thus, no initial ballistic movement can be made to rapidly close the distance to the navigation goal without the need of closed-loop control, but the whole navigation must be performed in fine-grained control movements, always cycling through all seven stages.

Using a timeline slider is not direct manipulation.

This unpredictability of the effects of any control input is also the reason why video navigation using a timeline slider with semantic navigation goals violates the basic principles

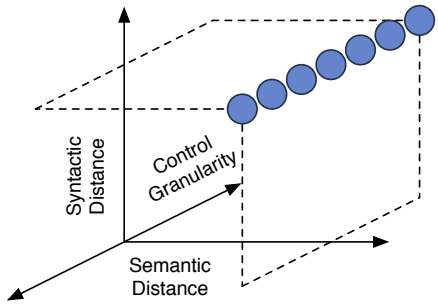

Figure 4.14: Design space position of a timeline slider video navigation interface in the context of a semantic navigation task. Both the semantic and syntactic distance are large: The syntactic mapping is unknown and variable due to the mechanics of the slider. The semantic mapping is unknown and variable due to the nature of the content.

of direct manipulation. In direct manipulation interfaces, "users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity." [Shneiderman, 1982]. Due to the non-linearity and possible non-monotonicity of the combined syntactic and semantic mapping, this claim does not hold true for timeline slider-based video navigation. This fact is also visible if we place such interfaces into our design space for media navigation.

4.2 DRAGON, an Interface for Semantic Video Navigation

To create a true direct manipulation interface for semantic video navigation, we again have to find a way to make the semantic structure directly accessible to the user. Moving through the four steps proposed in the theory chapter 2.2.4 "Generating New Interfaces Using the Combined Model", we will illustrate our design and development process for DRAGON, our direct manipulation video navigation (DMVN) interface. In this chapter, we will lay special em-

phasis on the third step of designing a suitable UI for semantic navigation.

Parts of the material found in this section have already been published in conference and thesis papers: The original interface idea, its design, and the results of an evaluation with users was published by the author et al. at CHI 2008 [2008]. Technical improvements and extensions to the algorithm have been proposed by Mortiz Wittenhagen in his Diploma Thesis [2008]. Work on interface-level solutions to DMVN interaction problems has been done by Christian Brockly in his Diploma Thesis [2009]. A number of experiments on allowing users to control the semantic granularity of the DRAGON system have been carried out by Alisa Novosad in her Master's Thesis [2012]. Anne Kathrein has developed a simple semantic query language in her Bachelor's Thesis [2011] for providing a search interface based on the semantic structure that DRAGON navigates. An analysis of the necessity to embed aspects of the syntactic structure into the semantic navigation in certain situations, and thereby creating more task-dynamic semantic structures, has been published by the author and Moritz Wittenhagen at CHI 2012 [2012]. For his Bachelor's Thesis, Dennis Lewandowski has performed experiments to enhance standard tracking algorithms with 3d-information about the scene gathered with a Microsoft Kinect™[2] depth camera [2011]. All thesis work referenced above has been conducted under the guidance of the author. These publications will also be again cited explicitly in the text below, where it is applicable.

4.2.1 Finding a Conceptual Model

Semantic navigation goals in video scenes often concern the locations or constellations of objects or parts of the scene.

Many navigation tasks in videos require the user to find and move to a certain moment or, more precisely, frame in a video where something of relevance *happens*. Such tasks cannot, as already discussed above, be easily formulated purely in the syntactic terms of playback time codes or frame numbers—especially not *a priori*, before seeing and learning the mapping between content and timeline. Often, however, they can simply be expressed in terms of the ob-

[2] http://www.microsoft.com/en-us/kinectforwindows

jects visible in the scene and their absolute or relative spatial locations or constellations [Karrer et al., 2008].

Many examples for such semantic navigation goals exist.

Examples include video analysis in traffic surveillance, sports, or security, but also video debriefing processes in the training of various skills such as presenting, artistic performance, and other physical activities. Typical tasks in these settings are: Checking if a car entered an intersection while the light was still red, checking for fouls such as handballs or offsides in a soccer match, finding the moment where an object is removed from the scene, or analyzing ones posture and position on stage when rehearsing an important talk. Other goals and requirements that can be similarly reduced to the spatial relations of videographed objects also exist in other fields. Behavioral researchers and video ethnographers create statistics over reactions of humans and animals to selected stimuli, and often these reactions are observable by spatial movement or gestures; and finding movement patterns of people in reaction to products by means of video analysis is also an important tool for market researchers [Kathrein, 2011]. Likewise, medical researchers use videos to investigate sleeping and other disorders [Babic, 2010] and possible treatments.

What is common to all of these areas, is that the relevant questions are answered by navigating to the locations in a video where the objects of interest in the scene occupy a certain position or are located in a special way relative to each other. This is not necessarily limited to a single object of interest but often is concerned with the interplay of changing sets of objects: In the soccer match example, the constellation of two players on the field is important for offside decisions, determining if a handball foul happened, requires to observe the relative position of a player's hand and the ball, and for analyzing situations like that of the famous 'Wembley goal', we need to know the location of the ball in relation to the goal line, a static part of the playing field. Therefore, the conceptual model for the aforementioned tasks must include the absolute location in the scene of any object; from this we can deduce the relative locations of any two objects at any point in time in the video.

Following this argument, a suitable semantic structure that covers the conceptual model of how we want to navigate in the video can be defined by the union of the locations at

As our semantic structure, we propose the space defined by all in-scene locations of a set of designated objects.

each point in time for all objects in the video. This structure is easy to imagine as the set of the motion paths of all objects (cf. Figure 1.3). The motion path of an object, of course, is in itself a function between the location of the object in the scene and the time in the video at which it occupies this location. Being able to navigate this semantic structure would mean that we could take one object and find the state and the relative position of all other objects in the video for every position of our object. In the traffic light scenario described above, for example, we could check the status of the traffic light in the moment the car enters the intersection by specifying the car's location as on the intersection. This semantic structure thus changes navigation in the medium from *when something happened*, which we cannot know a priori, to *where or in which context something happened*, which is usually already defined by our task.

4.2.2 Determine the Semantic Mapping

The semantic mapping for our semantic structure is really a set of mappings and has to be derived via its inverse.

To realize navigation through the semantic structure described above, we need to establish the semantic mapping from the spatial representation of the motion trajectory of every object in the video to the syntactic timeline. It is immediately clear that this mapping not only is actually a set of mappings—one mapping for each object—but also that it is infeasible to be created manually, e.g., by annotating the full trajectory for each object. We thus have to follow the approach of establishing an initial mapping $\varphi_{\text{sem}}^{-1}$ from the syntactic into the semantic structure in an automated way by means of algorithmic approximate extraction of the semantic structure, as described in 2c "Generating New Interfaces Using the Combined Model". This initial mapping then specifies the location of each object in the frame for every point in time and is best defined separately for each object. Because they can be visualized by the three-dimensional motion path of each object, we call these object-specific mappings the objects' *trajectories*.

Each object in a scene defines its own time-invariant semantic mapping.

For each object, we can invert the mapping by performing a reverse look-up in the trajectory and implicitly warp the syntactic structure of the timeline to normalize the temporal progression through the video to the object (Figure 4.15). This allows us to specify a number of objects and their ab-

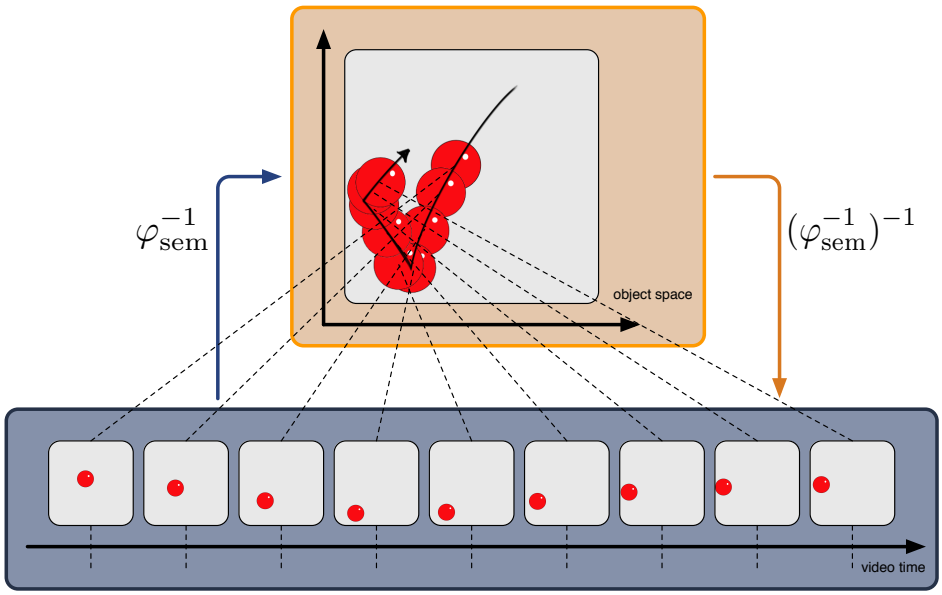

Figure 4.15: The inverse semantic mapping $\varphi_{\text{sem}}^{-1}$ in DRAGON describes for each object its spatial position in the scene over time. The semantic mapping itself can be implemented as an inverse look-up of this function, allowing us to know the time when an object occupies a certain position.

solute or relative positions to directly find the point in time where this constellation is given and then directly navigate there.

Unfortunately, the initial mapping $\varphi_{\text{sem}}^{-1}$ is generally not injective—an object may occupy the same spatial location in the video at two or more points in time—, thereby making its inversion over the entire image of the map mathematically impossible (Figure 4.16).

The initial mapping is invertible over a local neighborhood at any point as long as the object is moving.

This problem can be circumvented, however, for most real-world video material, because the trajectory of an object usually can be divided into continuous segments where it does not self-intersect. Thus, we can almost always define the inverse of the initial mapping over a local neighborhood of the current *temporal* position in the video if the object is actually moving. In cases where an object of interest pauses, $\varphi_{\text{sem}}^{-1}$ becomes singular even in a local neigh-

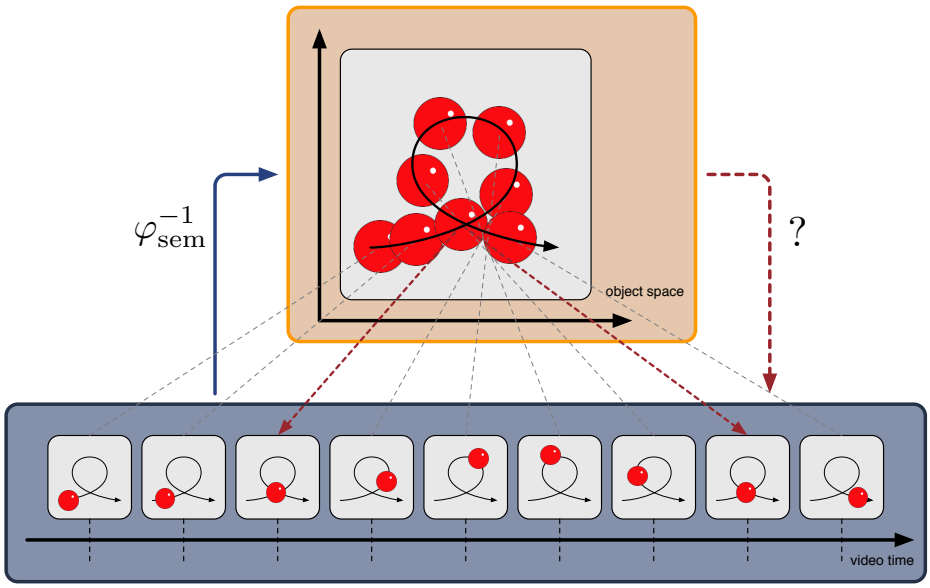

Figure 4.16: Singularity problem of the inverse semantic mapping $\varphi_{\text{sem}}^{-1}$. Often, objects occupy the same location in the scene at more than one point in time; in these cases, $\varphi_{\text{sem}}^{-1}$ cannot be inverted over its entire image. As long as the object does not pause, however, we can construct a semantic mapping $\hat{\varphi}_{\text{sem}}$ over a local temporal neighborhood.

borhood[3], and other measures, which will be explained in-depth below 4.2.3 "Object Pauses", must be taken to enable semantic navigation.

Acquiring the initial mapping is difficult because of tracking inaccuracies and the ambiguity of scope.

What is still missing at this point is, of course, how we can establish the initial mapping in the first place. Acquiring the trajectory of an arbitrary object in a video, however, is relatively difficult and primarily so for two main reasons:

- Object scoping
  Defining the scope of objects of interest is difficult and usually imprecise: If someone points at a person, we do not immediately know if the person, the upper torso, the shirt the person is wearing, or just the abstract concept of 'human being' is meant to be desig-

[3]Mathematically, this can be explained by the *implicit function theorem*, which allows to express relations as functions on local neighborhoods. If an object pauses, the partial derivatives of the trajectory at this point do not satisfy the conditions under which the theorem holds.

nated. Likewise, objects can consist of multiple parts that all move and behave differently, like the limbs of a person jogging, that may split off and join again later, like a cap coming off a bottle and being attached again, and that may change in appearance over time. Without an ontology, a closed systematic vocabulary of all describable entities and the relations between them, this scoping problem is very difficult to solve. Unfortunately, such ontologies are infeasible to construct for every video or even just for every genre of video.

- Object tracking
  The most basic requirement when we want to create trajectories for all objects in a video, of course, is that we can reliably track each object and find out where it is located spatially at any given time. Also, in order to create a consistent trajectory, we need to identify an object even if it temporarily leaves the frame, is occluded behind other objects, or changes appearance. Otherwise, a trajectory may be split up into two unrelated ones in these situations, reducing our ability to express certain semantic navigation goals. Objects may additionally exhibit different motion patterns on different scales, which makes tracking more difficult: A rolling soccer ball, for example, should have a different trajectory as a whole than any of the black spots on the ball each individually have. Lastly, the trajectory of an object that is tracked 'correctly' can exhibit spatial variations of high frequency that—depending on the task—may not be important for the semantic structure. These last two points are, of course, also closely related to the scoping problem above.

Solving these problems means solving computer vision in general.

Both of these problems basically require that the physical structure of the scene can be determined from the frame's visible two-dimensional projection of the camera frustum. This is one of the goals of *computer vision*, which is a large research field of its own; solving this problem is naturally out of the scope of this thesis. For our purposes, it suffices to work around these problems using a number of assumptions and approximative approaches. These will still enable us to navigate through the semantic structure of a video with only small limitations. In the following, we will describe several of these techniques for creating sets of object

trajectories in a given video scene and discuss their respective advantages and disadvantages.

We look at four approximative approaches to establish the initial mapping.

One main distinguishing factor between these methods is to what extend and how they try to solve the computer vision problem of recovering the structure of the depicted scene. We therefore divide the set into four classes:

1. Generating object trajectories in a structure-agnostic way.
2. Generating object trajectories by (partly and approximately) recovering the structure.
3. Generating object trajectories by measuring the structure at capture time.
4. Generating object trajectories by letting the user give cues about the structure.

Generating Object Trajectories in a Structure-agnostic Way

As a first attempt, we can ignore the structure of the scene and rely on pixel tracking alone.

One of the simplest approaches of generating object trajectories for a video scene is, somewhat surprisingly, to completely ignore the spatial structure of all objects and the scene itself. If we assume that objects are not occluded and if we neglect the wish to recognize reappearing objects and create trajectories across scene cuts, an object's trajectory will usually be approximated very well by the trajectory of any pixel on the object. The high spatial and temporal coherence of pixel clusters forming objects in a typical video is the reason why single pixel tracking instead of object tracking produces adequate results. This reduced problem can be solved more easily, for example, by using *optical flow* algorithms.

Optical flow is an approach to estimate the motion field at every pixel in a scene.

Optical flow, a concept originally described by Gibson [1950], is a measure to describe the apparent motion field of the content relative to the viewpoint between two images of a scene (Figure 4.17). Although the *apparent* motion field can deviate quite drastically from the real motion in a scene for certain cases (Figure 4.18), it generally gives good motion estimates between two temporally adjacent frames

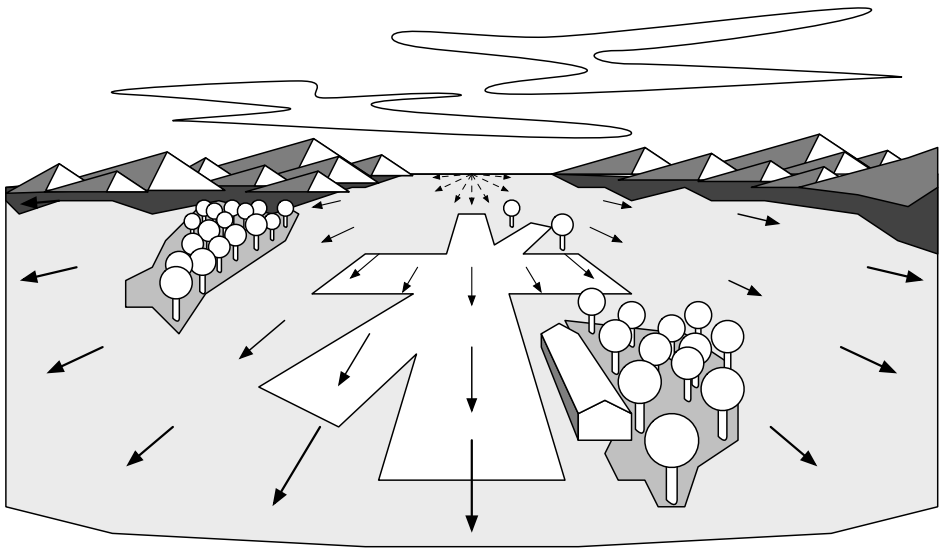

Figure 4.17: Optical flow of a scene with forward motion. Figure adapted from [Gibson, 1950].

for many, if not most, real-world scenes. Because of the relative simplicity of the problem and the wide availability of algorithms to solve it, the uses of optical-flow-based motion estimation are manifold and include areas like robotic control [McCarthy and Barnes, 2004], video segmentation [Galic and Loncaric, 2000], or input devices [Ballagas, 2007].

Mathematically, we treat an *optical flow field* as a function V from the spatial support domain of a video frame image—the discrete two-dimensional pixel space—into the continuous space of two-dimensional spatial displacements:

$$V : \mathbb{Z}^2 \to \mathbb{R}^2$$
$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix}$$

To obtain this function for a pair of video frames $I(x, y, t)$ and $I(x, y, t + 1)$, we make three assumptions:

- *Constancy of brightness:* the projection of a point in the first frame looks identical to its projection in the second frame.

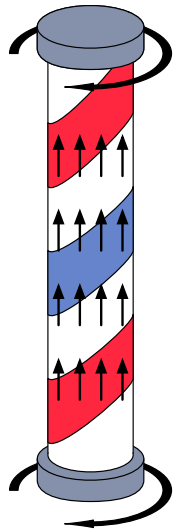

Figure 4.18: Optical flow when the apparent motion deviates from the true motion in a scene. The apparent motion can be ambiguous: the pole rotates but the stripes may appear as moving in the vertical direction. Similarly, the optical flow cannot be uniquely defined.

- *Small motion:* points do not move very far from one frame to the next. This assumption can be relaxed by iteratively computing and refining the flow through a scale space representation of the frames.
- *Spatial coherence:* the flow field is smooth in a local neighborhood.

The first assumption can be formalized as

$$I(x, y, t) = I(x + u(x, y), y + v(x, y), t + 1).$$

Approximating the right hand side of the equation by its first order Taylor expansion yields

$$I(x, y, t) \approx I(x, y, t) + \frac{\delta}{\delta x} I \cdot u(x, y) + \frac{\delta}{\delta y} I \cdot v(x, y) + \frac{\delta}{\delta t} I,$$

which can be written as

$$(\nabla I)^T \cdot V + I_t = 0.$$

The flow field equation is not uniquely determined, thus requiring additional constraints.

We see that we can get one equation in two unknowns for each pixel. This is an under-determined system that has no unique solution, a problem that is known as the *aperture* problem in the context of optical flow. The constraints necessary to solve for V can be gained from the two remaining assumptions; a number of flow algorithms have been developed that do so in different ways: Lucas and Kanade [1981] proposed a version that calculates a normalized weighted sum of flows over a spatial window, thereby creating an over-determined system of brightness constancy equations and then finding a solution through least squares optimization. Other algorithms employ global smoothness constraints on the flow field [Horn and Schunck, 1980] or additionally try to minimize the error of the approximation by delaying the linearization to a later point in the algorithm [Brox et al., 2004].

Object trajectories are created by following the flow volumes in both temporal directions.

For each pair of subsequent frames in a video, we can calculate two different flow fields V: one in forward direction $\overrightarrow{V}$ and one in backward direction $\overleftarrow{V}$ (Figure 4.19). To generate a pixel trajectory from a set of dense flow fields for a video scene, we can use a simple *dead reckoning* approach: from a single seed pixel $p_{t_0} = (x_{t_0}, y_{t_0})^T$ in the current frame at time t_0 we follow the respective flow fields in both directions, accumulating the displacement vectors from the field onto the pixel's coordinates:

$$p_{t_0-1} = p_{t_0} + \overleftarrow{V}_{t_0}(p_{t_0})$$
$$p_{t_0+1} = p_{t_0} + \overrightarrow{V}_{t_0}(p_{t_0}).$$

This step is then recursively applied to the two resulting locations using the next flow fields in forward and backward directions

$$p_{t_0-n} = p_{t_0-n+1} + \overleftarrow{V}_{t_0-n+1}(p_{t_0-n+1})$$
$$p_{t_0+m} = p_{t_0+m-1} + \overrightarrow{V}_{t_0+m-1}(p_{t_0+m-1}).$$

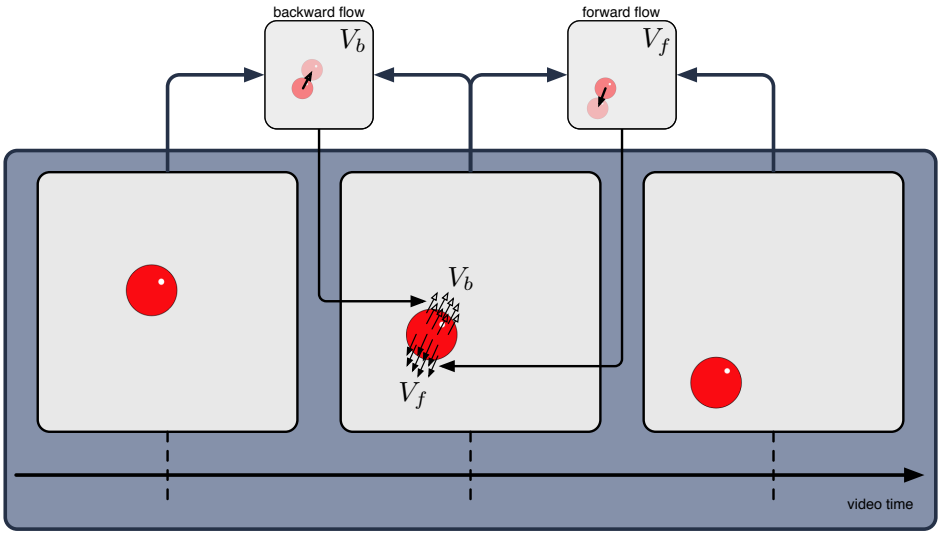

Figure 4.19: Optical flow between adjacent frames of a video. Each pixel in a frame is assigned a displacement vector that points to the location of the pixel in the neighboring frame. This is done in both directions.

Note that the result of any evaluation of the flow functions V are two-dimensional vectors in $\mathbb{R}^2$, but the input domain of these functions is $\mathbb{Z}^2$; we thus have to construct wrapper functions $\hat{V}$ by bi-linearly interpolating between the four nearest neighbors in $\mathbb{Z}^2$:

$$\hat{V} : \mathbb{R}^2 \to \mathbb{R}^2$$
$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \frac{\lceil y \rceil - y}{\lceil y \rceil - \lfloor y \rfloor} H_1(x,y) + \frac{y - \lfloor y \rfloor}{\lceil y \rceil - \lfloor y \rfloor} H_2(x,y),$$

where

$$H_1(x,y) = \frac{\lceil x \rceil - x}{\lceil x \rceil - \lfloor x \rfloor} V(\lfloor x \rfloor, \lfloor y \rfloor) + \frac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor} V(\lceil x \rceil, \lfloor y \rfloor)$$

$$H_2(x,y) = \frac{\lceil x \rceil - x}{\lceil x \rceil - \lfloor x \rfloor} V(\lfloor x \rfloor, \lceil y \rceil) + \frac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor} V(\lceil x \rceil, \lceil y \rceil)$$

In this way, starting from the initial seed pixel position p_{t_0}, we get a list of sub-pixel precise coordinates for the point

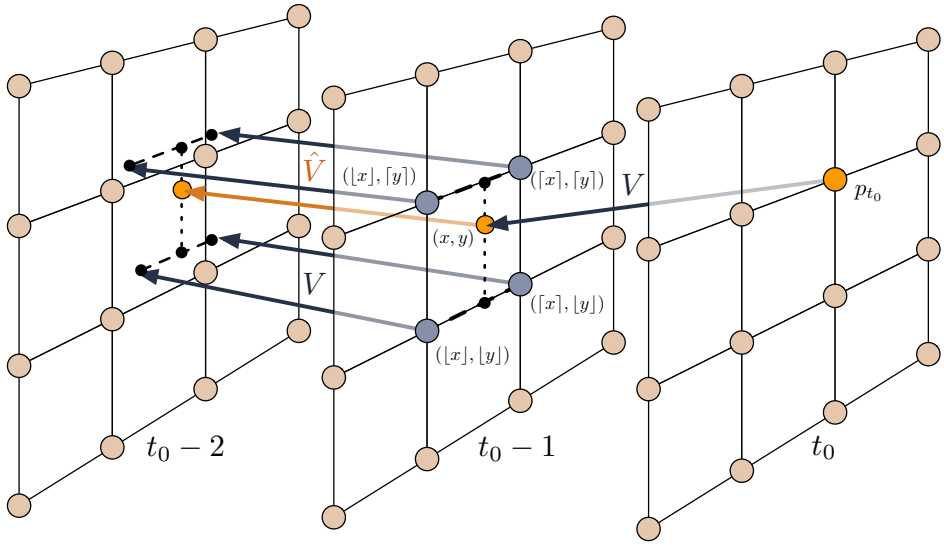

Figure 4.20: Bi-linear interpolation $\hat{V}$ of the optical flow function V allows the flow field to be evaluated at non-integer coordinates (x, y). This is necessary to recursively construct the trajectory of the seed pixel p_{t_0} through the video volume.

in every frame (Figure 4.20). This list then represents the trajectory of the seed pixel through the video volume.

Optical flow can create long, stable trajectories with high precision.

Modern optical flow algorithms produce very precise flow fields that allow to follow a pixel—and thus creating the illusion of tracking an object—over a large number of frames, which is important for creating long, connected trajectories of objects through a video scene. Also, the high density and sub-pixel precision of these flow fields gives us the ability to track very small objects that only consist of several pixels (cf. Figure 4.21). For the first prototype of our semantic video navigation system, DRAGON [Karrer et al., 2008], we therefore employed an adaptation [Weiß, 2007] of Brox's flow algorithm [Brox et al., 2004].

Optical flow also has a number of weaknesses.

While this or very similar structure-agnostic approaches of generating object trajectories for the semantic mapping have been used in our early versions of DRAGON and other existing DMVNs [Dragicevic et al., 2008], there are still a number of disadvantages, some of which we have already mentioned above:

Figure 4.21: Optical-flow-based tracking of small objects in videos. The pigeon covers less than 30 pixels in total (left); still, the optical flow field describes its motion very precisely (right). Faster but less dense tracking methods have problems in such cases—the pigeon, for example, does not contain a single *SIFT* feature.

- Object Occlusion
  If the object of interest is occluded temporarily, the pixels belonging to the object cannot be tracked through the occlusion with optical flow. Partially occluded objects can be tracked as long as the trajectory of the seed pixel stays clear of the occluded area. Otherwise, the tracking breaks down; or, even worse, as the optical flow field is determined by the foreground object, the resulting trajectory will switch over to the occluding object.

- Aggregate Objects
  Objects that consist of multiple parts, which may exhibit differing movement patterns, are handled relatively gracefully by optical flow tracking. Because only pixels are tracked, the trajectory implicitly follows a 'leaf' node of the aggregate object, thus representing the combined movement of both scales (Figures 4.22 and 4.23). There is no way, however, to explicitly state that the trajectory should only show the movement of the whole object.

- Apparent vs Real Motion
  As explained earlier, optical flow only represents the apparent motion field in a two-dimensional projection of a potentially three-dimensional scene—the real motion of the objects cannot be recovered with this approach. This limitation is, however, unproblematic

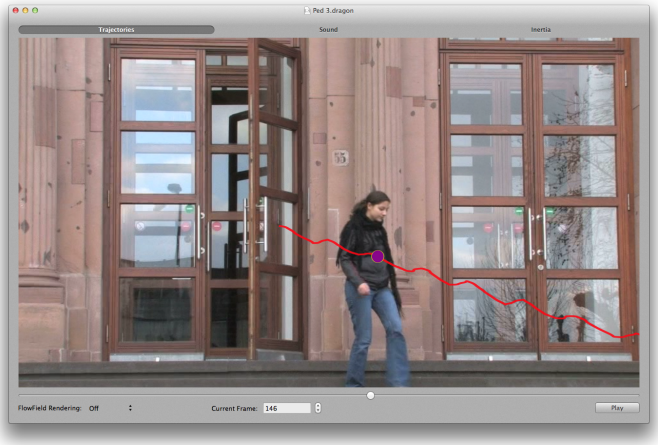

Figure 4.22: Trajectory of the person's body as tracked with optical flow. The steps of the stairway can be clearly seen in the person's motion.

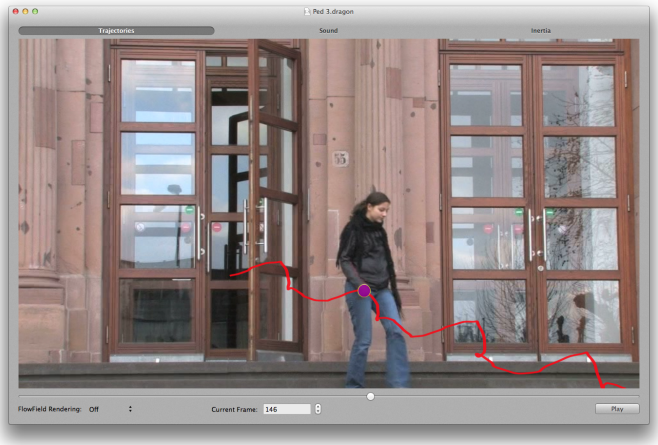

Figure 4.23: Trajectory of the person's hand as tracked with optical flow. In addition to the motion of the body (Figure 4.22), the trajectory shows the swing of the arm.

for semantic video navigation, because we expect the navigation goal to be specified in the projected space as well.

- Accuracy vs Speed Tradeoff
  A disadvantage of this kind of *dense* optical flow estimation and tracking is that the process is computationally expensive. For an actual implementation of a semantic video navigation interface, the optical flow fields for all frame pairs in a video have thus to be calculated in an off-line pre-processing step for the semantic mapping to be available at navigation time. It is possible to speed up the process up to interactive rates by only calculating a sparse optical flow field—for example, by finding and tracking *scale invariant feature transform (SIFT)* points [Lowe, 2004]—and interpolate between the flow vectors. Some existing DMVNs, such as *DimP* [Dragicevic et al., 2008], follow this approach at the cost of losing tracking accuracy and thus the ability to create trajectories for very small objects (cf. Figure 4.21). On the other hand, there are also tracking algorithms that are more precise than dense optical flow, but which take even longer preprocessing times. One example is the particle tracking [Sand and Teller, 2006] method employed in the DMVN by Goldman [2008].

Generating Object Trajectories by (Partly and Approximately) Recovering the Structure

Some of the problems listed above can be alleviated if the geometric structure of the scene is—at least partially—estimated from the pixel content of the frame sequence. In the context of video navigation, two methods have been proposed by Kimber et al. [2007] and Goldman [2008], and one has been developed at our own group by Moritz Wittenhagen [2008] in his Diploma Thesis.

Trailblazing tracks at the object level and can handle simple unary and binary operations.

In their *Trailblazing* video surveillance tool, Kimber et al. [2007] create a structure of the scene by detecting people, tracking their bounding rectangle, and classifying the interactions between them into five distinct classes. The video frames are first segmented per-pixel into background

and foreground by employing a Gaussian mixture model approach [Stauffer and Grimson, 2000] and then comparing neighboring frames using a fast normalized cross correlation [Lewis, 1995]. The interactions between the tracked objects are then classified and labeled as *appearing*, *disappearing*, *merging*, *splitting*, and *continuing*. Objects are identified for association across occlusions by a Bayesian segmenter based on appearance features. The system is able to create an integrated semantic structure over a number of synchronized video feeds (e.g., multiple security cameras in a building) and, unlike other DMVNs, even allows the underlying spatial domains of the semantic structures of each video feed to be combined into a joint world-coordinate space. For this feature, all cameras must be fixed, and the extrinsic calibration of each camera must be known in the world-coordinate space beforehand. The integration is handled by the *dynamic object tracking system* (DOTS) developed at FX-PAL [Girgensohn et al., 2006]. The results are finally entered into a database for lookup during the actual navigation interaction.

Trailblazing is built to handle partial and full occlusions of objects by maintaining an identity of the tracked objects, which is an advantage over the structure-agnostic approaches discussed above. The problem of object scoping and aggregate objects is mitigated by the fact that trailblazing is designed to track objects only at the coarsest scope and represents them through their bounding rectangle. Also, certain scoping problems are directly addressed by explicitly including object behaviors like merging and splitting into the tracked object model. In terms of accuracy and speed of their method, Kimber et al. give no information in their paper, but the system architecture, which depends on a relational database to hold the information about the objects and their interaction classes, suggests that *Trailblazing*, too, requires an off-line pre-processing step.

Goldman's DMVN employs particle tracking, a precise but very expensive algorithm.

Another example for a system that estimates part of the scene's structure is the video navigation and processing tool by Goldman et al. [2008]. Instead of tracking each pixel, their system is using a computationally expensive algorithm that tracks particles to create a semi-dense field of temporally very stable and consistent trajectories [Sand and Teller, 2006]. Up to that point, this approach is comparable to the optical flow method described above, only with the

difference of sacrificing computational efficiency for added accuracy of the trajectories.

Users need to give hints about the structure of the scene by specifying the number of particle clusters.

In a second step, the particle trajectories are clustered in affine motion space using the k-means clustering algorithm; the parameter k, the number of clusters, is tuned by hand. The resulting clusters each contain particles that follow a similar affine motion model, and thus each approximately represents either a part of the background or a part of a moving object in the scene. This set of particle clusters is, of course, only a very rough representation of the structure of the scene, but the technique helps to overcome some problems of object scoping, partial occlusion, and tracking under lighting changes.

DragonEye models object motion and camera motion separately.

DragonEye [Wittenhagen, 2008] is our own attempt at creating a trajectory algorithm that reconstructs parts of the scene structure by using two models: one for the camera motion and one for the shape and motion of objects. The main motivation for developing a new tracking system was to allow better occlusion handling than our optical flow approach, while at the same time increasing the run-time performance to a point where the semantic mapping can be constructed without the need for a pre-processing step. In most cases, the motion of the camera is not part of the semantic structure—first person perspective videos are an exception—, so we need a camera model that is separate from the object motion model to treat these two possible motion components in a scene differently. This factorization of the absolute motion of objects in the scene into camera motion and relative object motion is also beneficial from an interaction standpoint for the final navigation user interface, as we will see later in 4.2.3 "Designing the User Interface for Semantic Video Navigation".

The models are built from SIFT tracking data.

The DragonEye algorithm works as follows: In a first step, SIFT features [Lowe, 2004] are calculated for every frame, and an affine camera motion model is built by estimating background plane homographies from the largest affine motion cluster of matched SIFT points in each adjacent pair of frames. The SIFT algorithm is executed on the GPU using an implementation by Wu [2006] for added performance. For the homography estimation, we use the OpenCV [2000] implementation of the direct linear transform (DLT) algorithm and suppress the influence of out-

liers using our own implementation of the random sample consensus (RANSAC) algorithm [Fischler and Bolles, 1981]. The camera motion model is then used to warp all frames for each scene of the video to a single common image plane.

DRAGONEYE learns a set of feature points.

From this step on, for each object of interest in the scene, a 'bag-of-points' model is learned, and a trajectory is generated on-the-fly (cf. Figure 4.24). Similar to the optical flow approach detailed above, a single point on an object of interest serves as a *seed point* from which the trajectory is 'grown' in both directions (Figure 4.24 (a)). In a small radius around the seed point, SIFT points that are believed not to belong to the background—according to the already generated background motion model—are added to the object's bag-of-points (Figure 4.24 (b)). These points are stored together with a weighting score, which increases over time if the point stays compliant with the model, and the spatial offset vector between the SIFT point and the seed point. This model and the estimated object position are then propagated and adjusted in both directions through the video.

The feature points in the model determine the location of the object through a voting scheme.

For each next frame, the algorithm tries to match all SIFT points in the model to points in the new frame. The matching pairs of points with a displacement matching the background motion are, again, ejected from the model. Each remaining match then votes for the new position of the object indicated by its stored offset vector (Figure 4.24 (c)). This approach of finding an object in an image through offset-vector voting of feature points is conceptually very similar to a generalized Hough transform [Ballard, 1981]. The points are clustered spatially by their voted-for positions, and the clusters are weighted by the number, score, and length of the offset vector of all points contained in each voting cluster. By this weighting, we ensure that the voted-for position of a cluster is considered a good estimate only if it is supported by:

1. many SIFT features,
2. by features that have contributed to earlier iterations of the model and thus have a higher chance to represent a distinguishing optical feature on the object, and
3. by features that are close to the tracked center of the object.

The object's position in the new frame is finally determined by interpolating the predictions of each point in the best cluster.

The model is evaluated and updated in every frame.

Afterwards, all new SIFT points that have been detected in the frame and that do not have a match in the preceding frame are analyzed. If they lie inside an ellipse determined by the point cloud of the winning cluster, they are added to the bag-of-points model (Figure 4.24 (d)). Existing points in the bag have their scores increased if they were part of the winning cluster. Those that were not part of the cluster have their scores decreased; the same happens to those points that could not be matched to any SIFT feature in the current frame. Then, the next frame is loaded and processed in the same way. The object's trajectory is continuously formed by appending the position estimates for each frame.

If the model fails, a color-histogram-based CAMShift tracker takes over temporarily.

If at any point in time not enough points can be matched to ensure a clear vote for the position of the object, the tracking is suspended for this frame. Because this mostly happens in cases of extreme motion blur where it is difficult to find strong SIFT points, a color-space-based *CAMShift* tracker [Comaniciu and Meer, 2002; Comaniciu et al., 2003] is run in parallel and takes over if that situation arises. In every following frame, an attempt is then made to re-start the DRAGONEYE tracker in that direction using the position currently predicted by the CAMShift tracker as the new seed. A more detailed description of the algorithm and its mathematical background can be found in [Wittenhagen, 2008].

DRAGONEYE trades some precision for greatly improved speed and robustness.

The main advantages of using the DRAGONEYE algorithm to create the trajectories of objects in the video scenes are its ability to process videos taken with a non-fixed camera setup, its resistance to partial and full occlusions even over multiple frames (Figure 4.25), and its relative computational efficiency when compared to dense optical flow or particle tracking. Since aggregate objects are not explicitly modeled, it is difficult to track specific parts of such objects; as soon as the object exhibits a similar motion in multiple of its constituent parts, features of other parts than the initially specified are learned and integrated into the model, which usually results in the object being quickly tracked as a whole. Following small motion details with high spatial frequencies is problematic for the same reason. Although DRAGONEYE employs two tracking approaches in paral-

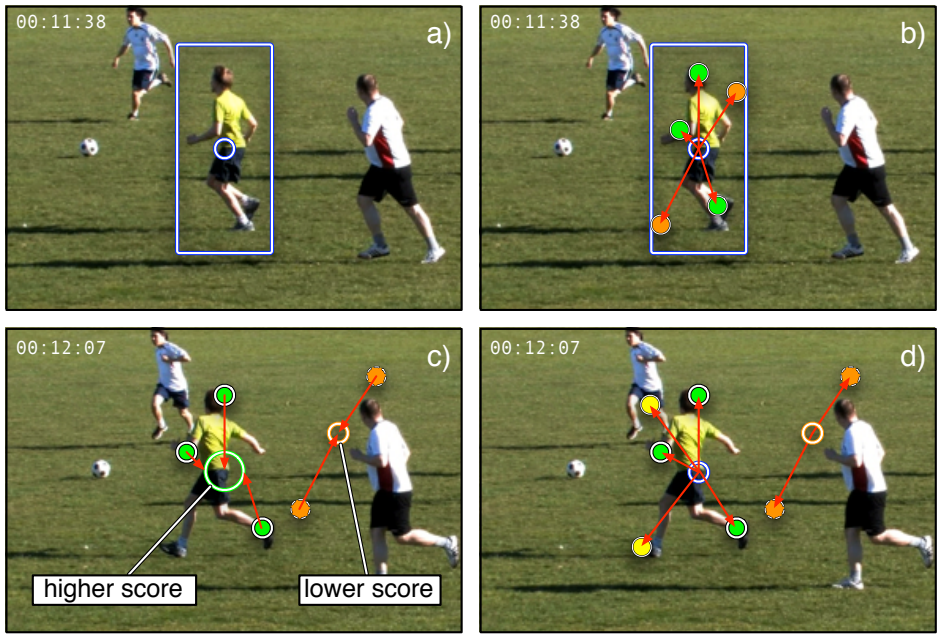

Figure 4.24: The DRAGONEYE algorithm: Starting from a single seed point (a), SIFT points in the neighborhood are collected into a bag-of-points model (b). In the next frame, corresponding SIFT points are found; the offset vectors to the original object center are applied, each voting for a new object position (c). The highest scoring vote cluster is accepted as the new object position, new SIFT points in the vicinity are added to the model, and points with low scores are removed from the model (d).

lel for increased robustness, it is much faster than any of the other trajectory generating methods discussed so far. If tuned for a tracking quality that is comparable to that of DRAGON [Karrer et al., 2008], exceeding that of *DimP* [Dragicevic et al., 2008], and just short of that of *Goldman's DMVN* [2008], the algorithm achieves a speedup factor of 58 over DRAGON's optical flow approach, thus making interactive operation without pre-processing possible. The main reason for the performance gain is that only a sparse set of features are detected in each frame and only those in the vicinity of the tracked object are actually considered for further processing. Another benefit of this method is its improved scalability with video resolution, following a less than quadratic curve (see [Wittenhagen, 2008], p.85).

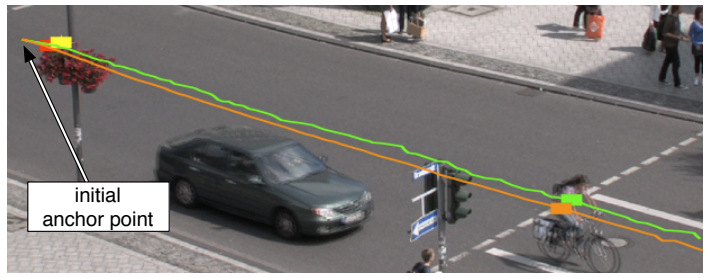

Figure 4.25: DRAGONEYE compared to other tracking techniques for DMVNs in the situation of object occlusion. Both dense (DRAGON) and sparse (*DimP*) flow estimation techniques are susceptible to losing the object; DRAGONEYE's model-based approach is much more robust. Image adapted from [Wittenhagen, 2008]

Generating Object Trajectories by Measuring the Structure at Capture Time

Even better trajectories could be created if the scene structure was completely determined.

The approaches for estimating parts of the geometric scene structure discussed above help to generate better trajectories—even through phases of occlusion of objects or optical sampling artifacts like motion blur. They therefore allow us to establish a more precise semantic mapping than the structure-agnostic methods, and some even do so at greater run-time efficiency. Still, it seems a valid argument that this idea can be pushed even further towards completely recovering the scene geometry and thus towards a solution of the object scoping and spatial trajectory frequency problems. Finding suitable ways to recover the structure of uncalibrated video scenes, however, has long been an unsolved problem and still is an active research area of computer vision (cf. [Hartley and Zisserman, 2003]).

Depth measuring cameras can give reliable hints about the geometric structure.

With the relatively recent advent of available hardware to acquire not only a full color shot of a scene but also an accompanying depth map, this situation may be remedied in the near future. Devices like the Microsoft Kinect™[4] now allow such capturing of videos and geometry at the same time while being affordable and easily obtainable. In the next years, we may even see standardized video formats

[4] http://www.microsoft.com/en-us/kinectforwindows

that include a depth track like most videos today include audio tracks. We hypothesize that, with the geometry information these depth cameras provide, standard image space tracking algorithms like the ones described above could be improved in terms of both speed and accuracy; object scoping, occlusion, and interactions could be detected and modeled much better using the depth map data. First approaches in this direction, for example, [Izadi et al., 2011] and [Newcombe et al., 2011], are certainly promising.

We conducted a formative study to gain insight into the possibilities of adding depth channel information to textbook tracking algorithms. The study was carried out by Dennis Lewandowski in the context of his Bachelor's Thesis [2011] under the guidance of the author. Two different approaches were tested; both were extensions to the OpenCV [2000] implementation of the CAMShift [Comaniciu et al., 2003] algorithm, where we injected the depth information at different stages into the processing chain.

Augmenting CAMShift histograms with an additional depth channel can help disambiguating similar colors.

The first includes the depth channel into the color histogram as another independent color component. The mathematical structure of the algorithm allows, in theory, to work on an arbitrary amount of data channels as long as a number of assumptions hold: for example, that the object can be characterized by a histogram over these channels and that directed distances in the combined histogram space can be defined. While this naive modification of the algorithm will clearly have some drawbacks—the depth value of an object in the scene already violates the constraint that it should be characteristic for the object if it moves from the back to the front of the scene—it can help us to understand if and how the added depth component changes the algorithm in occlusion situations or when objects have similar colors.

Using the depth channel to separate object from background can improve the performance of CAMShift.

The second approach uses the depth channel to find the boundaries of the tracked object in the histogram creation stage of the original CAMShift algorithm. It thus allows to classify the pixels in the selected region of interest (ROI) into pixels belonging to the object and pixels belonging to the background; the color histogram can then be defined using only pixels that have been classified to lie on the object while the background pixels are being masked out. Additionally, the same mask can be used to reject areas in the histogram's back projection that indicate a high probability

of the object due to image noise but are in the wrong depth range [Lewandowski, 2011].

Both approaches could be shown to enhance CAMShift tracking, but both still have drawbacks.

Lewandowski evaluated both techniques with eight different videos and showed that for most scenes the tracking performance of the standard CAMshift algorithm without depth information could be significantly enhanced by either of the depth-augmented versions. Especially the second approach generated promising results, while the first—as already predicted above—did not improve the tracking if the object significantly changed its position in the scene along the optical axis of the camera. A full report of the results and detailled explanations of the data can be found in Lewandowski's Bachelor's Thesis [2011].

Further research is necessary to integrate depth information with standard tracking algorithms.

This approach of using advanced hardware to measure the scene geometry at the same time as capturing the video itself seems very promising. The benefits that we can expect from this or similar techniques include better handling of occlusions, resolution of object scopes, and better tracking in difficult lighting situations. Extending other standard algorithms than CAMshift through depth data, for example, a depth-discontinuity respecting optical flow like the one developed by Weiß [2007], remains for future work. Similarly, integrating the depth-enhanced tracking into our existing DMVN algorithms [Karrer et al., 2008; Wittenhagen, 2008] is yet to be done.

Generating Object Trajectories by Letting the User Give Cues About the Structure

The fourth and last method to generate the object trajectories for establishing the semantic mapping that we will discuss here, is to offload the burden of giving hints about the scene structure to the user. While at first, this sounds like an idea which is contrary to a number of usability concerns, we argue that keeping the user 'in the loop' by giving them this bit of additional control can actually be beneficial if the resulting trajectories are closer to the user's conceptual model of the task at hand.

Goldman's particle-tracking-based DMVN [2007; 2008], for example, allows users to paint over the object of interest

in a frame of the video, thus allowing the particle clustering algorithm (cf. 4.2.2 "Generating Object Trajectories by (Partly and Approximately) Recovering the Structure") to merge the clusters that belong to the object. This reduces the tendency of objects to fall into multiple clusters. Similarly, the initial selection of an object of interest in DRAGONEYE [Wittenhagen, 2008] can optionally accept not only a single point location but also a bounding box to obtain a better estimate about which feature points to include in the object's bag-of-features model (cf. 4.2.2 "Generating Object Trajectories by (Partly and Approximately) Recovering the Structure"). Both are examples where users provide extra information about the structure of the scene through more complex selection mechanisms; and in both cases the information is used to generate trajectories that better reflect the users' intent.

In Goldman's DMVN, users have influence on the particle clustering by making more expressive object selections.

In her Master's Thesis, Alisa Novosad [2012] formally analyzed the effect such a bounding box or paint-over selection of objects has on the generated trajectory in contrast to single point object selections like we use in DRAGON. We developed a version of the DRAGON DMVN for this purpose where multiple seeds can be inserted into the optical flow field at the same time, generating multiple pixel trajectories simultaneously. An area that has been selected by the user through either of the above techniques is stochastically sampled and optical flow trajectories of the individual pixel samples created. These trajectory fragments generated from each sample are then clustered in affine motion space and fragments belonging to the background rejected. The final object trajectory is a weighted average of the remaining foreground trajectory fragments. Unlike Goldman's clustering approach [2008] that tries to group all particle traces in the frame, this localized clustering has the advantage that the number of clusters can be limited to two—foreground and background—, better justifying the use of a simple k-means algorithm with a fixed number of clusters. Novosad showed that, with her approach, the additional selection information supplied by the user considerably increases the tracking robustness of the optical flow tracking in the presence of input noise [Novosad, 2012] (Figures 4.27 and 4.28).

Novosad coalesces multiple single-pixel trajectories inside a bounding box selection for added robustness.

This result shows that such methods that rely on user-defined structural cues can help with improving the tracking quality. The other aforementioned problems of scop-

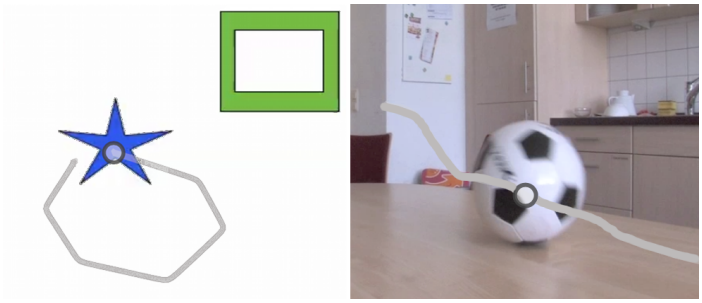

Figure 4.26: Two sample videos used in the experiment by Novosad. The synthetic test video shows two non-convex polygons—a frame and a star—that perform circular motions. The real test video shows a soccer ball rolling through the scene. Source: [Novosad, 2012]

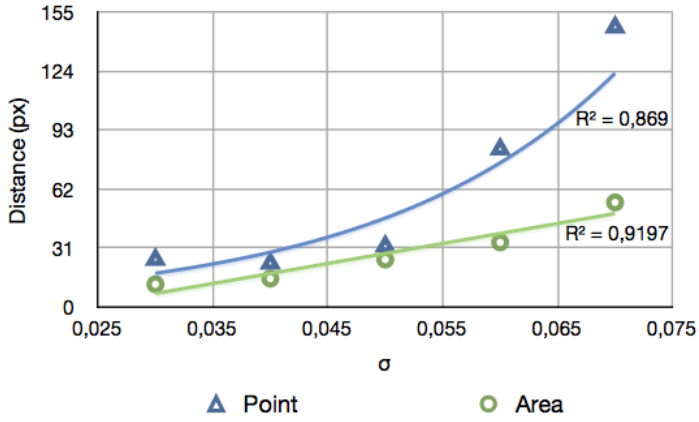

Figure 4.27: Average point wise distance between the generated trajectories and hand labeled ground truth data for a synthetic test video (Figure 4.26) for point and bounding box selections. In both cases, the trajectory deviates more strongly from the ground truth for less precise selections (simulated by AWGN overlaying the location/size of the selections). The overall error is much smaller with the area selection techniques. Source: [Novosad, 2012].

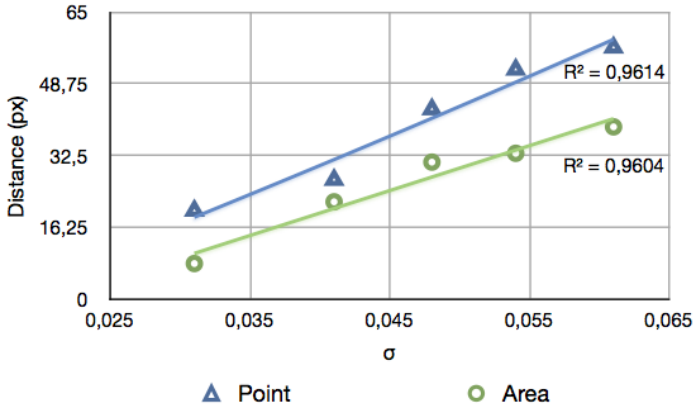

Figure 4.28: Average point wise distance between the generated trajectories and hand labeled ground truth data for a real test video (Figure 4.26) for point and bounding box selections. In both cases, the trajectory deviates more strongly from the ground truth for less precise selections (simulated by AWGN overlaying the location/size of the selections). The overall error is much smaller with the area selection techniques. Source: [Novosad, 2012].

ing and occlusion are also mitigated: Scoping is changed from an implicit mechanism of the tracking algorithm to something the user does on her own accord by selecting the relevant parts of an object. This is ideal, because in this way, the generated trajectory is much more likely to represent a semantic mapping into the 'correct' and therefore task-relevant semantic structure of the medium. At the same time, the averaging or smoothing that happens in all algorithms above, when areas are selected instead of mere points, often leads to a much better trajectory representation of objects with strong internal motion like, e.g., the rolling soccer ball in Figure 4.26. Excessive high spatial frequency motion in the trajectories that is caused by noise naturally occurs to a lesser degree as a side effect. This smoothing effect, however, is implicit and thus cannot be controlled by the user except through the selection of larger or smaller regions, which may be a disadvantage. The same averaging mechanism also helps to track objects across partial occlusions, whereas full occlusions remain a problem that can only be solved by building up a stateful model of

Allowing users to give structural cues improves DMVN tracking in terms of semantic scoping, robustness, and trajectory quality.

the scene structure as described in some approaches above (cf. 4.2.2 “Generating Object Trajectories by (Partly and Approximately) Recovering the Structure”).

All these techniques, of course, require the interaction to be adapted to the extended information input needs of the algorithms. As such, they are directly tied in with the design of the interface; we will discuss the implications regarding the interaction in the section 4.2.3 “Interactive Scoping and Trajectory Filtering” below.

4.2.3 Designing the User Interface for Semantic Video Navigation

After we have identified a suitable conceptual model for the users’ task and facilitated functional access to the according semantic structure of the medium by establishing the semantic mapping, we can now design the interface and interaction technique that allows the users to semantically navigate a video. We will first describe how the interaction works across most existing DMVNs, and then analyze how each of these systems solves some common interaction problems that arise during semantic video navigation. In this context, we specifically discuss the solutions that we have developed for our DRAGON DMVN system.

DVMN Interfaces and Interaction Techniques

DMVNs must allow to specify at least an object and its target location.

If we recall the setting outlined at the beginning of this chapter, our goal was to allow semantic navigation through specifying the spatial configuration of an object of interest inside the video scene. Naturally, any interaction technique that serves to accomplish this must allow the user to specify at least two things: it must be possible to indicate the object of interest through some selection mechanism, and it must be possible to express the desired spatial location of this object, either as an absolute position or in relation to another object in the scene.

We can easily see that these two information needs for the semantic navigation user interface are exactly fulfilled by

standard direct manipulation dragging interfaces, supporting our arguments regarding the suitability of direct manipulation for semantic navigation interfaces from the theory chapter (2.2.3 "Combined Model" (p. 50)). Consequently, such a simple point-and-drag interaction technique is employed in most DMVNs ([Kimber et al., 2007; Dragicevic et al., 2008; Karrer et al., 2008]):

This is compatible with standard direct manipulation interfaces.

First, the user clicks on the object of interest, thereby selecting one of the set of possible object trajectories that represent the semantic mapping functions.As explained above, there exist a number of valid approaches of how this single-point selection can be interpreted and how a trajectory is generated from this information—injecting a trajectory seed at the pixel coordinates for the current frame into the optical flow volume and growing the trajectories in both directions through bi-directional flow vector interpolation is what we use for the standard implementation of DRAGON. Independent of the technique to generate the mapping, the resulting trajectory can be represented as an ordered set T of (t_f, t_p)-tuples that indicate the position $t_p = (x, y)$ of the object for each frame time t_f of the video scene. The trajectory can optionally be visualized as a path that is overlaid on the video by projecting the three-dimensional structure onto the two-dimensional $X \times Y$-plane of the frame (Figure 4.29). We will discuss different ways of visualizing the trajectory later in section 4.2.3 "Visualization Options".

After selecting an object, its trajectory is generated as a list of time-space tuples.

After the object has been clicked on and its trajectory has been generated, the user can now specify its desired spatial location by dragging the object along its trajectory. This is the point where the semantic mapping is needed to associate the correct video time or frame number with each new dragging event location: While the object is being dragged, the video's current time is adjusted in such a way that the object stays as close as possible to the desired location.

While dragging the object, the semantic mapping determines the correct frame to display.

This is done by repeatedly minimizing some distance measure d over the full length of the object's trajectory to determine the next frame f that should be displayed during the interaction:

$$f(p, T) = [\underset{t \in T}{\operatorname{argmin}}(d(p, t))]_f$$

The distance between the spatio-temporal dragging location and the trajectory is continually minimized.

where p is the screen position the user is dragging the object to and $T \subseteq F \times P$ is the object's trajectory consisting

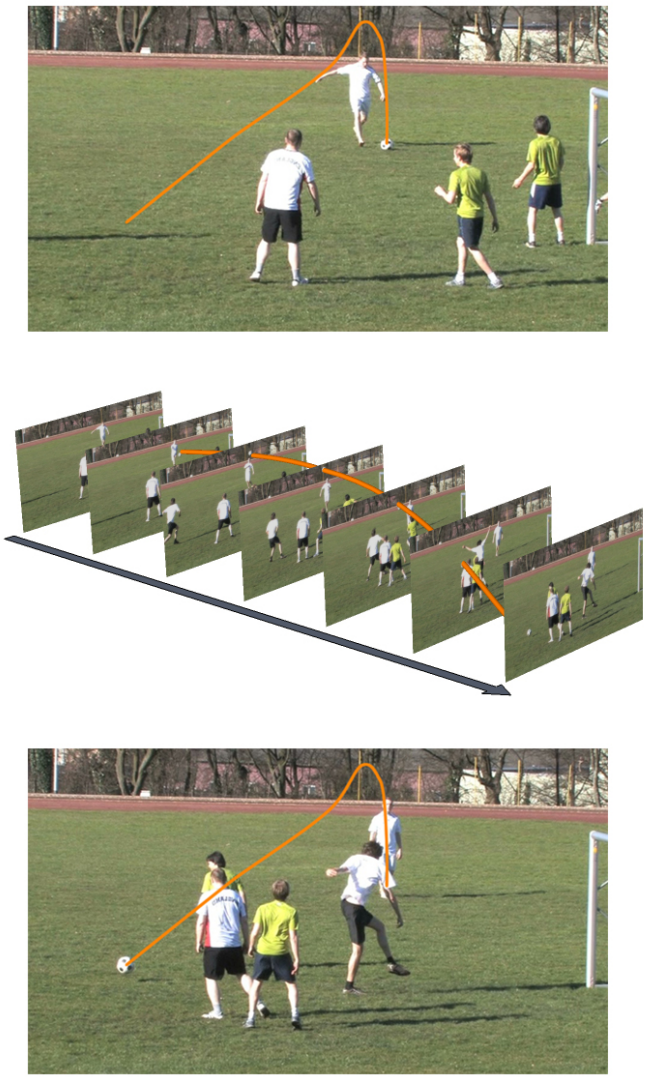

Figure 4.29: Trajectory of an object in the scene projected onto the currently visible frame. The three-dimensional trajectory of the ball (middle) can be visually represented as a path overlay over each frame. The top image shows the first frame, the bottom image the last frame of the scene.

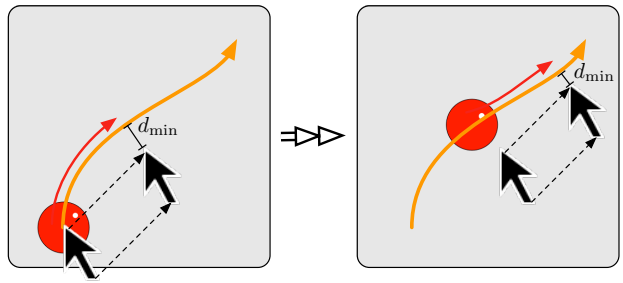

Figure 4.30: Continuous minimization of the distance between the object and the dragging position. In response to each dragging event, the video displays the frame where the distance between the event location and the object's trajectory is minimal with respect to some distance measure d. The euclidean line-point distance shown here is simple to compute but can cause problems with self-intersecting trajectories.

of tuples (t_f, t_p) of a frame number and a position [Karrer et al., 2012]. For each dragging event, the new frame to be displayed is determined by this formula before the navigation is performed by updating the video view and updating the context of the calculation to reflect that the currently displayed frame has changed. This procedure is then repeated for the next dragging event but in the context of the new temporal position (Figure 4.30).

Common Interaction Problems

The object cannot generally follow the dragging precisely.

The 'ideal' interaction, one might assume, would have the object following the dragging position precisely. This, however, is possible only if the dragging operation follows the trajectory exactly—other locations may not have been occupied by the object at any time in the video and are thus obviously not navigable.

Interaction problems occur at points where the semantic mapping is not uniquely defined.

This observation gives rise to at least two interaction design problems, both corresponding to cases where the semantic mapping, which should be the inverse of mapping constituted by the object trajectory, is mathematically ill-defined:

1. The indicated location is not covered by the object's trajectory.

2. The indicated location is part of the object's trajectory more than once.

In the first case, the initial mapping cannot be reversed because it is not a surjective function; generally, an object does not occupy every location in the scene at some time. As mentioned above, we therefore use a distance measure d to navigate to a point in time where the object comes as close as possible to the indicated location p.

If the semantic mapping associates two points in time with the same object location, this *temporal ambiguity* has to be resolved.

In the second case, the initial mapping cannot be reversed because it is not an injective function. Objects often occupy the same location at different points in time; an inverse lookup over the trajectory thus yields *temporal ambiguities*. In [Karrer et al., 2012], we have reported on three typical object motion patterns that lead to this behavior (Figure 4.31):

1. "*Recurring movements*, e.g., the rotating hands of a clock or a ball bouncing up and down." (Figure 4.31, left)
2. "*Self-intersecting trajectories*, e.g., a looping roller coaster or a car heading straight towards the camera." (Figure 4.31, middle)
3. "*Pauses*, e.g., a model walking down the catwalk, striking a pose and holding it for a few seconds before moving on." (Figure 4.31, right)

All three examples illustrate situations where a position an object is being dragged to will resolve to multiple points in time with our suggested semantic mapping.

In some cases, the distance measure can be augmented to allow disambiguation.

Clearly, we have to find some way to disambiguate between these results: For the first two motion patterns from the list above, we can do this by incorporating the current temporal context into the distance measure d that we minimize to find the video frame to navigate to. Below, we will examine possible distance measures and how they affect the semantic navigation behavior of a DMVN to demonstrate different solutions. For the third case—where objects pause their movement and thus remain at the same location for an extended period of time—, we will see later (cf. 4.2.3 "Object Pauses") that we need to make some alterations to the basic DMVN interaction.

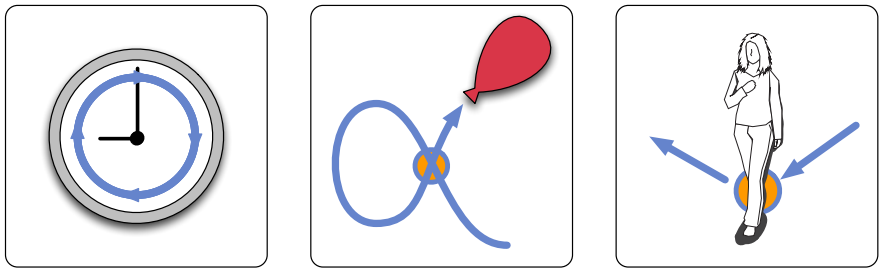

Figure 4.31: Three different examples of temporal ambiguities that can arise from object motion patterns: *recurring movements* (left), *self-intersecting trajectories* (middle), and *pauses* (right). In all three cases, the semantic mapping is ill-defined.

Distance Measures and Their Influence on Temporal Ambiguities

The most basic distance measure relies on euclidean spatial distances only.

We have explained above how in all DMVNs the next frame to be shown when dragging an object through the scene is determined by a distance function d. The most straightforward way to choose such a distance function is probably to define it purely in *spatial terms*. Goldman's [2008] DMVN implementation, for example, uses such a simple euclidean distance in image space:

$$d_1(p,t) \cong \|p - t_p\|$$

With this distance measure, the system will always display the frame where the object's position is closest to the mouse cursor. Note that this function does not depend on the temporal location of the frame that is displayed at the time the function is evaluated.

An euclidean distance measure does not guarantee temporal coherence during the interaction.

Interaction problems may arise, however, if the object is dragged along its trajectory, and the dragging point moves closer to a part of the trajectory that maps to a much later or earlier time point in the video. In these cases, the video does not 'follow' the spatial and temporal progression of the object through the scene, but the navigation exhibits incoherent temporal jumps (Figure 4.32). While this behavior enables a user to quickly reach the time for a certain location of the object, the jumps are almost always confusing

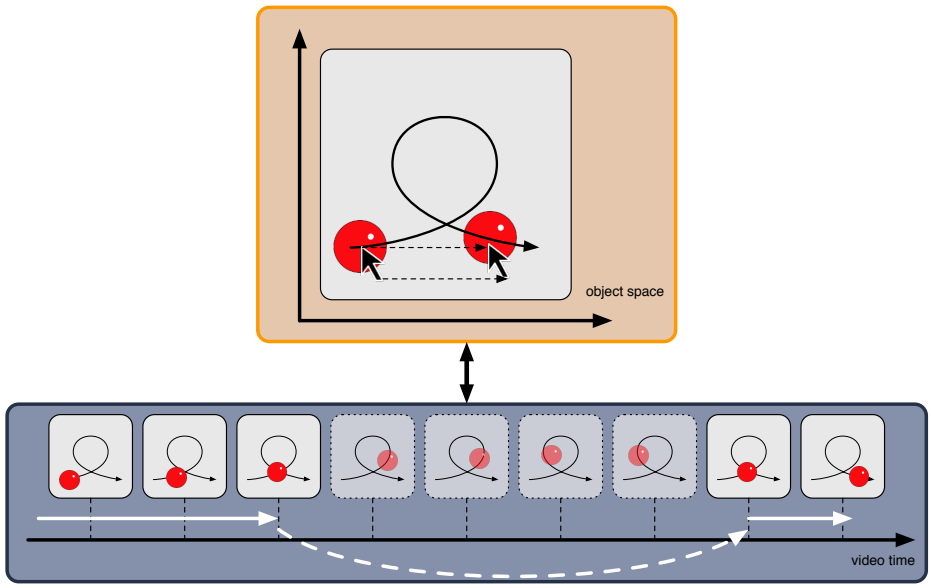

Figure 4.32: Purely spatial distance measures may cause temporal jumps in the video during the interaction. If the object in the image is dragged straight to the right, the spatially closest positions never include those on the loop; the according frames are consequently dropped during the navigation, and jumps occur.

and accompanied by a loss of context on the side of the user regarding what happens around the object in the scene [Karrer et al., 2012; Brockly, 2009].

Euclidean distance alone cannot resolve temporal ambiguities.

Another problem can occur when the object exhibits any of the motion patterns described above that cause the trajectory to self-intersect or fold back on itself. Minimizing the euclidean spatial distance measure may then yield multiple time points, and there is no reliable way to select the single one that would correspond to the user's conceptual model of the object's motion. Again, this can result in seemingly random temporal jumps during the navigation or in unintended reversal of the temporal direction of the navigation: dragging the minute hand of the clock in Figure 4.31 to the far right side of the clock's face, for example, can result in any of the multiple temporal locations where the time is 15 minutes after the full hour. A similar temporal ambiguity presents itself when dragging objects away from cusp points, where the direction of the trajectory is suddenly reversed (Figure 4.33).

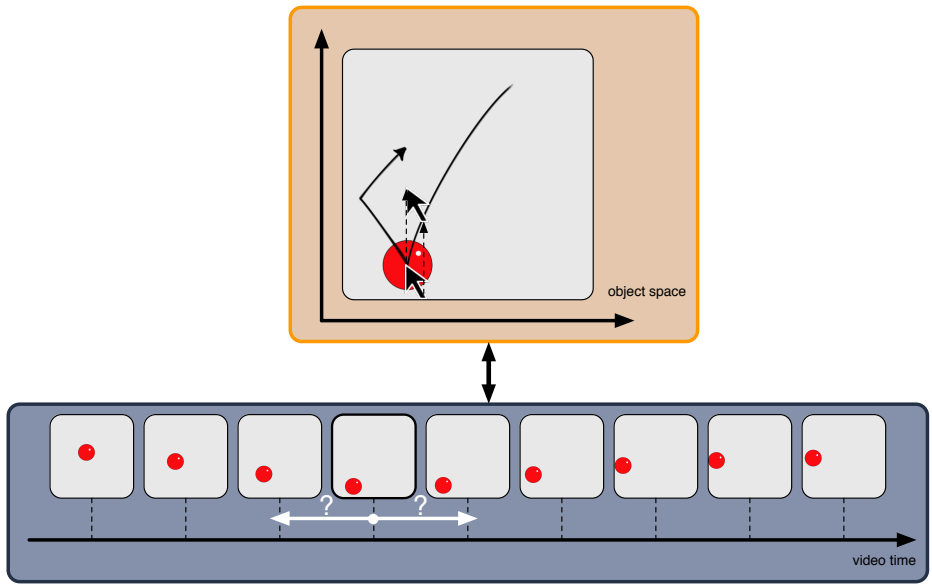

Figure 4.33: Purely spatial distance measures may cause directional ambiguities during the interaction. When dragging an object out of a cusp point in the trajectory, a spatial distance measure cannot consistently pick any of the two temporal directions. The video may move forward or backward through time, taking the incoming or outgoing branch of the trajectory, or it may arbitrarily switch between the two.

Arc-length distance is still spatial but can guarantee directional continuity and temporal coherence.

Other DMVNs use different distance measures that make such frame jumps and direction reversals less likely in order to create a more coherent navigation experience. The authors of *DimP* [Dragicevic et al., 2008], for example, propose 'directional continuity' and 'arc-length continuity' as two requirements for this kind of semantic video navigation. Consequently, they introduce appropriate terms into their distance function:

$$d_2(p,t) \cong \|p - t_p\| + \|\text{arclen}(t_p, o_p)\| + k_D$$

with o_p being the object's position in the currently displayed frame and a bias offset $k_D > 0$ being added whenever the arc-length changes signs from the last step.

Although the distance measure technically does not include any temporal terms, the arc-length distance between the current position and the target position of the object spatially reflects the temporal ordering of the object locations

along the trajectory. The euclidean distance measure above ignores this information. The result of these modifications is that the navigation is far less likely to 'lock on' to temporally distant parts of the trajectory if they are spatially close or even if they are intersecting like in the looping balloon example in Figure 4.31. Also, the directional term addresses the problem of reversing the direction of the navigation at cusp points like in Figure 4.33.

In addition to spatial terms, we can include temporal terms in the distance measure.

While both of these distance measures, d_1 and d_2, are only defined in terms of spatial positions, the latter already makes use of the temporal information in the trajectory—albeit through a spatial encoding—in using an arc-length distance. The two remaining existing DMVNs, DRAGON [Karrer et al., 2008] and *Trailblazing* [Kimber et al., 2007], are different in that regard as they use this temporal information directly in the distance function. We can therefore classify distance functions as either *purely spatial* measures that depend only on the positional projection t_p of the trajectory mapping or *spatio-temporal* measures that depend on both the positional component t_p and the temporal component t_f [Karrer et al., 2012].

DRAGON uses a 3D spatio-temporal distance measure that allows handling of temporal ambiguities and avoids frame jumps.

The approach used in our DRAGON DMVN defines the distance measure over the spatio-temporal video volume constructed as the direct sum of the pixel-based image space and the frame-based timeline (Figure 4.34). As such, DRAGON belongs to the *spatio-temporal* class of DMVNs, with each calculation including the current frame number o_f:

$$d_3(p,t) \cong \|[p_x - t_{p_x}, p_y - t_{p_y}, \alpha \cdot (o_f - t_f)]^T\|,$$

where α is a cross-weighting factor that is tuned according to the resolution and the frame rate of the video. Interaction with DRAGON does not suffer from problems with self-intersecting or recurring movement trajectories, because the temporal component disambiguates possible navigation target candidates.

The object can be dragged around trajectory concaves like it was attached to the pointer with a rubber band.

The effect of this combined spatio-temporal function is that the user perceives the object to be attached to the mouse pointer with a 'rubber band': The temporal component $o_f - t_f$ of the distance term forces the navigation to happen temporally continuous and in the correct order. If the trajectory forms a bulge like in Figure 4.35, the object first

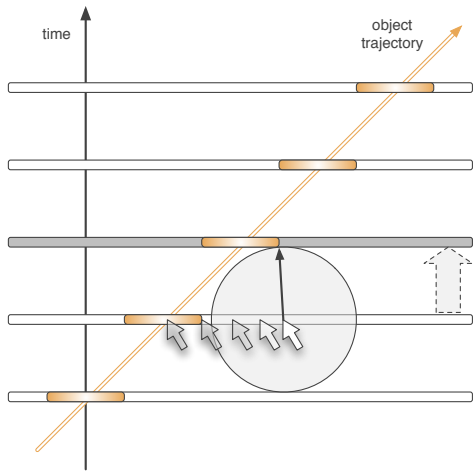

Figure 4.34: Spatio-temporal distance measure in DRAGON. The image shows a top view of a 'stack' of video frames with an object trajectory. The distance measure corresponds to the radius of a spatio-temporal hypersphere that has its center spatially at the drag location and temporally at the current frame.

gets caught in the bend until the spatial part of the distance outweighs the temporal part; then the object 'snaps' around the bulge until the distance is minimized again (Figure 4.35 (d)). The system also exhibits this behavior at points where the object pauses.

DRAGON resolves directional ambiguities through different interaction techniques.

The lack of a directional term, however, means that the directional ambiguity problem in the presence of trajectory cusps remains and needs to be handled separately in the interface. We have developed three possible solutions to this problem for DRAGON, which have been described by Christian Brockly in his Diploma Thesis [Brockly, 2009]:

1. *Directional continuity*
   When the object reaches a cusp in its trajectory during the navigation, the search space for the minimization of d is truncated; the part of the trajectory that has just been navigated will be skipped until the next cusp—or the same cusp, but from the other direction—is reached. The result is similar to that of including a directional term into the distance function, like *DimP* does, but computationally more efficient, because only parts of the trajectory have to be

Cusps segment the trajectory, and segment transitions are unidirectional.

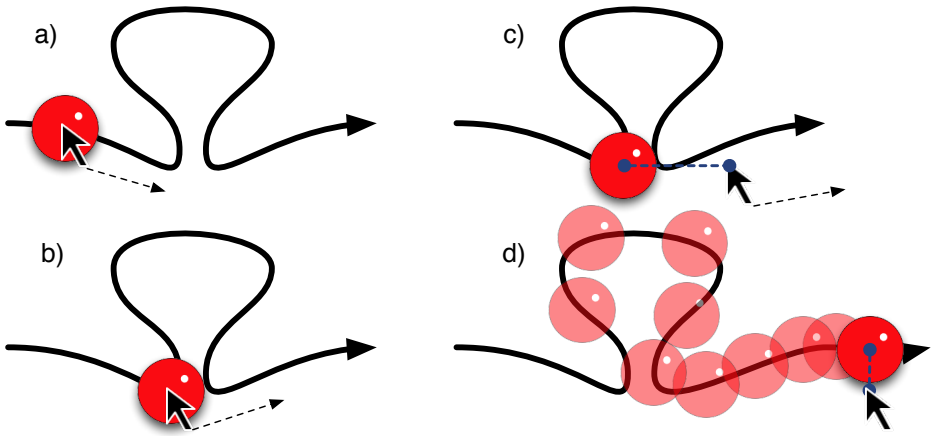

Figure 4.35: DRAGON's spatio-temporal distance measure at non-convex points in the trajectory. Where a purely spatial distance measure would cause the video to skip the 'bulge' when the object is dragged across the lower part, interaction with DRAGON feels more like controlling the object through a rubber band. As long as the distance can be minimized trivially, the object stays fixed to the mouse cursor (a). At trajectory concaves (b), the object fist 'hangs' (c) before the spatial component in the distance measure outweighs the temporal one, and the object snaps towards the mouse cursor (d).

inverted. On the other hand, this approach is more difficult to implement, and it relies on cusps to be detected reliably.

Objects are given virtual inertia, and they can be 'thrown' through cusps and across occlusions.

2. *Temporal navigation inertia*
Being the method we initially implemented in DRAGON, temporal navigation inertia adds a layer of physical simulation to the interaction. When an object is dragged along its trajectory and then released, it temporarily continues to travel in that direction while losing momentum all the time and finally coming to a halt. The technique is—and feels—similar to the inertial scrolling mechanism that can be found in a number of current graphical user interfaces, for example in table views on Apple's iOS[5]. In this way, users can drag an object towards a cusp, release it, let it travel through the cusp while retaining the desired temporal direction, and re-acquire it again after the cusp to continue the navigation by dragging again.

[5]http://www.apple.com/ios

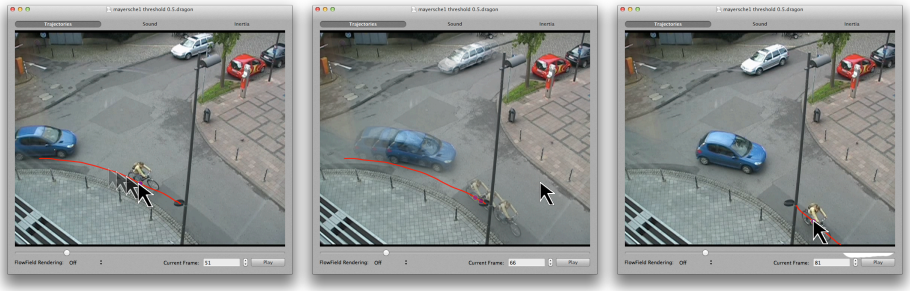

Figure 4.36: Temporal navigation inertia in DRAGON helps to navigate gaps in the trajectory. When temporal inertia is enabled, objects can be 'thrown' over gaps (left and middle) as they are caused by occlusion problems. After 'catching' the object again (right), the trajectory is re-calculated so that it shows the object's path after the gap.

An added beneficial side effect of this technique—and one of the reasons why this is the method we implemented in DRAGON—is that navigation inertia can also mitigate the interaction problems in the presence of malformed trajectories caused by tracking errors. If, for example, a trajectory ends prematurely, because the object was occluded in a scene for a number of frames, the object can be 'thrown' over this gap in the mapping (Figure 4.36). When it is re-acquired after the gap, the new click on the object selects a new trajectory to define the semantic mapping for the continuing dragging navigation. This new trajectory is not impeded by the tracking gap in the direction of the navigation.

3. *Explicit direction selection through temporary overlays*
   The third solution to the ambiguity problem at cusp points relies on a temporarily overlaid crossing-based interface that lets the user explicitly choose the temporal direction in which to leave the cusp. When the object is located at the cusp position, two semi-transparent crossing targets, each representing a possible temporal direction, are faded in (Figure 4.37). The user can then drag the object out of the cusps by dragging the pointer across either of the overlays to indicate the desired direction.

   Directional ambiguities are explicitly resolved by the user via a crossing-based interface.

   This technique is inspired by *Flow Menus* [Guimbretiére and Winograd, 2000] and possesses the same

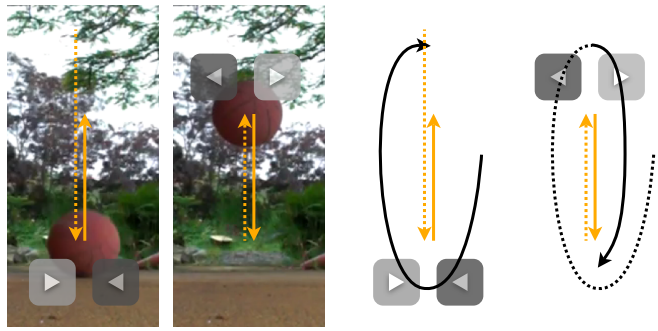

Figure 4.37: Temporal direction made explicitly selectable by overlaid crossing labels. The labels are arranged in a way that a clockwise circular motion always means temporally *forward* while anti-clockwise motion means *backward*.

advantage of turning into a gesture-based interface without the need for visual feedback through habituation. However, it requires the user to explicitly indicate the desired temporal direction in a syntactic way. This means, of course, that she must *know* the current temporal direction beforehand; such knowledge of the syntactic structure, on the other hand, is exactly what we seek to render unnecessary with semantic video navigation interfaces. Brockly thus proposed an alternative way to visualize the directional choices by making the part of the trajectory that lies in the currently selected direction visually distinct and optionally hiding the crossing area labels. This change again embodies the idea of semantic video navigation by making the syntactic temporal structure of the medium accessible by spatial direct manipulation. Also, with the changed visualization, user performance in controlled experiments increased dramatically [Brockly, 2009].

Trailblazing uses a spatio-temporal distance measure that also enforces directional continuity.

A fourth spatio-temporal distance measure was proposed by Kimber et al. [2007]; it combines temporal and directional terms to avoid frame jumps during the navigation and to ensure directional continuity at cusps.

$$d_4(p,t) \cong c_\theta \cdot \|p - t_p\| + \|o_f - t_f\| + k_D$$

with k_D defined in the same way as in *DimP*'s distance measure d_2. Generally, this approach behaves like d_3 but includes a time-dependent term c_θ that makes the tendency to 'snap' across trajectory bulges or object pauses depend on the interaction time: the longer the dragging takes, the more likely the object is to move across the interfering geometry.

Object Pauses

Temporal ambiguities due to objects pausing have not been discussed so far.

As we have shown above, the different distance measures that existing DMVNs employ in the optimization process governing the dragging interaction can take care of some of the temporal ambiguities that arise in semantic video navigation. One special case, however, was left out of the discussion deliberately: that of an object being located at the same location at multiple points in time because the object pauses its movement (cf. 4.2.3 "Common Interaction Problems" and Figure 4.31). We describe this situation in [Karrer et al., 2012]:

> "Ideally, when interacting with an object, a DMVN technique should allow access to those parts of the video where the object is pausing: In our example from above—a model walking down the catwalk, striking a pose and holding it for a few seconds before moving on—, assume that we drag the model into the pose and then want to navigate to a frame *inside that pause* where the lighting is perfect or the model assumes a certain facial expression. This is a situation in which most strategies of existing DMVN systems fail."

Object pauses are neither visible nor navigable in most DMVNs.

It is clear that a pause only occupies a single pixel on the movement trajectory but extends indeterminately in the temporal space (Figure 4.38). Therefore, a pause is neither represented explicitly in the visualization nor in the navigation interaction: in most DMVNs it is invisible and non-navigable. We will first briefly revisit the four distance measures d_{1--4} in the context of this problem before analyzing

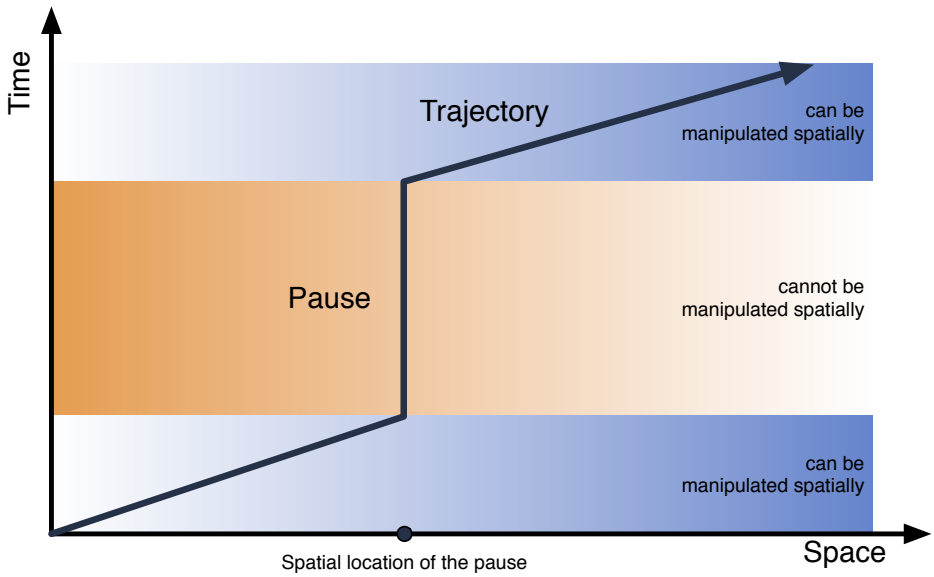

Figure 4.38: Schematic relationship between space and time in object trajectories. We can influence the temporal position of an object in the video through its spatial location as long as it moves. During pauses, the spatial projection of the trajectory collapses into a single point.

two approaches that require modifications to the interaction at higher layers of the interface language.

Spatial distance measures skip the pause entirely.

Both *purely spatial* distance measures (d_1 [Goldman et al., 2008] and d_2 [Dragicevic et al., 2008]) obviously cannot model the concept of a pause, because they do not even access the temporal structure of the trajectory: The two pixels on the trajectory spatially 'before' and 'after' the pixel that marks the location of the pause directly map to points in time temporally 'before' and 'after' the pause. The mapping of the pixel at the location of the pause is, of course, undefined (cf. 4.2.3 "Distance Measures and Their Influence on Temporal Ambiguities"). Thus, those time ranges where an object pauses are not accessible—users have to resort to using other tools, like the timeline slider, to navigate inside a pause. When dragging the object along the trajectory across the pause location, the pause is skipped completely and can only be detected by the user through the sudden changes in the rest of the frame that may be caused by the temporal jump. This means that in the presence of pauses, *purely*

spatial distance measures do not allow to access the frames inside the pause, and the frame jump disorients the user.

Spatio-temporal distance measures can acknowledge the presence of the pause but cannot navigate it.

The two *spatio-temporal* distance measures (d_3 [Karrer et al., 2008] and d_4 [Kimber et al., 2007]) can model temporal concepts and therefore can acknowledge the existence of pauses. The two pixels on the trajectory spatially 'before' and 'after' the pixel that marks the location of the pause again map directly to points in time temporally 'before' and 'after' the pause, just like with the *purely spatial* distance functions. However, when dragging the object across the pause position, the temporal component of the distance function will initially outweigh the spatial component—the dragged object will 'stick' to the beginning of the pause. Dragging the object further, the spatial distance component outweighs the temporal one, and the object will 'snap' to a position some time after the pause [Karrer et al., 2012]. Accessing the frames directly after the pause is therefore only possible by first overshooting and then backtracking along the trajectory.

Timeout-controlled mode changes do not deliver an optimal user experience.

Some different approaches have been tried regarding the exact weighting balance between temporal and spatial distance, including adaptive weighting factors that change according to interaction timeouts like d_4 [Kimber et al., 2007] or the *time scale timer* proposed by Brockly [2009]. During user tests performed by Brockly, however, such timeout driven interfaces have—somewhat unsurprisingly—led to confusion and decreased performance on the users' side.

Both kinds of distance measures do not solve the problem satisfactorily.

In summary, for *spatio-temporal* distance functions, the pause is a barrier that cannot be crossed instantly, which allows users to recognize it without effort but also requires additional controls to access content inside the pause and extra effort to navigate to points directly after the pause. Thus, with all proposed distance measures, users either miss everything that is happening in the video while the object stops, possibly failing to notice the existence of a pause at all, or they have to switch to another means of navigation, like the timeline slider, to access the video frames in the pause. Although pauses structure a scene temporally—which should help with navigation—DMVN systems cannot implicitly leverage this advantage but are hindered by them [Karrer et al., 2012].

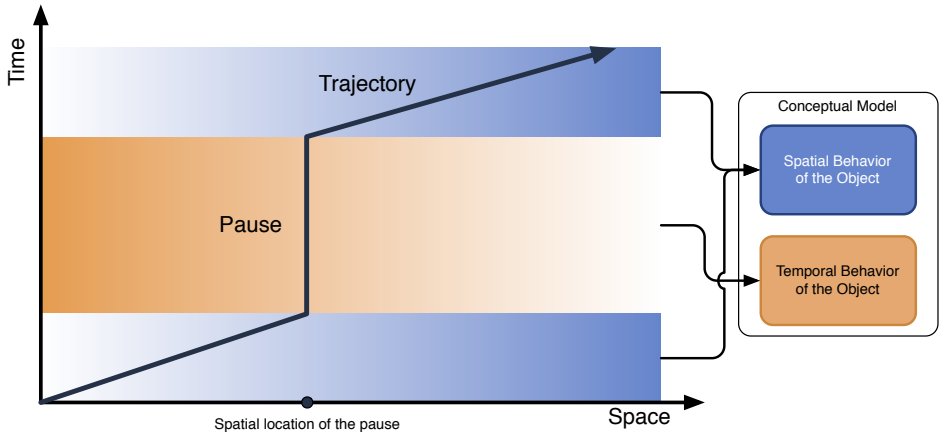

Figure 4.39: Conceptual model spanning multiple domains, depending on the behavior of the object of interest. During a pause, the object only exhibits variation in the temporal domain. There is no reason why temporal concepts like 'waiting' should not be part of semantic navigation.

The real problem lies with the restriction of the direct manipulation interaction exclusively to the spatial domain.

We can see that there is no easy way to address this situation only by adjusting the distance measure. The reason for this is that the actual problem is caused by the very interaction paradigm we chose for our concept of semantic video navigation: direct manipulation. So far, we have regarded direct manipulation only in terms of changing the spatial locations and constellations of objects in the video scene. From this point of view, an object that pauses and does not move gives us no leverage to navigate this purely temporal phenomenon (Figure 4.38)—we have to take a step back instead and realize that during a pause it may not be the *spatial* but the *temporal* progression of an object that forms the conceptual model (Figure 4.39). We thus need to take a wider view of the semantic structure of the medium and, consequently, of the way we are using direct manipulation to navigate this structure. The important part is that the conceptual model, and therefore the semantic structure, is not necessarily fixed to a certain domain but can dynamically shift to another domain depending on the behavior of the objects of interest.

The naive approach to solve this dilemma—to offer an extra tool, for example a timeline slider, that allows to directly manipulate the temporal structure—is clearly not what we should aim for. There is nothing wrong in offering a slider

in parallel, thus allowing access to the syntactic structure as an option; but requiring the user to change tools in mid-navigation just because the semantic structure temporarily unfolds along a different domain than its dominant one seems unnecessary. A better idea would be to dynamically adapt the semantic mapping to reflect this shift of domains: during a pause, the user could still spatially manipulate the object, but the result would be applied to the temporal domain.

If the primary domain of the object of interest in the conceptual model changes, our direct manipulation interface should reflect that change.

We have developed three methods to navigate pauses in DMVNs that follow this idea:

1. *Velocity Slider* [Brockly, 2009]
   This approach was inspired by the *fine slider* [Masui et al., 1995] and the *PVSlider* [Ramos and Balakrishnan, 2003], which both introduce rate-based control to a positional input element. For a DMVN, this concept can be applied to the positional direct manipulation of the trajectory space: Whenever the user crosses the location of a pause on the trajectory during the navigation, the object initially 'sticks' to the pause position, but the time of the video is advanced at a rate relative to the distance between the object position and the current dragging position (Figure 4.40). At that point, the interaction follows a syntactic navigation model. Once the object has 'caught up' with the mouse cursor, the system switches back to our semantic navigation model that uses spatial direct manipulation.

   Integrating rate-based controls is one way to navigate pauses.

   This method has the advantage of mitigating the loss-of-context problems that occur when the pause is skipped [Karrer et al., 2012] and that it can be used to navigate inside the pause. Its disadvantages include the automatic change from direct manipulation, position-based input to the rather indirect rate-based input, which requires some training and can be difficult to master. Also, with the maximum rate being dependent on the position of the pause on the screen—and the rate being rather limited—it can be time consuming to cross longer pauses, and it can be difficult to navigate precisely to the end of the pause where the object just starts moving again.

   Combining direct manipulation and rate-based controls would require a mode switch.

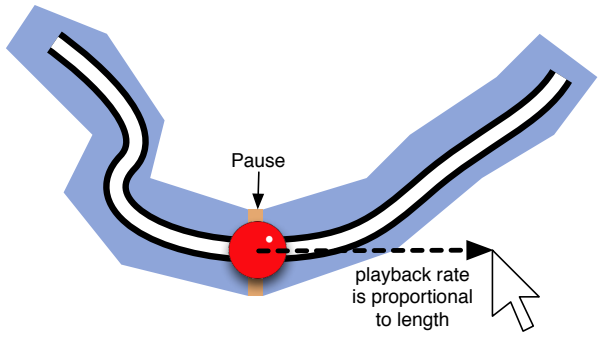

Figure 4.40: Velocity slider interface for pause navigation in DMVNs. Once the object reaches the pause position, the input mechanism changes from spatial direct manipulation to temporal rate-based. At the end of the pause, the interface switches back, and the object re-aligns with the mouse cursor.

A timeline slider can be embedded into the trajectory at the location of the pause.

2. *Embedded Timeline* [Karrer et al., 2012]
A straightforward attempt to get rid of the indirectness of the rate-based control offered by the *velocity slider* is to spatially reify the purely temporal construct of the pause, thereby returning direct manipulation spatial control over the progression of the video to the user. For this purpose, we have proposed to repurpose a small segment of the projected trajectory, both leading up to the pause and following it, to represent a weighted mix of spatial and temporal control (Figure 4.41): Near the location of the pause, the segment is mapped to temporal navigation, thus acting as a timeline slider that is embedded into the projected shape of the trajectory. Near the seams of the segment, it is mapped normally to the location of the object, like the rest of the trajectory is. In between, the segment represents a weighted affine combination of these two mappings with the weighting factors linearly interpolating between the two aforementioned extremes (Figure 4.42).

Over the trajectory, control gradually changes from spatial to temporal towards the pause location and then back again.

When dragging the object towards its pause position, it thus stays at a minimal distance to the dragging position until the special trajectory segment around the pause is hit (Figure 4.43, top). From there, the object will start to overtake the dragging location, moving towards its pause position until it arrives there (Fig-

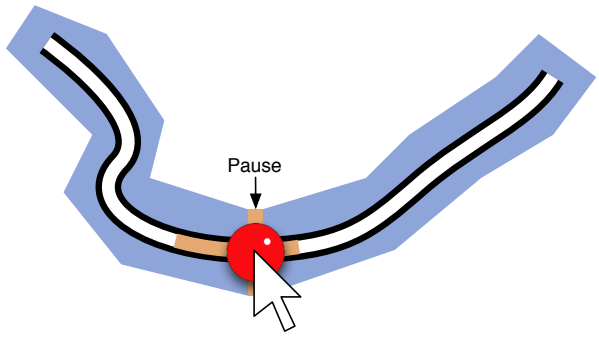

Figure 4.41: Embedded timeline interface for pause navigation in DMVNs. The pause is extended spatially form a single point to a range of the trajectory's visual representation. While this introduces spatial inaccuracies—the object will not follow the mouse pointer precisely in the vicinity of the pause—it dynamically allows full direct manipulation access to the temporal domain.

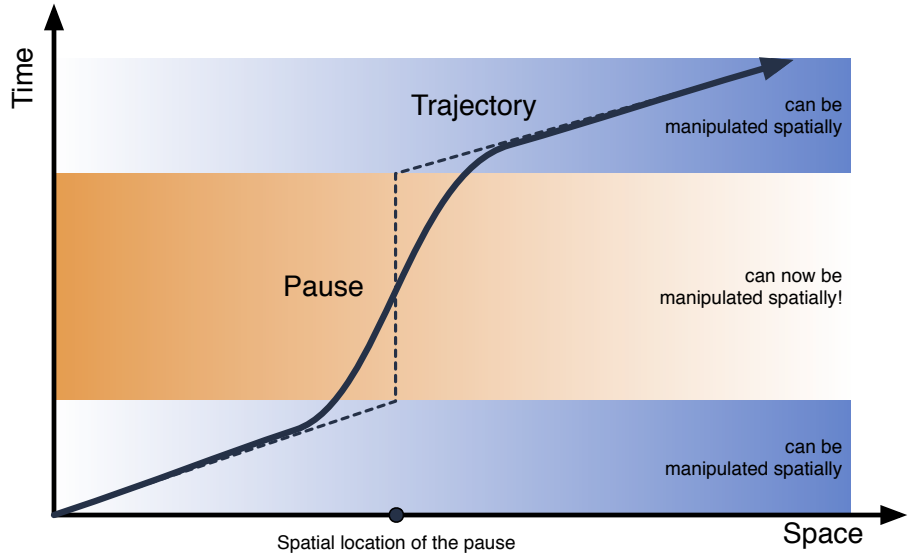

Figure 4.42: Schematic relationship between space and time with the embedded timeline interface. The association between time and space is skewed in a way that any spatial input in the region around the pause location has an effect in both the spatial and the temporal domains.

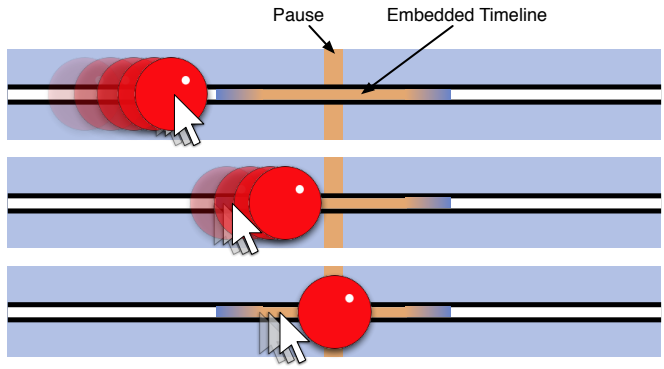

Figure 4.43: Navigation behavior at pause points with the *embedded timeline*. Since the spatial representation of the pause is extended from a single point to a range on the trajectory, the spatial coupling between control location and object location is less tight across the *embedded timeline*.

ure 4.43, middle); at this point, there is still enough space left on the trajectory towards the pause location to map it to the first half of the temporal extent of the pause (Figure 4.43, bottom). The second half of the special segment works analogically but in reversed order.

This method is especially suited for short, frequent pauses but—similar to the timeline slider—increases the syntactic distance.

The advantage of this approach is that by using the existing navigation control space on the trajectory, we do not have to introduce any additional control elements to navigate pauses. This makes re-homing times between UI elements or different navigation tools, like trajectory and timeline slider, unnecessary. The method is ideally suited for short pauses of just a few frames where the introduced discrepancy between the expected direct manipulation mapping and the actual weighted combination of spatial and temporal mappings remains small. For longer pauses, it is still superior to the standard DMVN approaches, because the *embedded timeline* does allow to navigate pauses, but the visual salience of the shift in the semantic mapping might distract the users. Also, if the object pauses its motion multiple times in short spatial intervals, there can be very little room on the trajectory to assign to the special segments; in such cases, the syntactic distance of the *embedded timeline* suffers

in a very similar way as that of a regular timeline slider, like we have discussed above in 4.1.1 "Analyzing the Syntactic Distance of Slider-based Navigation and Selection Interfaces".

One should keep in mind that these disadvantages are merely due to the fact that we are re-purposing existing control space on the trajectory, thereby ambiguating the interaction in these areas. It is not a conceptual problem of our claimed necessity that the interface reflects the shift in domain exhibited by the semantic structure at object pauses.

3. *Loop* [Brockly, 2009; Karrer et al., 2012]
   An alternative approach to offer a spatial direct manipulation representation of the temporal semantic structure inside of object pauses is to make use of the space *next to* the trajectory instead of the trajectory itself. This, of course, opens up a vast set of possibilities regarding the location, shape, and visual gestalt of this space and the interaction element that it contains.

   Instead of utilizing space *on* the trajectory, temporal control can be spatially located *off* the trajectory.

   We can define a number of restrictions that ensure a smooth integration into the DMVN interaction technique and thus narrow down our search for a suitable design: First, we want to have that space topologically reflect the temporal semantic structure; this means that we should restrict ourselves to a linear space. Second, we want to guarantee a smooth transition in terms of the interaction between the trajectory—the interaction element that normally navigates the semantic structure—and the interaction element that navigates time during the pause. Third, we want to allow the user to optionally skip the pause easily in cases where the navigation target clearly lies beyond the pause, and the loss of context caused by the skip is acceptable.

   There are some restrictions on the design of the loop control.

   With these restrictions in mind, we have designed the *loop* interaction element that takes over the navigation every time the domain of the semantic structure shifts from spatial to temporal. The *loop* element is an extension to the shape of the trajectory itself, thereby being compatible with both the visual shape and appearance of the trajectory and the topology of the semantic structure it can navigate. It is a circular loop that starts—and ends—exactly at the location of the pause,

   A loop shape fulfills the requirements and can be flexibly attached to the trajectory.

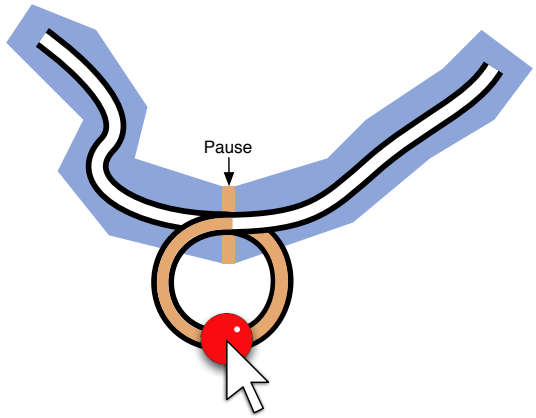

Figure 4.44: Loop interface for navigating object pauses in DMVNs. The loop interaction element makes use of the space next to the trajectory, thereby leaving the time-space mapping on the trajectory intact. The temporal range of the pause is fully mapped to the attached loop. As navigation in the loop is controlled by the same distance function as on the trajectory, the loop acts like a radial controller, with the control granularity being dynamically adjustable via the radius of the circular dragging motion.

which allows the user to smoothly drag the object to this point, then continue in a circular motion, and finally resume dragging along the trajectory at the end of the loop (Figure 4.44).

The loop integrates well with the existing interaction paradigm of DMVNs.

Because the loop is laterally attached to the trajectory and the interaction handled by the same distance measure minimization process, users can also just continue dragging along the trajectory, which makes the object 'snap' to the cursor after a short delay and causes the pause to be skipped. Also, because of the distance measure, the loop can be used as a linear angular control whose granularity can be continuously adjusted by changing the radius of the control gesture [Hürst and Götz, 2008]; it therefore does not suffer the same syntactic distance penalties as a timeline slider or the *embedded timeline* when used with pauses of different lengths.

The loop is especially suited for longer, less frequent pauses.

While the *loop* approach remedies most of the disadvantages of the *embedded timeline*, it is naturally more suited for less frequent but longer pauses because of the visual clutter generated by the trajectory exten-

sion. In [Karrer et al., 2012], we thus argue that a combination of both techniques—using the *embedded timeline* for short pauses and using the *loop* for longer pauses—may be the best way to handle the pause-implicit shift of the semantic structure domain.

Visualization Options

After having discussed the interaction concept for semantic video navigation as well as a number of problems and our solutions, we will now have a closer look at the different visualization options in the interface. Clearly, the video itself should be the main visible element in any video navigation interface, as it is both interaction surface and the main feedback channel that allows users to check if they have reached their navigation goal. There are, however, a number of factors to consider with regard to how we can incorporate affordances or extra information channels into the UI.

There are many options how to visualize trajectories.

Probably the most important information that we have to think about is if and how the trajectory of an object—and thus a representation of the currently selected semantic mapping—can be visualized. This, of course, first and foremost pertains to the spatial progression of the selected object's path; additional information and data, like the velocity of the object or, as a special case, if the object pauses its motion, can then be incorporated into the trajectory visualization or conveyed through other channels.

Brockly analyzed different trajectory visualizations

In his Diploma Thesis, Christian Brockly [2009] analyzes three trajectory visualizations by comparing how well they can guide users during the interaction in the presence of different spatial trajectory features: Trajectories that form *straight lines* are the baseline case for evaluating the visualizations, because they are the simplest form of a trajectory and can be described with very little visual information. Also, dragging an object along a linear trajectory does not cause any problems with temporal ambiguities or irregularities—such as discontinuities—in the distance field defined by the distance function (cf. 4.2.3 "Distance Measures and Their Influence on Temporal Ambiguities"). *Curved trajectories* can already exhibit discontinuities in the

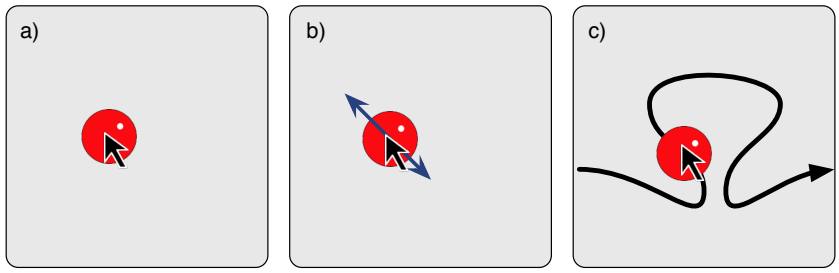

Figure 4.45: Different visualization options for object trajectories. The trajectory information can be visualized in various degrees: invisible (a), only cardinal direction indicator (b), or the full trajectory in a local neighborhood of the current location. Source: [Brockly, 2009]

distance field (cf. Figure 4.35). Therefore, it is more important for the user to drag the object in a precise fashion, and we need to check how well the different visualizations facilitate this. *Angular trajectories* happen when objects collide or ricochet off each other or static parts of the scene (cf. Figure 4.33). Users tend to overshoot their dragging movements toward these points dependent on how the trajectory is shown. *Waves*—trajectories that progress along a general direction but exhibit frequent spatial deviations from that direction—are also handled differently depending on the visual feedback.

For the first version of DRAGON, we did not visualize the trajectories at all.

The first—and probably the most simplistic—trajectory visualization is to not give any visual indications about the motion path of the object of interest at all (Figure 4.45 (a)). This also was the visualization option we chose for the first experimental evaluation of DRAGON, which was the basis for our initial publication on DMVNs [Karrer et al., 2008]. Of course, the possible motion patterns of many real world objects in typical video scenes are physically and culturally constrained (cf. [Norman, 1988]) and thus can be anticipated by the users up to a certain extent. This is also the reason why users are able to drag objects along their (invisible) trajectories with surprisingly high precision: Straight line trajectories are followed easily, waves may even—if they correspond to some characteristic movement of the object of interest—be over-pronounced (Figure 4.46). Dragging along curved and angular trajectories, however, often suffers from some 'corner cutting' and overshooting (Figure

Figure 4.46: Original motion of the object (red) and user dragging path (blue) for non-visualized straight and wave trajectories. Users interpreted the scenes and easily anticipated the motion of the objects even without explicit visualization of the trajectory. In the case of very characteristic movements, these were sometimes overemphasized. Source: [Brockly, 2009]

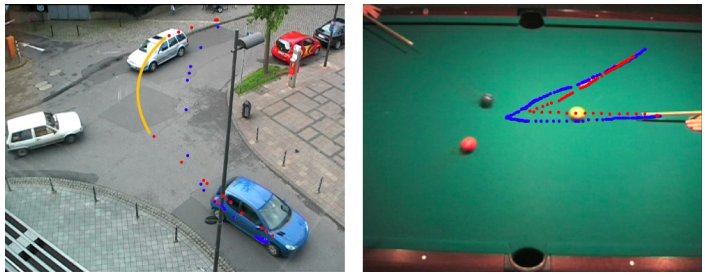

Figure 4.47: Original motion of the object (red) and user dragging path (blue) for non-visualized curve and edge trajectories. In both examples, the motion has been predicted correctly but the dragging path exhibits corner-cutting and overshooting. Source: [Brockly, 2009]

4.47) if the trajectories are not shown. This may cause dragging paths to potentially cross distance field discontinuity boundaries, which can result in confusing temporal skips.

The trajectory can only be visualized in a small local neighborhood.

The second visualization Brockly [2009] looks at is to indicate only the directions of the incoming and outgoing tails of the trajectory at the current point (Figure 4.45 (b)). This is basically a simplified form of the *starburst* manipulator widget that has been proposed by Goldman et al. [2008] as a form of trajectory visualization (Figure 4.48). The *starburst*

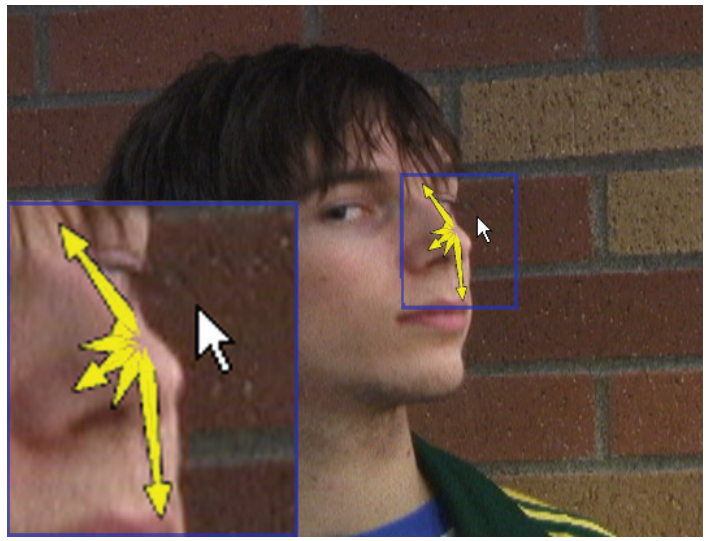

Figure 4.48: The starburst indicator by Goldman [2008]. The polar coordinates of all points on the trajectory are accumulated in a radial histogram and the bins represented graphically by arrows. Source: [Goldman et al., 2008]

manipulator basically is a weighted polar coordinate histogram of the points on the trajectory relative to the current location: The points on the trajectory are weighted according to their distance and binned according to their angle from the current location. Instead of this negative exponentially scaled distance weighting, Brockly's arrow indicator uses a boxcar function on the trajectory neighborhood. The result is that only the cardinal directions in the immediate frame neighborhood are visualized.

Showing only the direction of the trajectory tails works well for trajectories that form straight lines (Figure 4.49). If the motion of the object describes a curve or a more complex shape, users again start to 'cut corners' in their dragging path. At the cusps in angular trajectories or waves, users tend to severely overshoot with the arrow visualization; this, however, usually does not produce any frame skips or cause the navigation to fail (Figure 4.50).

Figure 4.49: Original motion of the object (red) and user dragging path (blue) for straight and curved trajectories with visualized direction. The dragging trails are relatively close to the original trajectories; some corner cutting still occurs. Source: [Brockly, 2009]

Figure 4.50: Original motion of the object (red) and user dragging path (blue) for edge and wave trajectories with visualized direction. Severe overshoots still occur, but the trajectories were followed faster than without any visualization. Source: [Brockly, 2009]

Showing a large neighborhood or the full trajectory is the best option to guide the users' dragging motion.

In contrast to the arrow method, which gives only local information about the shape of the object's motion path, a third method is to show the image space projection of the complete trajectory (Figures 4.51 and 4.52). This method results in the best performance of users following the object's path [Brockly, 2009], the distance between the real trajectory and the dragging path is much smaller than with the other two visualizations. We therefore used this visualization option for all later prototypes of our DRAGON DMVN, rendering the full trajectory of an object as soon as it has been selected by the user.

Figure 4.51: Original motion of the object (red) and user dragging path (blue) for straight and edge trajectories with fully visualized trajectory. The dragging trails are faster and closer to the original trajectories than in the other visualization conditions. Source: [Brockly, 2009]

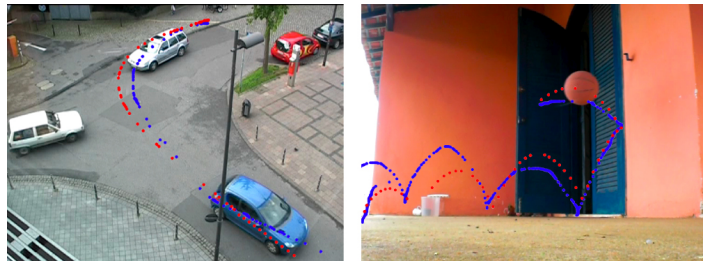

Figure 4.52: Original motion of the object (red) and user dragging path (blue) for curve and wave trajectories with fully visualized trajectory.The dragging trails are faster and closer to the original trajectories than in the other visualization conditions. Source: [Brockly, 2009]

A disadvantage is that the full trajectory may clutter the video frame.

A disadvantage of this solution is that very long and complex trajectories—or trajectories with a large number of *pause loops* (cf. 3 "Object Pauses")—can produce excessive visual clutter. This problem could be overcome by showing only a local neighborhood, either measured temporally or by arc-length, and fading out the rest of the trajectory.

It is possible to augment the visualization to show additional information like object velocity.

Information other than just the object's path, for example, its velocity can also be conveyed by modifying the visualization of the trajectory. Brockly [2009] has experimented with representing object velocity with the trajectories in two ways:

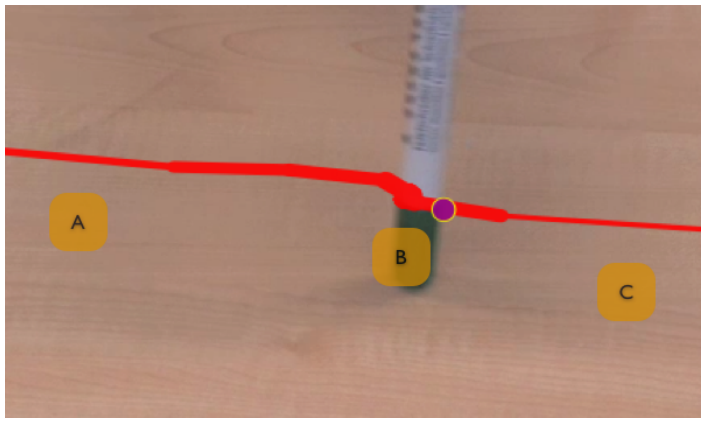

Figure 4.53: Visualization of object velocity through variable line width of the trajectory. Some users associated thin lines with faster motions, some associated thick lines with faster motion. Source: [Brockly, 2009]

- as dotted lines where a dot is placed along the path every n-th frame, thus indicating higher object velocities through sparse placements of dots and lower velocities through denser dotting (Figures 4.46–4.52 use this technique to visualize the velocity of the objects and the user dragging paths), and

- as lines of variable width where the trajectory is drawn as a thin line in places where the object moves fast and as a thick line in places where it moves slowly (Figure 4.53).

He could show that both approaches convey the information of object velocity in a way that is understandable by most, but not all, users by measuring the participants agreement on either of the possible interpretations. In the case of the variable line width method, however, 14% of the participants were undecided about the meaning of the visualization while the dotted-line method always invoked a clear decision for 'faster' or 'slower'. A disadvantage of the latter technique is, though, that dots may be placed so sparsely that the trajectory is not discernible any more or so densely that individual dots cannot be distinguished.

Additional information can also be given through other channels, e.g., audio.

Another technique to convey velocity information to the user is an auralization variant of the dotted-line method. Instead of drawing a series of dots along an object's motion path, we can draw the trajectory normally and play back a short tick sound on every n-th frame. Since the direct manipulation navigation technique warps the regular timeline of the video to 'normalize' time relative to the object of interest and does it in a way that is transparent to the user, this added information channel can help to understand how the scene context around the object behaves temporally. The auralization of object velocity is the technique that is implemented in DRAGON, because it possesses the advantages in terms of interpretability but not the graphical disadvantages of the dotted-line approach.

Interactive Scoping and Trajectory Filtering

The scoping problem is also present during the interaction.

In the discussion about how the semantic mapping can technically be established via object recognition and tracking algorithms, we have already identified the difficulty of identifying and scoping the semantic objects of interest (cf. 4.2.2 "Determine the Semantic Mapping"). The same problem, of course, also poses itself on the interaction and user interface side of a DMVN. Most DMVNs [Karrer et al., 2008; Dragicevic et al., 2008; Kimber et al., 2007] let the user specify which object in the video scene should become the reference object for the semantic mapping by a single mouse click on the object in the scene: they use the common selection techniques long-known from desktop computing for the semantic selection process.

Selecting objects via a single click in an unstructured medium leaves many degrees of freedom open.

This technique works well in standard WIMP interfaces where most of the objects or elements—such as Icons, Menus, Windows—have a simple and well-known structure and are separate logical entities to the system; every pixel on the screen of a window system belongs to a certain window and can be mapped to the according input element or widget [Scheifler and Gettys, 1986]. An indication of a single point in 2D screen space, usually conveyed through a click, is thus enough to specify an object.

For most types of visual media, such as drawings, photos, or videos, the situation is much more complicated; these

media generally do not possess an explicit syntactic representation of semantically meaningful content elements. A digital photo, for example, usually is only a collection of pixels—perhaps with some associated metadata describing the context in which the photo was taken—and does not contain any information about the structure of the scene it depicts.

Also, some software packages designed for the creation of visual media, for example vector drawing programs like Adobe Illustrator, try to preserve the structural information inherent in the creation process of these media. In the complete absence of structural information like we usually encounter in photos or videos, however, selection and representation of objects becomes a difficult process. Pointing at a certain location in an image or video is not any longer enough to associate this action with an object depicted in the medium.

Single click selection relies on the spatial and temporal coherence of objects to do the scoping.

The reason why DMVN systems like DRAGON [Karrer et al., 2008] or *DimP* [Dragicevic et al., 2008] still work for many cases with single-click selections is that they exploit the implicit spatial and temporal coherence of objects in videos. As explained above in 4.2.2 "Determine the Semantic Mapping", the respective flow tracking algorithms of these systems just track the selected single pixel through the flow fields, using a dead-reckoning approach. Only the fact that the pixel cloud that constitutes the real object of interest exhibits relatively little changes in its spatial configuration over time—most objects tend to have similar shape and location between adjacent frames in a video—suggests that the object itself is being tracked and manipulated.

For some classes of objects or some types of motion, the implicit scoping is not the intended one.

It is easy, however, to imagine numerous scenarios where this 'illusion' breaks down: Objects that change their shape or color significantly over time are likely to have erratic or interrupted trajectories. Partial or full occlusions of objects will regularly cause the trajectory to end abruptly or, even worse, to stay with the occluding object, which is often a part of the stationary scene background. But even in the absence of rapid changes or occlusions, a single point selection is sometimes not enough to specify the user's intention; aggregate objects or such that have differing motion fields and optical flow fields can exhibit unexpected behavior when selected in DMVNs.

Figure 4.54: Different trajectories for different selection scopes of a climber. The trajectory for the torso differs considerably from the trajectory for the hand or the knee.

Aggregate objects may violate users' expectations about the resulting trajectories.

The rock climber in Figure 4.54 is an example for an aggregate object. While a click on the torso yields a trajectory that progresses upwards the rock face in what is more or less a straight line, a click on an arm or leg results in a very complicated and maybe overly detailed trajectory describing the movement of the limb. Without a technical understanding of the underlying algorithms, users are likely to be surprised by the result of their one-point selection, which violates the *principle of least astonishment* [James, 1986], a fundamental rule in the design of user interfaces.

A similar problem is evident in the examples of a rolling soccer ball (Figure 4.55) or a barbershop pole (4.18): each single pixel describes an (apparent) movement that is very different from the movement of the whole object.

A more explicit scope control mechanism is needed.

These examples show that, in the absence of additional cues, single-point selection can be insufficient for DMVNs to capture the user's intention. In the case of aggregate objects, we need to give users a way to specify which part or parts of an object they want to interact with. These problems are also related to the difficulties when navigating very complex object trajectories (cf. 4.2.2 "Determine the Semantic Mapping"): often, users are not interested in following

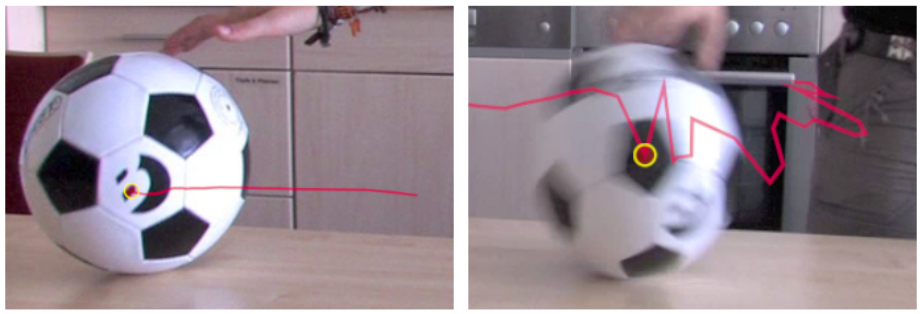

Figure 4.55: Differences in the trajectories of a soccer ball and one of its black spots. While the trajectory of the ball in full represents what most people would intuitively describe as the motion of the ball, each of the black spots on the ball follows a rather chaotic trajectory. Source: [Novosad, 2012]

all the nooks and high-frequency details of such trajectories, which may only reflect the detail motion of an implicitly selected part of the actual object of interest, but want a way to quickly and easily displace the object across larger distances on the screen. This has been partially addressed by designing the distance measures of the DMVN algorithms in a way that the parts of the trajectories with high spatial frequencies can be skipped or 'slipped through' (cf. 4.2.3 "Distance Measures and Their Influence on Temporal Ambiguities"), but the users have no explicit control over this behavior and no way to express their intention in this regard.

Considering these problems, we can propose three possible solutions:

1. Let the user gives cues about the structure of the object.
2. Find the structure of the object automatically.
3. Ignore the structure but emulate interactive object scoping by giving the user implicit or explicit control over the shape of the trajectory.

We have already discussed the first two of these solutions in the second step of our four-step-approach to generating semantic video navigation interfaces 4.2.2 "Determine the Semantic Mapping": Allowing the user to give cues

about the structure of the object of interest is most easily accomplished by offering area selection mechanisms such as paint-over selection [Goldman et al., 2008] or bounding box selection [Goldman et al., 2008; Wittenhagen, 2008; Novosad, 2012]. Automatically determining the structure of the scene and of the object is still either an active research topic of computer vision or still in its infancy when measuring the structure directly with appropriate hardware [Lewandowski, 2011]. This approach would make a single-point selection of the object of interest feasible again for some cases, and it would allow to interactively refine or redefine the selection while respecting object boundaries or offering assistance to select parts of objects. This class of selection techniques that are guided by algorithms that know the spatial structure of the scene thus remains a prospective area for future work.

Even for single-point selections, we can offer ways to control the selection scope during the interaction.

For the third solution, we have studied two different approaches that emulate interactive control over the semantic granularity of single-point selections a user has made during the interaction with our DRAGON DMVN: one adjusting the granularity implicitly and one where the granularity is controlled explicitly by the user. The underlying idea of both is to remain with the convenience of a single-point selection of the object of interest but to allow some adjustments to the object scope and dynamically adjusting or generating the appropriate trajectories:

We can use pressure as an additional input channel to control the selection scope.

- *Using Pressure for Explicit Scope Control*
  To allow users to actively and explicitly determine the scope of the object selection after only indicating a single point—and possibly during the dragging navigation—we need an additional input channel. The input parameter space of a regular computer mouse is mostly reserved for the dragging gesture, leaving only possible extra mouse buttons for other uses. Since we aim for an additional linear control to be able to control the scope in a continuous way, we employ a pressure sensitive input device, for example a digital pen and tablet. Using pen-based input for video navigation has been successfully demonstrated in earlier work, for example in [Ramos and Balakrishnan, 2003]. In our prototype [Novosad, 2012], the initial single-point selection represents the finest scope

of a single pixel trajectory like it is generated by our dense optical flow approach. Exerting a higher pressure while dragging the object gradually selects larger scopes until the selection represents the whole object.

- *Using Dragging Velocity for Implicit Scope Control*
  As an alternative approach that does not require the explicit extra input, we have experimented with inferring the appropriate scope from the users' dragging behavior. From existing basic perception and motion models, like CMN [Card et al., 1986] or Fitts's Law [Fitts, 1954], we know that most simple, targeted motions start with a large ballistic movement, which covers most of the distance, followed by multiple corrective 'homing' movements that decrease in size and increase in frequency. If we assume that dragging an object—or parts of an object—for direct manipulation video navigation follows a similar mechanism and exhibits similar patterns (cf. [Nett, 2012]), we can increase the scope of the selection during the ballistic phase and decrease for the fine tuning of the object position. Implicitly controlling the selection scope with the velocity of the dragging gesture, similar to the acceleration interface of the *Alphaslider* [Ahlberg and Shneiderman, 1994], is close enough in its behavior to be employed as a reasonable first approach.

We can try to derive the scope from the velocity of the dragging motion.

Implementing this kind of dynamic change in the scope of the object selection, of course, is difficult—especially when we only have a single-point selection to begin with. A possible solution would be to generate a set of trajectories around the initial single-point selection and filter this set through hierarchical clustering in affine motion space, similar to [Goldman et al., 2008] and [Wittenhagen, 2008]. The same set could also be generated by pre-calculating a hierarchical set of flow fields for the full video volume pyramid where each frame is a scale space representation of the projected scene, usually in the form of an image pyramid (Figure 4.56). Different object scopes could then be realized by taking different layers in the hierarchy, the trajectories in the current layer being averaged to form a single trajectory representing the object's motion [Novosad, 2012]. While this solution works, it is also very costly in terms of computational power.

For the implementation, we need the trajectories on different scopes.

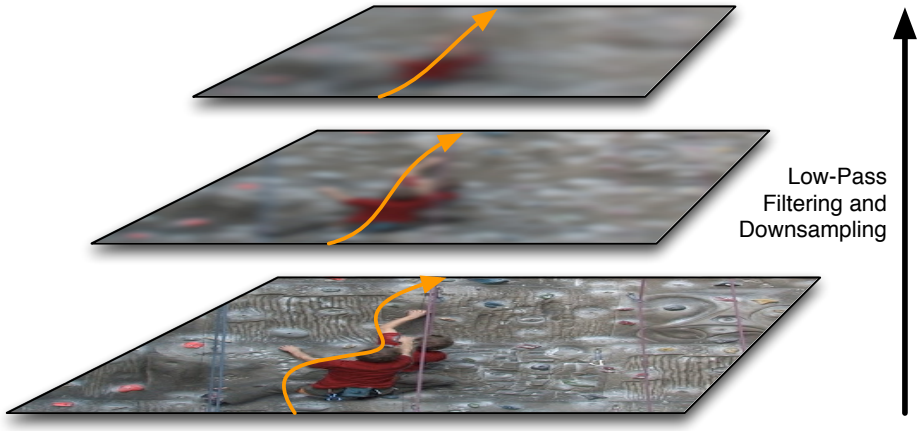

Figure 4.56: A hierarchical set of trajectories that approximate the object's movements in different scopes can be created using an image pyramid for each frame of the video.

It is more efficient to approximate the coarse trajectories by filtering.

We therefore chose to implement an alternative solution to the problem by only generating and then modifying the—possibly imprecise and wrongly scoped—trajectory resulting from the first single-point selection. For this purpose, we generate the trajectory at the maximum possible detail so that it contains all variations in the high spatial frequency bands, ideally by employing our dense optical flow technique 4.2.2 "Generating Object Trajectories in a Structure-agnostic Way", which is—together with particle tracking—the most precise pixel tracking available across all existing DMVNs. After this, we emulate more coarsely scoped object selections by dynamically filtering the trajectory by applying an adjustable spatial low-pass filter mechanism.

We rely on the assumption that object size and motion frequency are anti-correlated.

This idea is based on the assumption that the semantic scope of the object of interest—or the part thereof that is important for the navigation—negatively correlates in its size with the spatial frequency of the resulting trajectory. In the climbing video shown above, for example, the trajectory of a limb possesses a high degree of spatial detail; the trajectory of the torso is, in contrast, much more featureless. But since the limb is a part of the climber, the general low-frequency progression of its trajectory is very similar to that of the torso—in fact, a low-pass filtered version of the limb trajectory is very similar to that of the torso (cf. Figure 4.57).

Figure 4.57: Comparison between a low-pass filtered trajectory of the arm (right) and a trajectory of the torso (left). Both trajectories are very similar; the unfiltered trajectory of the arm is shown in Figure 4.23.

Similar assumptions can be made for objects with internal motion, like a rolling soccer ball. From these observations, we can create an interface that 'cheats' by seemingly offering a post-hoc selection of object scope while keeping the computational cost of its implementation at a minimum. The precise details of the implementation of both methods—explicit, pressure-based scope control and implicit, dragging-velocity-based scope adjustment—can be found in Alisa Novosad's Master's Thesis [2012].

We have analyzed both techniques and compared them to a regular single-point selection DMVN interface in a controlled experiment. Across four videos, each containing object trajectories of a different complexity (Figure 4.58), we measured the time it took our test users to perform a simple navigation task in each of the three conditions.

Users could complete navigation tasks with complex trajectories faster with the scoping control techniques than without.

While for the first two videos, which contained only relatively simple trajectories, we could find no significant differences in the users' task completion times, we found that for the more complex trajectories, users were significantly faster using either of the dynamically scoped interaction techniques than with the conventional DMVN interaction [Novosad, 2012]. Regarding the differences between the explicit, pressure-based and the implicit, velocity-based techniques, the latter was both quantitatively faster and qualitatively preferred by our test users [Novosad, 2012].

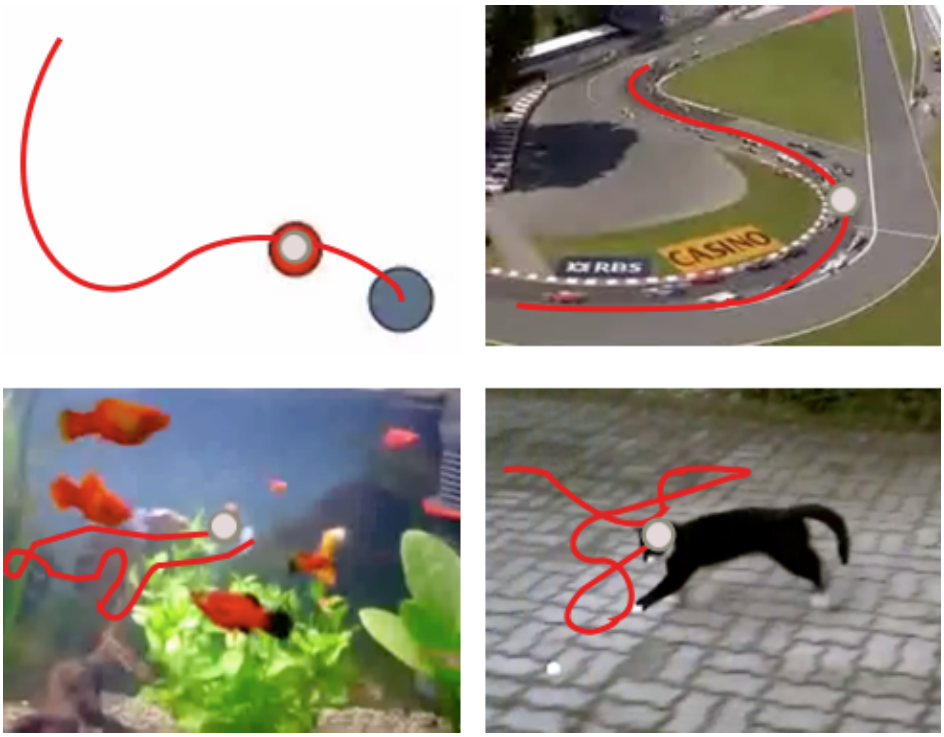

Figure 4.58: The four different test videos featuring four trajectories of varying complexity. Source: [Novosad, 2012]

4.2.4 Evaluate the Interface

All of our findings above have been verified in controlled experiments and user tests.

The basic conceptual ideas of semantic direct manipulation video navigation described in this chapter as well as all of the proposed extensions to the interaction technique—such as trajectory visualization alternatives, methods to dynamically adapt the direct manipulation paradigm to the changing domains of the semantic structure, approaches to user-controlled semantic granularity, and others—and algorithmic advancements in establishing the semantic mapping have, of course, all been thoroughly analyzed and evaluated in a number of user studies and controlled experiments. These evaluations constitute the fourth step in our generative model for semantic navigation interfaces 2.2.4 "Generating New Interfaces Using the Combined Model" and are necessary to ascertain that the design goal of creating a better way to navigate digital media has been met. The ex-

act descriptions of the apparatus and methodology of each of these tests, however, would not add to the discussion at hand; we will therefore only recap some of the most important results together with our interpretations and conclusions in the context of the model proposed in the theory chapter (cf. 2.2.3 "Combined Model" (p. 50)). For further details on these experiments, we refer the reader to the individual publications referenced in this chapter.

The main advantages of DMVNs are the direct representation and manipulation of the objects of interest and the simple mapping between effector space and target space.

The core concept of the DMVN technique is to replace the syntactic navigation that is offered by conventional timeline-based user interfaces by a navigation method that utilizes a spatial direct manipulation interface to interact with the semantic structure of the video scene. As explained in the first part of this chapter, this concept was envisioned to have two main benefits: First, the users of a DMVN system can interact with entities that correspond to their conceptual model of the task; the manipulators and control elements of such an interface are exactly the objects visible in the video scene and thus the objects of interest in many navigation tasks. Users can therefore concentrate on adjusting object locations and spatial constellations and do not have to perform the conversion of these concepts to the syntactic temporal structure of the video themselves. Second, the mapping between manipulating these control elements and the effect of this manipulation in the syntactic structure of the medium is kept as close to the identity mapping as possible—it is, in fact, the identity for most cases but for pauses and other singularity points in the semantic mapping (cf. 4.2.3 "Object Pauses"). This is a stark contrast to the syntactic timeline navigation where the mapping between manipulating the temporal position and its effect on the constellation of objects depends on many factors—including the geometric shape of the slider, length of the video, and content of the video—and is hence generally unpredictable (cf. Figure 4.10); users can infer knowledge about this mapping only through rote memorization of the contents of the video or, in a limited scope, through semantic constraints inherent in the content.

We can analyze the interaction with DMVNs in terms of the seven stages of action.

One way to conceptually analyze these beneficial properties of DMVNs is to check how they can affect the different steps—and gulfs—in Norman's *seven stages of action* model [Norman, 1988]. In the case of syntactic navigation using the timeline slider, we have already discussed in sec-

tion 4.1.2 "Analyzing the Semantic Distance of Slider-based Navigation in Videos" that the interaction can be problematic, and several gulfs of execution and evaluation can arise:

With the timeline slider, several gulfs of execution are evident.

First, there is a gulf between the *goal* and the *intention* stages, as the intention is difficult to formulate using the available concepts in the interface. These neither represent the objects of interest from the user's conceptual model, in terms of which the task or goal are formulated, nor do they offer meaningful operations on such objects. Second, there is a gulf between the *intention* and *action sequence* stages, because a precise action sequence cannot be formulated without knowing the effect of each action. This, however, is precisely the point where knowledge of the mapping between the syntactic and the semantic structures would be required. Third, depending on the ratio between the length of the slider and the length of the video, execution of any action sequence might be difficult or even impossible because the size of the pointing targets may shrink to subpixel scales. Changing the geometry of the slider or loading a video with a different length additionally changes the control gain, which is a problem of its own [Jenkins and Connor, 1949].

On the evaluation side, the direction and distance to the goal are not visible.

On the evaluation side of the seven stages, perception and interpretation remain conceptually simple, as they are directly reflected through the video image. Also, the comparison with the goal is straightforward. One important aspect, however, is that the outcome of this comparison can only be binary in the case of syntactic navigation; either the goal has been reached or not. Very little or no information can be gained regarding the remaining distance or direction towards the goal—the following seven-stages-cycle thus has to be traversed in exactly the same way as the current one.

Users must resort to alternative interaction strategies to reach their goals.

The result of this latter insight in particular is that the only viable strategy for syntactic navigation is a linear search through the syntactic structure of the video, always checking if the goal has been reached. This also means that the full length of the interaction is performed in a closed-loop fashion; the characteristic two-phase approach to interactions where the target is visible and directly approachable—a 'ballistic' open-loop phase to get close to the target followed by a closed-loop 'homing' phase [Woodworth, 1899; Buxton et al.]—is not applicable.

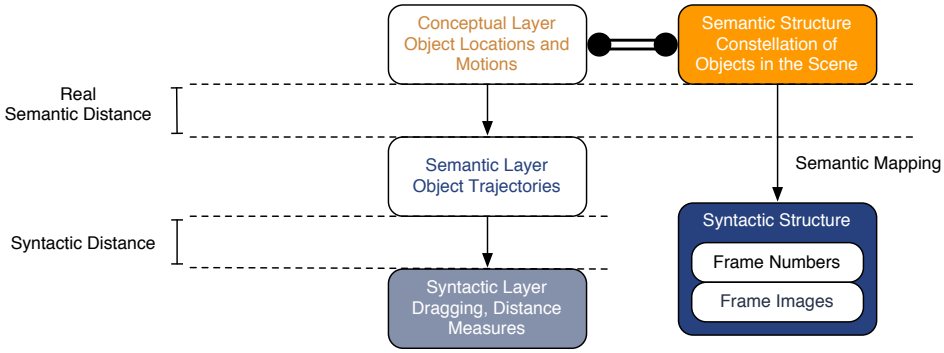

Figure 4.59: Combined navigation interface model for DRAGON and a semantic navigation task based on object constellations.

DMVN interaction eliminates the gulfs that are present when navigating via the timeline.

With our semantic video navigation model (Figure 4.59), none of the aforementioned gulfs is present. This, together with the fact that both the goal is visible and the mapping between the control space and the semantic structure is fixed to be the identity mapping, should allow the task to be performed in a classic steering-law [Accot and Zhai, 1997] fashion. Direct manipulation video navigation essentially short-routes the seven stages by formulating one intention and action sequence for the ballistic phase and one for the homing phase. Only the latter is then run in closed-loop mode but with a much easier, gulf-less cycle.

User tests support these theoretical considerations.

These considerations strongly suggest that video navigation tasks can be accomplished much faster with a DMVN as we have described it than with a traditional syntactic timeline interface and that the cognitive load for the user should be reduced at the same time. We, as well as other groups, have conducted several studies to check if and how these theoretical advantages are reflected measurably when testing semantic video navigation with users [Karrer et al., 2008; Dragicevic et al., 2008; Brockly, 2009]. These studies have reported significant quantitative and qualitative improvements to task performance, user satisfaction, and system usability:

Dragicevic et al. [2008] compared task performance times and navigation error between their *DimP* DMVN and a regular timeline slider interface. They used two synthetic video scenes containing stylized objects moving independently

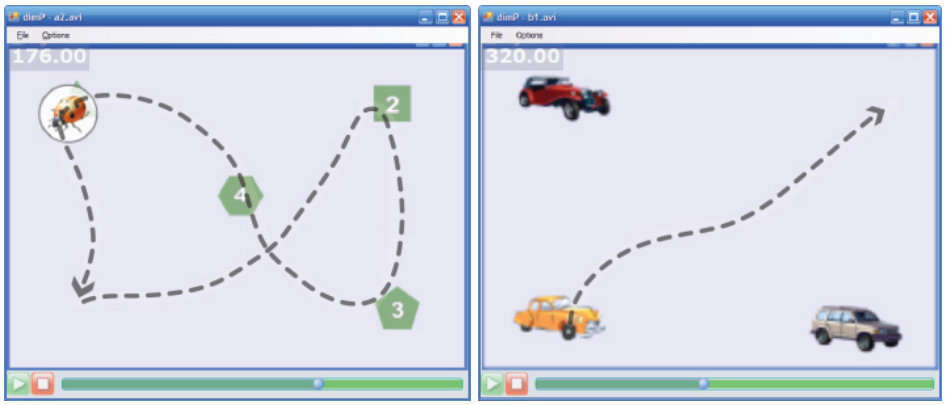

Figure 4.60: Experiment setup in [Dragicevic et al., 2008]. The users were given these two synthetic videos for the navigation tasks. Figure taken from [Dragicevic et al., 2008].

over a solid colored background (Figure 4.60); their tasks included navigating to the points in time where certain objects occupied a designated position or where certain objects changed their behavior (e.g., started moving).

They measured the time between the first mouse down event of the interaction and the last mouse up event and the navigation error as the frame offset in a video between the intended goal frame and the one navigated to by a participant. Their results support our argument for direct manipulation semantic navigation being easier for the user and much more efficient: Participants could complete the tasks at least 250% faster with *DimP* than with the timeline slider with high statistical significance ($p < 0.0001$). Also, participants felt that the DMVN interaction was both easier and faster and that it led to higher user satisfaction. This was also reflected quantitatively in the fact that the average navigation error was less in the DMVN condition.

We performed a similar experiment [Karrer et al., 2008] comparing navigation using DRAGON with a regular timeline slider interface. In contrast to the study by Dragicevic et al., we used real video footage (Figure 4.61) and a wider range of tasks including navigating to the point in the video where: a certain object was located at a designated position, two certain objects were in a designated spatial constellation, a certain object interacted with a static part of the

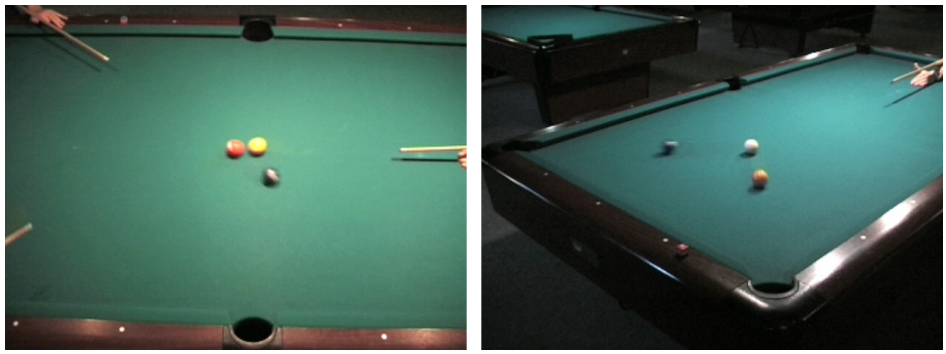

Figure 4.61: Two of the five video scenes from the DRAGON user test in [Karrer et al., 2008].

scene, and a certain object changes its behavior (e.g., stops moving). The task completion time was measured in the same way.

Our results are consistent with the predictions we have made above regarding the advantages of semantic navigation and direct manipulation under an identity mapping; users could complete all tasks significantly faster ($p < 0.01$) using DRAGON than using the conventional timeline slider interface. The speedup factors in task completion times ranged between 24% and 75% depending on the task and video.[6] Our explanation for the difference in task efficiency increase between the DRAGON and *DimP* studies is that some of our tasks were more difficult and that the reduced visual clutter and 'perfect' trajectories of synthetic videos may have biased the results towards unrealistically short task completion times.

Users found efficient DMVN-specific strategies for certain events and object locations.

During the experiment, we could also observe a number of emerging navigation strategy patterns that were unique to the DMVN condition. The rubber-banding effect of the DRAGON distance measure (cf. 4.2.3 "DVMN Interfaces and Interaction Techniques" (p. 132)), for example, was used to exploit the constraints for the object's location offered by some spatial features of the trajectory: To quickly navigate

[6]Note that in the paper [Karrer et al., 2008], we reported the ratio of the task completion time delta and the slower task completion time. The numbers in this text denote the ratio between the task completion times in the two conditions. This was done to be compatible with how the results are reported in [Dragicevic et al., 2008].

to a position in the video where two objects collide, users found that they could just drag one object towards the other and carelessly overshoot. The spatial constraint imposed on the object's trajectory by the two objects changing direction after the collision would then keep the dragged object in place precisely at the collision point. Users also quickly discovered successful strategies for navigating to locations where objects start or stop moving by utilizing the temporal component in *DRAGON's* distance function. Some of these strategies, however, would not work with the purely spatial distance measures employed in *DimP* [Dragicevic et al., 2008] or Goldman's DMVN [Goldman et al., 2008].

Video navigation with DMVNs was preferred over timeline navigation and was considered quicker and easier to use.

We also collected qualitative feedback in a post-session questionnaire, finding that the majority of our users preferred DRAGON as the technique for navigation in video scenes. Likewise, DRAGON was perceived to be both quicker and easier to use than the standard timeline sliders for the tasks given in the user test (Figure 4.62). There was a strong agreement among the users that a DMVN would also be the preferred system for video navigation in the context of video editing and that using DRAGON in the study felt natural and always behaved like they expected. The questionnaire, however, also revealed some of the limitations of semantic video navigation: Simple video consumption—like watching a movie from the beginning to the end—and similar tasks do not benefit from the semantic navigation capabilities. Also, the systems discussed so far are only useful for fast and precise navigation inside a single video scene; at cuts or scene changes, as they happen frequently in movies, the trajectories end, thus prohibiting semantic navigation across these deliberate breaks in the medium. This last limitation, however, is merely technical in nature—some existing systems, for example *Trailblazing* [Kimber et al., 2007], already employ an object recognition database to continue the trajectories across scene boundaries and different cameras.

4.3 Summary and Conclusion

In this chapter, we have demonstrated how we can use our interaction model (cf. 2.2.3 "Combined Model") to design

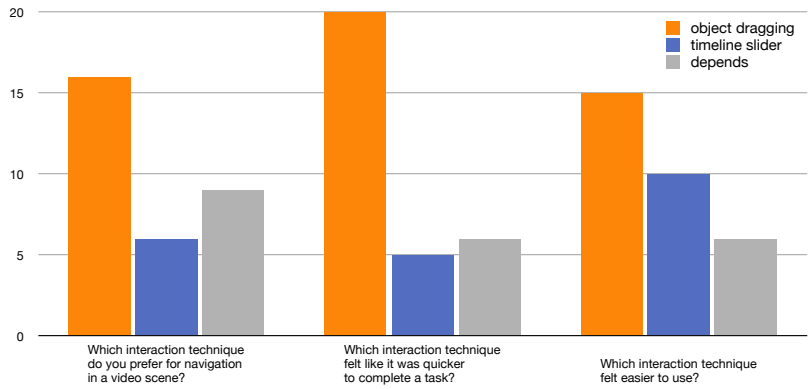

Figure 4.62: Questionnaire evaluation for the DRAGON experiment from [Karrer et al., 2008]. Most of the 31 test users perceived DRAGON to be quicker and easier to use for video navigation than the timeline slider.

a semantic navigation interface for video. After reviewing and analyzing the status-quo of video navigation interfaces, and the timeline slider in particular, in the framework of our model of navigation in digital media, we described the concept for DRAGON, our semantic direct manipulation video navigation system. For this, we followed our proposed four-step approach:

We designed our semantic video navigation interface according to our proposed interaction model.

We first identified the conceptual model of the semantics for common video navigation tasks, resulting in the definition of the set of spatio-temporal subspaces representing the locations over time of all objects in the video as a suitable semantic structure of the medium. Then, we analyzed and discussed a number of different possibilities to access this semantic structure through the concept of motion trajectories of potential objects of interest in the video scene. The result is a semantic mapping that is defined by the content of the medium itself, thus allowing users to directly interact with a representation of the objects of interest. The third step, our focus in this chapter, was concerned with the implications of the interaction and interface design. It introduced a number of extensions to the basic DMVN idea, which allow to mitigate or negate limitations like temporal ambiguities, tracking problems, or selection scoping. We found that the usability problems that arise at temporal ambiguities indicate domain shifts in the semantic structure,

so we proposed ways to reflect these shifts in the semantic mapping. This also led to a more open interpretation of the direct manipulation idea: instead of the traditional restriction to a single domain, we found that we could include changes across different control domains in a consistent way. To address the scoping problems stemming from the aggregate structure of objects and their motion patterns, we demonstrated approaches to dynamically select a semantic mapping resulting in a continuous spectrum of control granularities. In the final step, we presented results of the experiments conducted to evaluate the benefits of this approach to semantic video navigation.

4.3.1 Non-Navigation Applications of DMVN

This application of our interaction model to the problem of video navigation has proven to be successful for a wide class of use cases. Apart from the positive results of user studies with various DMVN prototypes, the concept of video navigation by direct object manipulation has also been shown to be applicable to a number of related problems:

Video content can be edited on the basis of object trajectories.

Shah and Narayanan [Shah and Narayanan, 2011], for example, have introduced a powerful video manipulation tool based on modifying the same semantic structure. They extend the object trajectories by defining them as a function from the temporal domain not only into the domain of locations but into the domain of bounding rectangles, turning object trajectories into object 'tubes'. These tubes can then be removed from the video volume, shifted along the temporal axis, or linearly transformed along the temporal axis. Where such an operation leaves gaps in the volume, these are filled by means of frame-wise background synthesis. Manipulating the semantic structure of a video in this manner allows users to easily alter the contents on a very high semantic level. Removing an object from a scene or reversing the order in which two objects pass through the scene while not affecting the behavior of a third object that is visible at the same time—both editing operations that are extremely difficult to perform with only access to the semantic structure—can thus be applied in an easy and straightforward way.

Direct manipulation of object locations along their motion trajectories can be used to interactively re-time computer animations.

Another example for the application of the concept is to generalize the interactive semantic navigation through a video to higher dimensional visual data that is arranged along a timeline, such as 3D computer animation. Recording the temporal progression of navigation runs through spatially pre-defined animation paths offers an intuitive way to define the animation timing by example—an operation that normally has to be performed by means of filling out detailed timing data in numerical form for each key frame, followed by automatic or manual 'tweening'. We have built DRAGIMATION [Walther-Franks et al., 2012] as a demonstration system for this kind of interaction and could show that it can be used easily even by animation novices and that it significantly outperforms timeline-based performance animation in terms of measured precision as well as user assessment regarding ease of use, mental load, and overall preference. DRAGIMATION thus is on par precision-wise with the existing *sketching* technique [Terra and Metoyer, 2007] for performance animation but was likewise rated as being significantly better regarding ease of use, mental load, and user preference.

DMVN has already been used for commercial purposes.

Commercial applications that use semantic video navigation by direct manipulation can already be found. In one of their advertising campaigns, the fashion company Wrangler™ embedded short DMVN-enabled video clips on their online shop website. Users could pull zippers of the advertised garments and make the (male) models interactively dress and undress. While there was ever only one active dragging handle in each video that represented a predefined object of interest and while the implementation was done in Flash™ and ActionScript™ with hard coded trajectories for each clip, the rest of the semantic navigation concept was essentially the same as in our DRAGON DMVN. Reportedly, the technology is planned to be deployed on Wrangler's mobile iOS App as well [Novosad, 2012]; a short video of their 2010 prototype is available on YouTube[7].

[7] http://www.youtube.com/watch?v=2zuOdJQiljw

4.3.2 Alternative Choices for the Conceptual Model

One can also generate semantic video navigation interfaces based on a different conceptual model.

For other video navigation tasks that do not fit the conceptual model that we have identified in the first step and used for our DRAGON system, we can use the interaction model to design appropriate interfaces by starting from a different semantic structure that better reflects the nature of these tasks. As a proof of concept, we have done this and created an alternative semantic video navigation system that caters to a different conceptual model.

Kathrein created an event-based semantic video navigation system.

The idea was to offer a higher semantic layer than DRAGON, allowing to directly access a structure consisting of *events of interest* instead of the spatio-temporal structure induced by the motion of objects of interest in the video. Thus, the expressiveness of the supported navigation goals would expand from a first-order concept, i.e. concerned with the behavior of objects, to a second-order concept, i.e. concerning the behavior of sets of objects. The system that realizes this idea was designed and implemented by Anne Kathrein for her Bachelor's Thesis [2011] under the guidance of the author. From a number of interviews with behavioral researchers and video ethnographers, she learned that for their tasks, it was typically very important to quickly navigate to a situation that could be described in very concise terms before precisely navigating inside that video scene. They were missing a macro-level navigation tool to complement DRAGON, which in comparison works on a micro-level scale with higher temporal locality, in order to support their two-tiered conceptual model of video analysis: first, find the set of interesting situations; then, explore each of these situations interactively.

For the semantic structure, we created a hierarchical system of events.

Taking a large number of example navigation targets from the interviews and from related literature on several areas of video analysis (surveillance [Hu et al., 2004; Saxena et al., 2008; Medioni et al., 2001; Lv et al., 2006], market research [Babic, 2010], sports analysis [Tovinkere and Qian, 2001; Ekin et al., 2003]) and clustering them, Kathrein created a hierarchy of spatial behaviors, in which each of the examples can be described. From there, she defined a semantic macro-structure of the medium as a language of possible events where the micro-structure of *DRAGON's* object trajectories

serves, together with a small number of spatial primitives, such as locations and areas, as the alphabet. Her language consists of four main classes (from [Kathrein, 2011]):

1. *Area Dependencies* denotes the set of events where the relative location of one or multiple designated objects and a static area defines the navigation task. Typical patterns of this class include *object crosses an area* or *object is far away from area*.

2. *Objects Act* is the second class of events describing object trajectory patterns that do not depend on other objects or spatial primitives but can be described by certain characteristics. Examples are *object appears*, *object disappears*, and *object moves in a circle*. The first two are widely recognized behaviors in object tracking software packages (e.g., [Girgensohn et al., 2007] or [Favalli et al., 2000]), the latter plays an important role in the behavioral analysis of animals [Zipser; Yanagisawa, 1982]).

3. *Objects Interact* is the class of events representing binary and ternary operations on designated objects. In contrast to the other classes, these events have no temporal invariant—all participating objects must be collocated in space and time to fit any of the event patterns. The class is further structured into two subclasses, *distance patterns* and *correspondence patterns*: The former contains both thresholded distance functions—for example *small distance*, *large distance*, *objects meet*, *one object meets others*, and *objects are visible at the same time*—and temporal sequences of distance functions—for example *distance increases* and *distance increases after objects meeting*. The latter describes characteristic multi-object motion patterns like *objects move in parallel* or *object moves from one object to another*. Containing events that are defined on a high semantic level, this class can describe complex behavioral patterns of interest like the left-luggage scenario [Lv et al., 2006] or ball-passing in soccer [Tovinkere and Qian, 2001].

4. *Direction and Velocity* is a class of similarly high-level events, describing situations that are defined by a single object deviating some larger movement pattern;

object moved in a different direction than other objects, *object moves at a different velocity than other objects*, and *object moves at a different velocity than its own average velocity* all describe typical situations that are important in a security or surveillance context [Siebel and Maybank, 2004; Saxena et al., 2008].

A detailed description together with implementation details of the different algorithms that detect these events in a given DRAGON trajectory set can be found in Kathrein's Bachelor's Thesis paper [2011].

The interface for navigating this discrete semantic structure is not direct-manipulation-based but follows the conversation metaphor.

The interface for this event-based navigation is an extension to the DRAGON interface described earlier in this chapter (cf. 4.2.3 "Designing the User Interface for Semantic Video Navigation" (p. 132)) and adds an augmented timeline where the detected events are represented visually in a way similar to the interface of ChronoViz [Fouse et al., 2011], and a side pane where the navigation target can be formulated using a simple query 'language'. A query in this language is formed by combination of graphical function selectors and interactive object selection for the arguments (Figures 4.63 and 4.64).

For navigation tasks concerning events, the system outperformed the timeline slider significantly.

Kathrein conducted a user study to compare task completion times for two complex first-order and one second-order navigation tasks between her system and standard timeline slider navigation. Consistent with our other studies on semantic video navigation (cf. 4.2.4 "Evaluate the Interface"), she found that, on average, participants were able to complete all three navigation tasks significantly faster ($p < 0.01$) using her event-based semantic navigation interface. The average resulting speedup factors ranged between 150% and 160%.

4.3.3 Closing Remarks

The successful application of our proposed semantic navigation model for digital media on two different conceptual models for video navigation supports our argument for the feasibility of creating such content defined semantic navigation interfaces and demonstrates the positive impact of

Figure 4.63: User interface for semantic video navigation using an event-based conceptual model. The type of event is specified from a hierarchical list; the 'arguments'—objects, locations, or areas in the video scene—are directly specified in the video frame. Source: [Kathrein, 2011]

such interfaces on the users' ability to navigate easily and efficiently through a medium. It also underlines the generative power of the model when following the four-step approach to designing these systems.

In these first two system chapters, we have discussed our model in the context of audio and video, which are both time-based media. Also, both cases mainly required a semantic mapping between structures of the same dimensionality: For PERSONAL ORCHESTRA, we established a mapping between the original timeline and a semantically reparametrized version of time. For DRAGON, each trajectory mapping connected the timeline with a one-dimensional subspace of the video volume, which was projected into a two-dimensional plane.

The event-based navigation that was built on top of DRAGON (cf. 4.3.2 "Alternative Choices for the Conceptual Model") was already a first step into semantic structures of

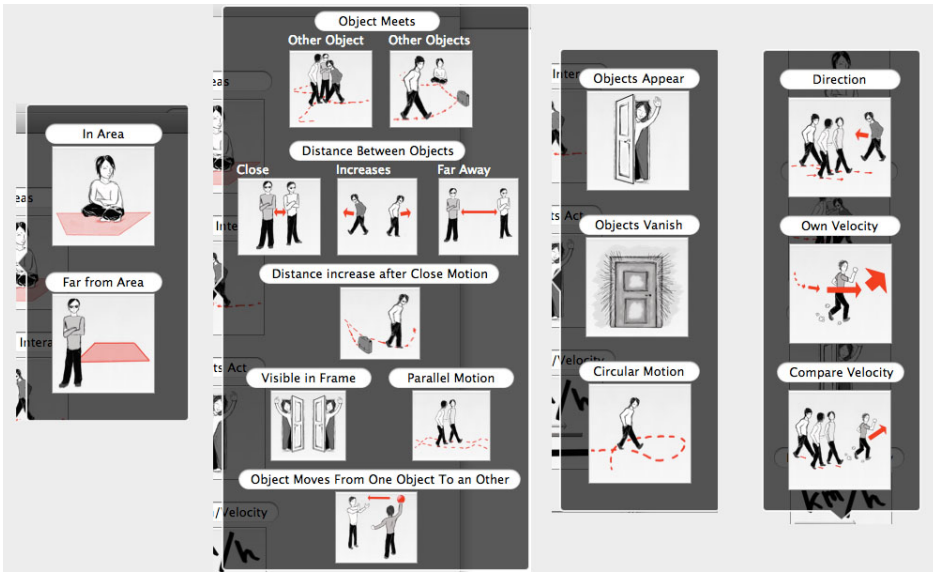

Figure 4.64: Overview over the different event types supported by Kathrein's prototype. Source: [Kathrein, 2011]

higher dimensions. In the next two chapters, we will move away from time-based media entirely and investigate how we can offer higher dimensional semantic navigation spaces for other digital media. In particular, we discuss *Presentation Visuals*, which are defined over very limited syntactic structures where the common syntactic navigation interfaces are not expressive enough in chapter 5 "Hybrid Media: Presentation Visuals"; and we investigate the medium of *Source Code*, which is defined over higher dimensional and fragmented syntactic structures where syntactic navigation is too unconstrained to be helpful, in chapter 6 "Non-time-based Media: Source Code".

Chapter 5

Hybrid Media: Presentation Visuals

Presentation visuals are a hybrid type of media insofar as they—in the form they are usually employed—seem to live somewhere between time-based and non-time-based media. The reason for this is, of course, that the current practice of supporting talks and presentations with slide decks imposes a temporal constraint on the visuals: only one slide can be shown at a time. While this limitation is widely accepted during the actual presentation, there is no obvious need that all navigation at all times, e.g. during the authoring process, is bound to the syntactic linear domain of the slide deck. After discussing the status quo and identifying some of its problems and drawbacks, we explore alternative ways of structuring and navigating presentation visuals and investigate how this influences the authoring process, the presentation delivery, and the ability of the audience to perceive, process, and remember the content of a talk.

Presentation visuals are a hybrid medium; they are non-time-based during authoring and time-based during presentation.

After giving a short overview over the characteristics of presentation visuals as a medium, we give a short explanation how and why today's *slideware* format has emerged from the historical context. We then analyze the status quo of navigation in slide-based presentation systems and discuss some of the problems that come from current interface designs but also the technical metaphor of the format it-

The predominant format of presentations as slide decks is due to historical reasons and causes interaction problems.

self. Some of these problems have already been identified and solutions proposed in related work; we will summarize these findings and extend them by proposing FLY, our own interpretation of how the medium can be represented in a way that respects the semantic structures required to reify the conceptual models of different user groups. This will, again, be done by following the four steps of our interaction model for semantic navigation in digital media (cf. 2.2.4 "Generating New Interfaces Using the Combined Model").

With three equally important user roles for presentation visuals, we focus on the evaluation step in this chapter.

This chapter focuses on the fourth step of our model to generate semantic navigation interfaces, explaining in detail how the concept and actual system have been evaluated with different user groups and their respective tasks. The individual expectations of these groups—represented by the author, the presenter, and the audience—towards the medium and the methods to navigate it to facilitate the creation, mediation, and consumption of its contents require different forms of experiments and analyses. We present the body of studies we have conducted for this purpose and also point out where additional work remains to be done in the future.

Parts of the material found in this chapter have already been published in conference and thesis papers: The original idea of how we could navigate presentation visuals in a semantically more meaningful way was devised by the author and a first draft published by David Holman [2006]. In his Diploma Thesis, Leonhard Lichtschlag concentrated on the authoring aspect of presentation visuals, conducting two user studies and iteratively developing the first fully functional FLY prototype [Lichtschlag, 2008]. A combined presentation and video navigation system, which uses the semantic structure of the FLY presentation format to navigate through videos taken during the presentation, was created by Christian Corsten for his Bachelor's Thesis [Corsten, 2009]. Thomas Heß refined the FLY software prototype for his Master's Thesis and performed both an extensive study on the influence of the new presentation format on the audience and a survey on emerging authoring strategies [Hess, 2011]. A mobile version of the FLY presentation system for the iPad™was implemented by Claude Bemtgen for his Bachelor's Thesis [Bemtgen, 2012]. All thesis work regarding FLY has been done under the guidance of either the author alone or in cooperation with Leonhard Lichtschlag.

These publications will also be again cited explicitly in the text below, where it is applicable.

EXCURSUS: PRESENTATION VISUALS:

Technical Representation

Presentation visuals are today usually represented by a succession of slides, which are static or animated frames of text and imagery. The whole presentation document therefore consists of a single linear list of such slides, and each slide consists of a single linear list of staged build stops—in the most simple case, there is just one stop, and the slide can be represented by an image. While the slides and their builds are presented strictly sequentially, there is still a major difference to a video in that there usually is no fixed playback rate; the temporal structure *can* be pre-determined by the presentation author, but usually each upcoming sample is navigated to manually by the presenter.

Syntactic Structure

The support domain of the syntactic structure for the kind of slide-based presentation visuals we are concerned with here is the linear, sequential list of slides. We will not consider staged slide builds explicitly, because they can also generally be realized by using multiple slides. The slide frame itself is a second-tier syntactic structure that supports the spatial arrangement of a slide's actual content. While the slide list is the main syntactic structure for navigation inside a presentation document, the spatial arrangement is commonly used to sequentially refer to individual bits of content that are visible at the same time.

Semantic Structures

Maybe like for no other medium, the semantic structure of presentation visuals is tied to the contents of the presentation. Since it is the singular purpose of the visuals to help conveying, explaining, teaching, or convincing an audience to believe the content, McLuhan's often quoted "the medium is the message" [McLuhan, 1964] is especially appropriate here. The semantic structure thus has to be a reflection of the structure of the topic that is to be presented: logical or referential links between subtopics, different views onto a topic, and representations of the content on different scales of interpretation are all part of the semantic structure.

It should be noted that, because of this special relationship between content and semantic structure, the latter will potentially be different for different 'users' of a presentation document. In particular, we must closely examine the roles of the *presentation author*, the *presenter*, and the *audience* to find a navigation technique that is adequate for all three.

5.1 A History of Presentations

Presentation quality is often associated with the quality of the visual support.

Giving good talks and convincing presentations is considered an important skill for today's career paths, in academics as well as in other professional fields. While there are speakers that rely on oral presentations alone, most will accompany their talk by some form of visual support such as blackboard sketches, flip-chart diagrams, or digitally projected slides. Especially the latter has become almost the canonical form for a presentation: preparing a talk often means creating a deck of digital slides. Similarly, the perceived quality of a presentation is often strongly influenced by the perceived quality of the slide deck [Tufte, 2003; House et al., 2005].

Presentations have a long history.

Earnest [2003] describes in his chronology of presentation visuals that the task of presenting a topic to inform or illustrating a point of view to convince others has been a recognized skill for more than 2000 years; in the early republics of the Greek and Roman cultures, public speech was what could sway the political opinion in one direction or another. Consequently, the art of rhetoric had already been studied extensively in both Greek (e.g. in [Plato, 380 BC] and [Plato, 370 BC]) and Roman (e.g. in [Unknown Author, approx. 90 BC]) literature. No mention of visual presentation support is found in these documents, however—they focus solely on the spoken word and the supporting body language.

Visual presentation support emerged just over 200 years ago.

It was not until the industrialization that presentation visualizations appeared: After blackboards had been introduced in 1801, their use quickly spread and was widely established in the middle of the century. Leveraging the ability to reproduce real world imagery through photography, 35mm color slide projection became a popular means for visualization in the 1930s. Overhead projectors, as they are still being employed in classrooms, saw their first (military) use in 1945. And by 1960, Television CRT screens were widespread in the US (CRT projectors lacked the necessary brightness for large screens until Advent Corporation released a 7 foot system in 1972 [Hornbeck, 1998]). Today, most of these technologies have been succeeded by digital projectors of various design or by large-scale digital display panels, which are used to show the output of some presentation software—usually a series of (possibly animated)

slides that can contain text, images, and even video or audio clips.

Digital presentation software was originally a tool to design physical slides.

The first commercially available presentation software package, *PowerPoint*, is still the most widespread today [Parker, 2001]. It was originally designed by Forethought for the Macintosh platform in 1987 but bought by Microsoft in the same year to be re-released as part of Microsoft Office in 1990. At that time, PowerPoint's main purpose was still to aid the design and productions of physical slides—either as 35mm diapositives or printed-out overhead slides. Today, PowerPoint and similar software packages are mostly used in conjunction with digital projection or display technology, thus—in theory—liberating the tool from the rigid restrictions of the physical slide format. The format, however, prevails; most presentation software is still *slideware*, deeply rooted in the slide metaphor and inheriting its technological limitations (cf. 2.1.4 "Edge Cases and Exceptions"). Therefore, presentations that consist of a linear sequence of images and bullet points grouped into equally shaped rectangular regions are not an exception but the rule.

5.2 Describing Slide Navigation in Standard Interfaces

The strength of slides is creating a linear, sequential narrative; but often topics are more complex than that.

Slide-based presentation visuals are a series of rectangular information snippets, usually filled with visual media such as sketches, videos, photos and text, and shown in a pre-defined sequence. For some forms of content, this may be completely acceptable; telling a story with just one plot line, for example, or presenting a single-track schedule for a project works well when using slides. In these cases, the content has a linear dominant structure (here, it is temporal), and the projection onto the linear, sequential slide format is compatible with that (Figures 5.1 and 5.2). Other kinds of content, for example the explanation of complex and highly interconnected topics, are more difficult to project onto this one-dimensional, linear container. Regularly providing overviews over subtopics or the whole talk to give context to the content currently in focus, as it is recommended, does not come naturally when using slide

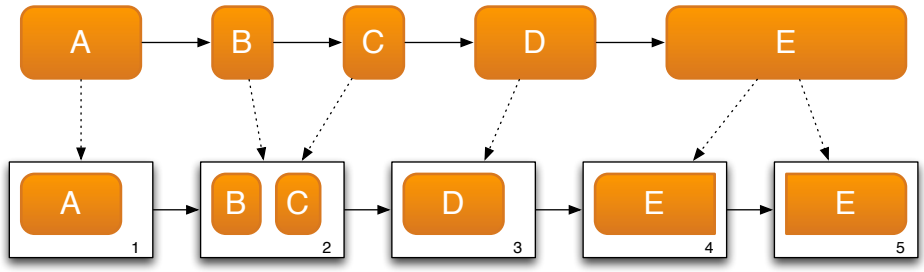

Figure 5.1: Content with a dominant linear structure mapped to a sequence of slides. Since the semantic structure of the content and the structure of a slide deck are topologically compatible, the mapping is easy. The semantic chunking of the content, however, may already not be representable in the single, fixed chunk size that slides offer.

Projecting complex topics into the slide format requires multiplexing and deliberate redundancy.

decks. Also, visualizing cross-links between different parts of a subject or even just following branching or merging lines of explanation can require an immense amount of care and deliberation on the sides of both the slide author and the presenter to do well with slides. In media like this that possess a possibly high-dimensional semantic structure, which, during the authoring, had to be projected onto a one-dimensional syntactic structure, multiplexing is often the only way to deal with these situations. For slide decks, this means repeatedly inserting copies of overview slides and frequently switching between related topics that possibly had better been shown in parallel to allow direct comparisons (Figures 5.3 and 5.4).

Slideware divides content and navigation into chunks that are defined by the available space on one slide.

Apart of the linearity of slide decks, another point is the granularity at which information can be navigated and shown at a time. With slides, content chunking is often not determined by natural grouping of the content or even the human capabilities of the intended audience (cf. Figure 5.1). Instead, the decision of navigation granularity, and thus of what to show during the presentation at any given time, is mainly governed by what fits the rectangular frame of a slide. Slides, in other words, effectively localize design and decision making to arbitrary chunks of content.

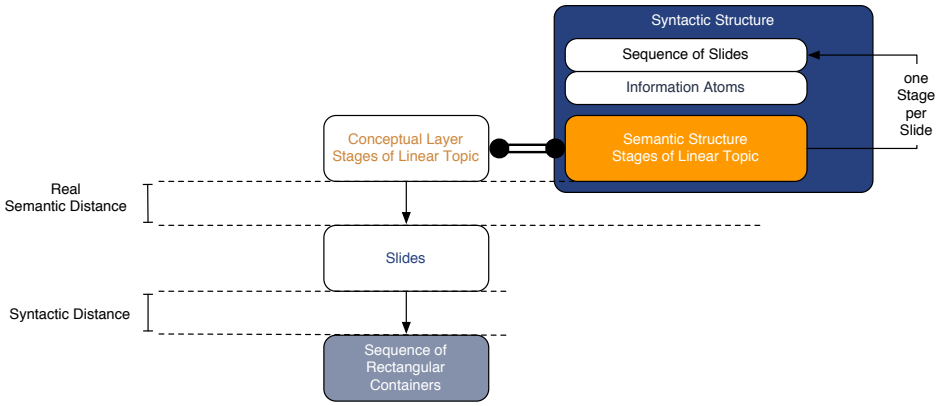

Figure 5.2: Combined navigation interface model for a regular slideware when presenting a purely linear topic like, for example, a story with a single plot. The semantic mapping of how the stages of the topic are represented on the slides is comparatively simple.

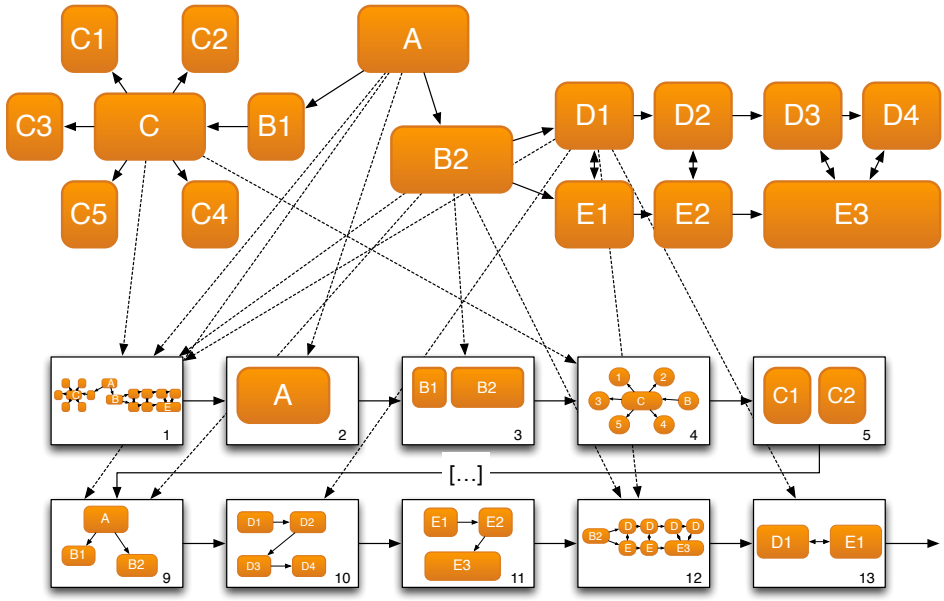

Figure 5.3: Content with a complex structure mapped to a sequence of slides. The semantic structure of the content and the structure of the slide deck are topologically different. Mapping the content onto the slides in a way that is optimal for the audience takes a great amount of care and deliberation. Content has to be repeated in different contexts, and overviews have to be inserted regularly.

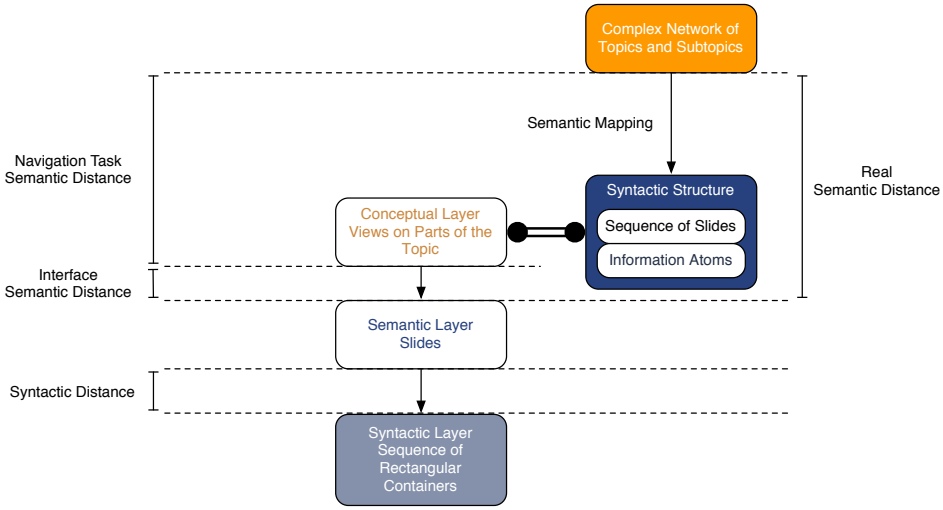

Figure 5.4: Combined navigation interface model for a regular slideware when presenting a complex topic with many semantic relations. The semantic mapping of how the different parts of the topic—as well as overviews or side-by-side comparisons—are represented as a sequence of slides has to be determined by the author and becomes an immutable part of the slide deck.

5.2.1 Common Problems of Slideware

Slideware has been subject to vocal criticism.

As giving talks or presentations can be a very stressful situation [Moscovich et al., 2004], and nobody likes to find themselves unprepared at that time, presentation authors often spend a considerable amount of time and effort to prepare the presentation visuals. They need powerful and reliable software for this task, and, unsurprisingly, current slideware packages are among the most frequently used computer applications overall [Parker, 2001]. Yet, there seems to be a consensus that slideware-driven talks often are of minor quality, and there is a vocal group of slideware critics who claim that PowerPoint and similar software are to blame for this (e.g., [Gopal and Morapakkam, 2002; House et al., 2005; Parker, 2001; Tufte, 2003]). Edward Tufte is only one prominent member of this group with his well-known entertaining and sometimes polemic writings about the influence PowerPoint has on presentation quality [Tufte, 2003]. Next to him, a large body of slideware-related criticism by other authors exists:

Slides often take precedence over the talk and homogenize presentations.

House [2005] states that instead of supporting the talk, slides often substitute the talk, with the presenter being degraded to just orally supporting what is shown on the slides. Also, talks that are projected onto the concept of slides would often become more similar, because the constraints and limitations of the format would narrow down the possible variations of expressing ones ideas.

Slides may take control over the talk or become its subject rather than its support.

Farkas [2005] criticizes how with slideware, presenters often rather talk about their slides than the topic itself. The slides, again through their linearity and their pre-defined pacing, take control over the presentation away from the presenter. This can be observed especially at slide changes when a presenter pauses for an instant to re-orient to the 'new' content that appears on the slide.

Slides may truncate the creative space, because they are tedious to change after they have been authored.

Johnson [2005] and Craig [2006] share the view that using slides for a presentation creates a finalized mindset, comprising only those parts of a topic that have made it into the slide deck. This discourages making ad hoc changes to the structure or the content of the talk and inhibits discussions about related topics that are not part of the final set.

Slides invite micro-scale beautification instead of getting the big story right.

Another widespread point of criticism [Wright, 1983; Parker, 2001; Tufte, 2003; Good, 2003; Li et al., 2003] is that presentation authors often concentrate of the beautification of single slides instead of communicating the gestalt of the topic or making high level decisions about the structure of the talk. Slides narrow the focus during both authoring and presenting the talk to a 'slideful' of content, hiding away the relations with and connections to other subtopics.

Slides are a poor tool for education.

Parker [2001] as well as Johnson and Sharp [2005] believe that the static nature of slide decks makes them not well suited for education. Slides typically are used to present results, not to illustrate the process of obtaining them, which is a crucial point in teaching.

Many points of criticism could be avoided if slides were used properly.

Opposing views exist, of course. And ultimately, it is not the tool which is to be blamed but the author or presenter who uses it—often not to the best of the tool's capabilities. This is why recently a number of researchers—among them Holmes [2004], Brown [2007], Norman [2005], and Kjeldsen [2006]—advocate *media literacy* [Buckingham, 2006] and the need for proper training for presentation authoring and de-

livery. Although this is undoubtedly a valid argument, we still argue that presentation software could do a better job in helping non-professional authors and presenters to communicate their ideas to an audience effectively.

Some of the criticism is generally applicable to poorly prepared visual support for talks.

It should also be noted that the aforementioned points of criticism about slides being bad presentation visuals are not exclusive to PowerPoint but apply to most other presentation software as well. An interesting fact, however, is that such arguments had already been brought forward as early as in 1950 by Van Pelt [1950]:

> "We have seen overcrowded slides projected by machines that could not be focused. We have watched while speakers in a large room tried to use maps or charts that could not be read beyond arm's length. We have listened in vain as able scholars talked confidentially to a blackboard while writing illegible symbols with invisible chalk. We have fidgeted, mentally if not physically, as the remarks of a renowned scientist came to a dead stop while he readjusted some ill-arranged piece of apparatus or hunted for a scientific specimen to illustrate his point. The habit of using bad visual aids is rampant among those who 'speak to inform'."

The similarity of the lamented problems suggests that some of the weaknesses of PowerPoint that have been repeatedly pointed out over the last decade are actually rather rooted in the technical metaphor of slides that most presentation software still adheres to.

In the context of presentation authoring, slides exhibit three problems.

These points of criticism mostly apply to the presentation as well as the creation of presentation visuals; in Lichtschlag's Diploma Thesis [2008], he explicitly analyzed the potential problems of slideware in the context of the latter. In particular, three major problems can be observed [Lichtschlag, 2008]—*content cutting*, *time dominance*, and *falling into the detail trap*—that may force authors to adopt the sequential and discrete conceptual model of slideware [Parker, 2001; Lovgren, 1994].

Content Cutting

Content cutting happens when the chunking of the content is influenced by the syntactic constraints of its form.

Slide decks are sequential collections of discrete, equally sized containers. Thus, they separate the content into a series of discrete chunks, acting like folders into which the information has to be sorted [Good, 2003]. Since the amount of content that fits any single slide is solely determined by the spatial size of its visualization, this separation and sorting of information chunks has to follow constraints that are independent of the content. Slide design guidelines that ask for a visual balance of the graphical layout on a slide and discourage authors from creating overloaded or very empty slides only add to this situation. The result is that the chunking often will be arbitrary in terms of the semantic structure of the content. This, in term, leads to a whole series of problems:

- Since content cannot physically span slide boundaries, it has to be repeated—in parts or as a whole—when referred to again later in the talk.
- If the visualization of a semantically coherent chunk of content overspills the slide frame, the slide either has to be left in a visually unpleasing state (and is likely difficult to parse and comprehend) or the chunk has to be split up between neighboring slides, effectively always hiding half of it.
- Similarly, comparing two aspects of a topic side-by-side requires either flipping back and forth between two slides or a special slide where both aspects are visualized next to each other. Due to the limited space on a slide, this often requires a miniaturized and therefore simplified visualization which dilutes the learned relationship between the signifier and the signified (cf. [Saussure, 1916]) and makes it difficult to retain the association between a topic and its visualized shape.
- Content that does not fit the regular visual chunking of slides—either because it would overspill a slide or leave it underpopulated—is often dropped entirely from a talk [Parker, 2001]. This is probably the most severe case of content cutting.

- Complex topics spanning a range of slides with only one slide visible at a time are reportedly difficult to present and difficult to understand as a whole [Good, 2003].
- Transitions between slides usually abruptly substitute one chunk of information with another. The granularity of how upcoming content is revealed is again fixed through the concept of slides. This makes transitions often neither calm nor predictable.
- Transition effects should only be used in a way that they are semantically meaningful and illustrate the content. Slides make this difficult [Zongker and Salesin, 2003].

Time Dominance

Time dominance causes non-linear relations between topics to be sequentialized to fit the linear syntactic structure.

Due to the linearity of slide decks, the timeline of a talk is syntactically hard-coded into the presentation visuals at creation time. Non-linear semantic structures have to be projected onto this linear domain and may lose their structural gestalt; in order to mitigate this effect, overview slides have to be deliberately inserted, and content has to be repeated to present non-linear relations between topics (Figure 5.3). This *Time Dominance* leads to a number of undesired effects for the author:

- Relations between topics can be presented at most in linear form, and the only possible 'gestalt laws' to express these relations are *closure* when being on the same slide or *proximity* when being temporally close in the slide deck, e.g., on neighboring slides. Some slideware systems offer hyperlinks on slides that allow to connect topics in a non-linear fashion; these connections, however, are opaque to the audience unless a link is invoked. Also, navigating hyperlinks on slides requires the presenter to acquire a pointing device, bring up a mouse cursor, aim it at the link, and click, which can easily interrupt the flow of the presentation and is a burden on the presenter (and often the audience).

- The *Time Dominance* of slides worsens the problem of *Content Cutting*: not only content that does not fit the slide frame is often excluded from a presentation, also content that does not fit the singular trail of explanation is more likely to be pruned. This hinders prototyping and exploration during the authoring process [Gopal and Morapakkam, 2002; Good, 2003], discourages from preparing several alternative explanations of a topic, and brings the danger of oversimplifying topics by communicating a streamlined and linearized mental model.
- Similarly, it is difficult to include optional material into the linear structure of a slide deck. Extra slides will either have to be skipped over during the talk if the presenter opts to not show them—which is often confusing to the audience [Moscovich et al., 2004]—or they have to be added into an 'appendix' after the last slide, which is then difficult to bring up at the right time.
- Content repetitions, like they are necessary for overviews and for multiplexing or revisiting ideas in slide decks, make later changes to the talk more difficult and prone to inconsistencies.
- Reusing a slide deck for a different talk, e.g. for a different audience or with a slightly shifted focus, will likely result in the need for a re-projection of the content onto a new timeline. This means that the slide deck has to be either jigsaw-puzzled together from salvaged fragments of the existing talk, which can easily result in poor consistency and a non-ideal 'story' arc, or that the deck has to be completely re-created, which is a waste of time. In any case, it is the author's burden to implement a system of version control for slide decks to manage these situations [Drucker et al., 2006; Moscovich et al., 2004].

Detail Trap

In most authoring environments for slideware, the editing scope for the visual design of a slide is determined by that single slide's frame. Thus, the author is visually and

Slideware authoring interfaces show only a local neighborhood of the content, luring the author into the *detail trap*.

interaction-wise locked in at the detail level without visual reference or easy access to the overall layout of the talk. She has no context for the current slide available to ensure it fits a consistent line of narration or explanation in the talk [Gopal and Morapakkam, 2002]; there is no 'stepping back' to look at the shape of the whole. Problems that result from the *Detail Trap* include:

- Limiting the focus to a single slide during authoring often results in authors basing their decision making only on this local chunk of information. This 'beautifying' of slides instead of refining the overall shape of the talk has been repeatedly observed and reported (e.g., in [Good, 2003; Li et al., 2003; Parker, 2001; Tufte, 2003]).
- The absence of a 'big picture' view, barring thumbnail collections of slides, in current slideware not only impedes the author's ability to create contextually consistent presentations but also requires her to manually create such overviews and context visualizations. This, of course, is not only a lot of work but also requires changes to other slides being reflected on these overviews as well, which leads to inconsistencies if forgotten.
- Although this creation of overviews is considered good practice and helps the audience later [Good, 2003], it is often neglected, as the authoring software does not offer the affordance.

These problems are all related to the dominant syntactic structure of the medium.

In summary, we see how these limitations, which slideware inherits, because it is constrained to the technical possibilities of physical slides, adversely affect not only the authoring process itself but also how authors are able to design and think about presentations. In this context, *content cutting* and *time dominance* both actively separate where humans associate. They are examples for how offering navigation only in the syntactic structure of a medium makes the navigation inefficient and difficult. Maybe even more importantly, it also influences the mental model of the medium and, thus, its creation process and ultimately its content. With the technical constraints that required content to be presented on slides long gone, we will try to abandon this re-mapping of the mental model of a topic onto the rigid

structure of a slide deck. In the remainder of this chapter, we describe the design of FLY, our presentation software that allows semantic direct manipulation navigation of presentation visuals, and how it affects the authoring, presenting, and learning processes.

5.3 Fly, a Semantic Presentation Tool

Semantic navigation interfaces from time-based media are not applicable here.

Developing a concept for semantic navigation in presentation visuals cannot be done in the same way as for audio (cf. 3 "Time-based Media: Orchestral Music" (p. 65)) or video (cf. 4 "Time-based Media: Video Scenes" (p. 89)) because of two main differences in how we use these media:

Firstly, for video and audio, the syntactic structure reflects a fundamental property of the medium and its content—everything that happens in these media happens over time, so the fact that the syntactic structure coincides with the temporal support domain is not only suitable but necessary to be able to represent the content. Therefore, it was a valid approach to provide a mapping between the semantic structure and the syntactic structure for semantic navigation.

The syntactic structure of slides is a technical relic.

Slide-based presentation visuals are different; the syntactic structure of this medium is not a reification of any concept that would be inherent to the content. In a lecture, for example, the knowledge that should be transported by means of the presentation and its visual support does not necessarily share any structural characteristics with the linear, sequential, and discretely chunked slide deck. In many cases, it can rather be represented as a hierarchical network of semantically related bits of information, like a concept map or a hypertext document. Insofar, the temporal linearity and sequentiality of the syntactic structure are artifacts of the act of presenting the content and not of the content itself. The technical implementation and the user visible representation of the medium, unfortunately, have already embedded these concepts.

This difference negates our straightforward approach of just modifying an interface layer to represent the semantic structure and then constructing a semantic mapping to leverage

the existing syntactic navigation facilities. While we can still follow our four-step-approach, we will need to change the syntactic structure of the medium as well in the process.

We have to consider different user roles: author, presenter, and audience.

Secondly, in the last two chapters, we did not have to deal with the different user roles pertaining to the medium. In the case of user A navigating a video scene, for example, we paid no attention to the implications *DRAGON* might have for another user B, who is *watching* the process. Similarly, the idea of navigating video is simply not applicable to the act of creating or shooting a video, only to editing it later.

With presentation visuals, we have another situation; navigating the presentation visuals means different things in the context of the three stages of the medium's 'lifecycle'—*authoring*, *presenting*, and *learning and understanding*—and for the respective groups of users who perform these tasks. And while specialized views on the medium and additional navigation techniques exist for, say, the author of a presentation, all three groups are usually faced with the same basic representation of the medium. This, again, calls for a more comprehensive approach in redesigning the navigation. And changing the established syntactic structure is one way to gain the necessary flexibility for offering appropriate views on the content and navigation techniques that support the primary tasks for all three user roles.

5.3.1 Finding a Conceptual Model

As outlined above, we have to understand the tasks of three distinct user roles if we aim for a unified conceptual model that is useful to all of them. Therefore, we start by reviewing the three aspects of this medium's 'lifecycle' and the three agencies that act as users in those aspects:

- the creation or *authoring* of the presentation visuals, performed by the *author*,
- the actual *presentation* of the topic by a *presenter* who is using the visuals,
- and *understanding, remembering, learning, and enjoying*

the content of the presentation, which the *audience* has to do.

Authoring—Author

Author and presenter are separate roles.

A mistake that is sometimes made when talking about presentations is that the author of the talk and the presenter are viewed to be the same person. While this may often be the case, this perspective leads to a conflation of the different roles; the task and context of the author and those of the presenter are distinct, as they differ in factors like *time constraints*, *undoability* of work steps, and *persistence* of the results. Therefore, we will analyze and discuss them separately.

Authors research the supporting material and create or compose the visualizations.

In our definition, the author is the one to design the structure of a talk, select any material to be presented, prepares the medial support for the talk (visuals, audio, handouts), and—maybe together with the presenter—composes any presenting aids like presenter notes or cue cards. The result of her work may in turn be the input for one or many instances of presenting. While for some presentations there may be no visual medial support necessary, we will restrict our examinations to the cases that are relevant for the discussion at hand.

Experience levels and work practice are very diverse for authors.

The author of a presentation should be an expert in the domain of the presentation topic, and researching and sense making of related material may be necessary before beginning to work on a presentation. When creating visual support for the talk, employing slideware is most common, but authors also use graphics packages, text processors, and even specialized animation software. In his analysis of presentation authoring practice, Johnson [1996] distinguishes between different types of authors: those who are professional slide authors, those who create slides peripherally for their main job, and 'slide clericals' with less training who often create run-of-the-mill presentations. He found out that with increasing expertise, authors tend to employ larger tool repositories. For very polished presentations, each slide may be designed separately in graphics software or an animation suite and then only composed into a slide deck by means of slideware.

Typical authoring is an iteration of: collection of material, organization, composition, and revision.

The *authoring process* has been studied in detail by Good [2003]. He found that typically the authoring begins with a collection of visualization materials—often even before a plan for the final layout, order, or arrangement has been made. Then, the process enters an iterative cycle of generating ideas, organizing them, composing the materials that illustrate the idea, and revising the results. For this, both top-down and bottom-up strategies for the design of the topic's gestalt are common. Good concludes from his observations that the ability to explore design alternatives is critical for successful presentation authoring: "The more difficult it is to explore alternatives, the fewer alternatives the presenter is likely to consider" [Good, 2003]. He also states that this exploration happens to a far lesser degree if the authoring environment lacks support for these activities: "In addition, formal structures can introduce modification costs that reduce the chances that an author will explore alternative organisations" [Good, 2003].

Presenting—Presenter

The presenter must deliver the information understandably; many different presentation styles exist.

The task of the presenter is, according to Holman et al. [2006], to present a topic clearly and understandably and to answer questions from the audience during or after the talk. There is no single best way to achieve this, and a plethora of different practices and styles exist, ranging from very speaker centered presentations with no or very little visual support, over minimalistic slide support—for example, the well-known Lessig Method (cf. [Buttigieg, 2010])—to talks that center around the visual presentation of the topic. The latter can often be observed in the context of teaching, like in lectures and classrooms.

Presenting is a cognitively demanding task that leaves little free resources for complex navigation.

Presenting is usually a stressful task, because the whole audience is focused on the speaker, so it is very important that any technological support (visuals, clicker, audio) does not 'get in the way' and does not require an undue amount of cognitive resources. For presentation visuals such as slides that offer no content-based navigation, this often means that navigation is restricted to linear, incremental advancement in forward and backward directions. These limitations are also mirrored in the available selection of presentation remotes or clickers; a detailed discussion on and comparison

Figure 5.5: Physical slides with presenter notes written onto the individual slide frames (source: [Lichtschlag, 2008]).

between presentation control devices for slides and for FLY can be found in [Bemtgen, 2012]. Another aid for the presenter are written cues: Traditionally, these were sometimes directly written onto the plastic frame of 35mm slides (Figure 5.5). Today, most software packages support displaying these presenter notes on an auxiliary display that is only visible by the presenter.

Presenters often have to present a topic multiple times in different contexts: different audiences may have different levels of prior knowledge or different information demands, and the time frame allotted for the talk may vary. Sequential linear media like slides thus usually require reauthoring of a presentation to make it appropriate for another context. This leads to the need for a stringent and disciplined versioning system of the different slide decks [Drucker et al., 2006; Moscovich et al., 2004], which is a problem of its own nature.

Presenters may have to adjust the scope or style of the presentation according to the audience.

Understanding, Learning, and Enjoying—Audience

The role of the audience is to understand what is being presented, to learn something from the content, and—ideally—to enjoy the presentation. From these three, the learning aspect is the one that is deemed most important and consequently has been studied in the most detail. Learning through media is a research field in its own regard; however, the question if media can have an influence on learn-

The audience should understand the topic and enjoy the presentation.

ing performance at all has been the subject of an ongoing debate for decades.

If the medium can have an influence on learning success is a heavily debated question.

Prominent voices in this debate are, among others, Clark [1983; 1994; 2001], who opposes the idea of improving learning of students and audiences through media, and Kozma [1991; 1994], a strong proponent of that idea. Clark repeatedly states that "media do not influence learning under any conditions" and that "[...] media are mere vehicles that deliver instructions but do not influence student achievement any more than the truck that delivers our groceries causes changes in our nutrition." While he dismisses the positive results found by some studies on the matter to be produced rather by better methods of instruction than different media, Kozma (together with Reiser [1994]) and Brown [1992] claim that they found different students to perform better or worse depending on the employed media. What is undoubtedly true is that there are successful teaching and education methods that depend on certain media. Clark's statements also seem not to fit to some of today's interactive media that allow dialog learning rather than just be vessels of information. The question remains unanswered, although a number of more recent studies [Hoyt, 1999; Russell, 1999; Ramage, 2002; Joy II and Garcia, 2000; Clark, 2001] mostly could not find significant influences (see Table 5.1). Russell still maintains a public database[1] of research articles and papers under the name of 'no significant difference' where authors can register their published experiments on the influence of media on learning.

Significantly better with technology	46
Significantly better in the classroom	3
Mixed results	7
No significant Difference	299

Table 5.1: Distribution of studies with effects on learning as categorized by Thomas Russel [1999].

[1] http://www.nosignificantdifference.org

An Information Landscape Model

Creating an optimal experience for the audience requires extensive preparation from author and presenter; even then, integrating ad hoc questions or demands of the audience remain a challenge.

It seems clear that while a well-prepared linear slide deck can probably be a suitable medium for the audience, this puts the burden of didactically arranging the material to be presented into this limited structure on the author of the presentation. The presenter then has to stick to this order of topics and to the line of explanation envisioned by the author, which, again, requires extensive preparation. Questions from the audience require ad hoc navigation through the sequential information space, often accompanied by the less than desirable visual experience of flicking through a large slide deck until the topic of the question has been found. Similarly, the presenter usually is not able to adapt the presentation to different audiences on-the-fly apart from rapidly skipping over sections of the slide deck.

Adding an extra spatial dimension helps with some of the problems.

When looking for a conceptual model that does not suffer so much from the aforementioned problems of linearity and chunking, more flexible and interlinked structures come to mind. And defining these structures over more than one dimension suddenly allows concepts like parallelism or branching to be easily and conveniently expressed in a spatial way. Arranging information in such a manner not only adds an extra dimension along which we can navigate, it is also free of a dominant direction, possesses a more intuitive concept of scaling and granularity, and is easily compatible with the ideas of direct manipulation.

Mind maps follow a similar concept.

One widespread method to structure information outside of presentations are mind maps: related information is arranged in a tree structure around a single central topic at the root, creating a spatial layout. Similarly, one could imagine the conceptual model for presentation visuals to be structured this way, where navigation through the information space would be aligned with the edges of the tree.

A semantic scale space representation could enable scoping and abstraction of details.

Of course, since not all presentation topics are strictly hierarchical, a tree structure might be too restrictive, and a free placement of information along both spatial axes seems more useful. Such a structure is then called a *concept map* [Novak and Gowin, 1984; Novak, 1990]. Also, to respect the concept of subtopics of varying complexity and to include the idea of topic overviews, we can make use of the

scalability of space and allow distance to the plane of information as an additional navigation axis. The resulting scale space model should ideally reflect the scale semantically, automatically abstracting groups of information bits or whole subtopics if they would become information clutter in that scope of the scale space.

The gestalt of the topic can be directly visualized, more easily navigated, and information more flexibly chunked and presented.

A conceptual model for presentation visuals that is defined in such a continuous spatial domain, instead of the discrete temporal domain of slides, would have several advantages: First and foremost, now the gestalt of the topic can be represented directly in the model, which makes the spatial arrangement of information semantically meaningful, leverages spatial memory, and thus helps information retention [Dillon et al., 1996]. Second, a spatial arrangement of information can later be easily represented graphically and navigated in a continuous way by using established spatial direct manipulation techniques. In contrast to switching between slides, this could give us seamless transitions between related subtopics, which prevents losing the context, because it is constantly being visualized. Zooming out for overviews could be done 'on the fly' whenever needed, further aiding spatial memory. And, in the same way, a presenter can always choose to alter the path of the talk through this space or deviate from a planned order of topics to accommodate for different audiences, meet time constraints, or show extra material in answer to questions. Third, the scalability of a spatial layout enables a much more liberal use of gestalt laws to chunk individual items of content. Instead of being pre-determined by the syntactic frame of a slide, groups of information items can now be built by giving them similar shape, placing them close together, or arranging them along geometric entities.

The temporal aspect of the presentation order can be completely de-factorized from the spatial aspect of the information arrangement.

Most importantly, however, this idea factorizes the conceptual model for presentation visuals into two orthogonal subspaces: that of the semantic structure of the topic, which can be represented spatially, and that of the temporal ordering of the talk during the presentation. Both can be manipulated separately, and navigation through the medium is determined by the layout of the content and not by the fixed ordering and chunking of content that slides impose.

5.3.2 Related Work

Parts of this idea of a more flexible model of information arrangement or the inclusion of an explicit semantic scale have already been proposed in earlier projects, some dating back to the early 90s. The following is a short list of the most important related work and by no means comprehensive. For more detailed discussions, we refer to [Lichtschlag, 2008] and [Hess, 2011].

Controlling Presentation Flow by Physical Interaction

Physical proxies for slides can break time dominance and can give the presenter a better overview.

The problem of slideware presentations being inflexible with regard to the order in which the material is presented has been identified before. One possible solution is to use physical proxies for the slides during the talk. These have the advantage that they can easily be re-arranged or spread out on the presenter's table for a quick overview (unfortunately just for the presenter, of course). Using physical slide proxies thus allows better random access navigation to specific parts of the talk; normal presentation software usually only offers navigation to the next and previous slides, or random access via the slide number, which, as explained above, is usually not related to the semantics of the content and thus of limited use. One has to note, however, that employing slide proxies links the presentation visuals even more closely to the technical metaphor, with all the consequences discussed in the beginning of this chapter.

Palette uses barcode augmented paper slides to trigger the slide projections.

Palette [Nelson et al., 1999; Pedersen et al., 2000] was one of the first systems that included slide proxies printed on paper cards to control the presentation. Every printed card automatically contained a barcode that identified the represented slide inside the deck. During a talk, the presenter scanned the barcode using a scanner mounted to the presenter's desk to advance to the respective slide. Advantages of the system were, as already stated above, the possibility to restructure the talk ad hoc within certain limits (e.g. skipping slides) as well as easier overview of and random access to the upcoming slides. The paper proxies were also helpful for the presenter to take notes of questions or comments from the audience. Resembling a deck of classical

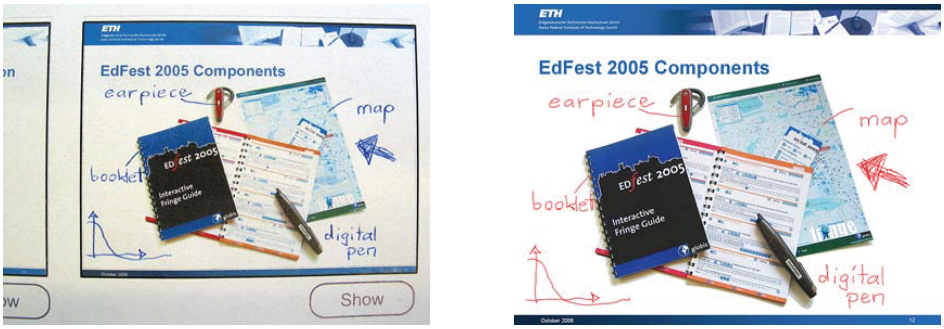

Figure 5.6: PaperPoint slide proxy (left) and digital projection (right). The slide deck can be navigated by tapping each slide's individual *show* button with a digital pen. Also, annotations made on the paper version of the slide propagate directly to the digital projection. Source: [Signer and Norrie, 2007]

cue cards, a widely used technique to aid the presentation, the slide proxies were easily accepted by presenters. They also were reported to facilitate discussions between presenter and audience, because cards could be quickly shared and passed around. Drawbacks of *Palette* included the limited re-usability of printed out slides—the barcode was only meaningful in the context of the slide deck it was created for, so slides could not be re-used for overlapping decks—and the fact that the presentation itself was still a normal slide show for the audience.

PaperPoint uses Anoto paper to trigger slides and to annotate them on-the-fly.

PaperPoint [Signer and Norrie, 2007] was an extended version of that idea using Anoto™[2] digital pen technology. A proxy version of the slides was printed on dot-coded paper, and the proxy contained printed buttons: a 'show' button for each slide and a set of global command buttons, similar to what can be found on most presentation 'clicker' devices. Using a Bluetooth Anoto pen, the presenter could tap on the buttons to directly invoke these commands. The second feature of *PaperPoint* was that sketches and inkings drawn onto the proxy could automatically be updated on the audience-visible projection in real-time (Figure 5.6). Thus, it combined the flexibility in the order of the presentation with the ad hoc augmentation capabilities of tablet presentation systems like *Classroom Presenter* [Anderson et al., 2004, 2007].

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

Concept Maps and Graph Layouts

There are already a number of presentation systems that are based on concept maps.

A different way to break up the strictly linear and predefined order of slide presentations has been pursued by systems that organize the content in structures with higher connection degrees. For this purpose, we make no distinction between mind-maps [Buzan, 1991], concept-maps (cf. [Novak, 1990; Carnot et al., 2001; Wiegmann et al., 1992; Chau, 1998]), hierarchical maps or other graph structures and layouts; all have in common that the syntactic structure can be designed to represent the semantic structure of the topic and that we can use them as frameworks to navigate through a talk. This also allows different traversal paths of a network of connected topics; presentations can thus be altered according to the occasion, or multiple paths pre-planned for different audiences or time frames [Shipman et al., 1996, 1998]. The concept of offering different ways to traverse and information structure is also an established paradigm in the related areas of hypermedia [Zellweger, 1989] and distance learning [Goyal et al., 2006].

Moscovich proposed laying out PowerPoint slides in a graph to enable different presentation paths.

One system that allowed to arrange slides in a graph structure was proposed by Moscovich et al. [2004]. The slides were first authored in PowerPoint and then, in a second step, imported into the graph tool where they could be connected to form a directed hierarchical graph (Figure 5.7). During the presentation, the presenter could always choose after each slide which of the outgoing edges in the graph should be taken. Together with the hierarchical structure, this provided an easy way to choose between short explanations and in-depth discussions of a subtopic or allowed to skip it entirely. The graph was only visible at the presenter's display, however; for the audience, the presentation visuals resembled a normal PowerPoint slide deck.

The graph shows how far the talk has progressed.

A similar prototype system by Gopal and Morapakkam [2002] gave the presenter the possibility to show the graph of slides to the audience. Through a set of Visual Basic macros in PowerPoint, the slides could be assembled in a graph and the graph be visualized in a side column or as a fullscreen overview for the audience. Slides that had already been visited were shaded gray to give the audience a perspective on how far the talk had progressed through a topic. The system was tested in the class room with stu-

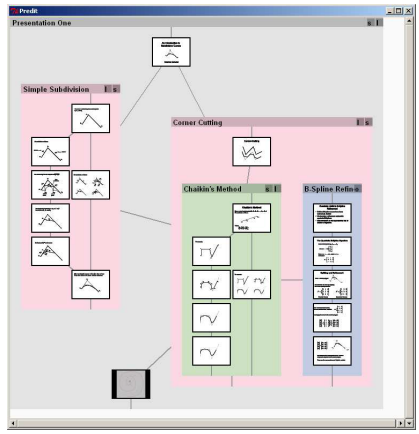

Figure 5.7: Moscovich's tool allowed authors to arrange PowerPoint slides in a graph structure and then present them by navigating this graph. Source: [Moscovich et al., 2004]

dents and received broad acceptance among the participants: over 70% reported that they felt it made learning easier and that they liked this form of presentation. Also, referring back to a certain subtopic for questions or additional explanations was reported to be very easy thanks to the visible overview map of the talk. Observed problems include the amount of extra work needed to author the graphs using the prototype system and the abrupt transitions between content and meta-content when switching between content slides and map view.

Zoomable Presentation Interfaces

Zoomable interfaces for presentation are less structured but allow the author to semantically scope the material.

As discussed in 5.3.1 "Finding a Conceptual Model" above, we not only want to exploit the beneficial properties of spatial layouts but also want to facilitate easy overviews and seamless transitions between subtopics and different information hierarchy levels. One class of interfaces for the exploration of such structures are *zoomable user interfaces* (ZUIs), pioneered by Bederson, Hollan, and Perlin in the early 90s [Perlin and Fox, 1993; Bederson and Hollan, 1994]. In contrast to graph layouts, they don't use edges but the

spatial layout itself over a range of scales to convey these relations. A number of zoomable presentation interfaces have been proposed, with Prezi[3], a fairly recent one that succeeds some of our work on FLY, being one of the first commercially successful presentation applications of this kind.

CounterPoint is a presentation ZUI but is still based on slides.

CounterPoint [Good and Bederson, 2001, 2002; Good, 2003] is a presentation ZUI where authors can arrange slides spatially and at different scales on a canvas. Similar to the systems by Moscovich et al. [2004] and Gopal and Morapakkam [2002], the slides are first created in PowerPoint. While the main structuring element of information is still a slide, *CounterPoint* allows to place simple text labels on the canvas next to single slides or groups (Figure 5.8). The layout can also be created automatically if a tree-hierarchy of slides is given. The final presentation is a sequence of viewports at different scales onto the canvas, whereby each viewport can show a single full slide, the whole arrangement, or anything in between. For the audience, the transitions between these viewports are smoothly animated, helping to retain the context and understand the shape of the topic through the visual layout. Authors can create multiple independent sequences over the same information canvas. In an experiment, feedback from 73 presentations delivered with *CounterPoint* revealed that the ability to show overviews as well as focus on details was appreciated by the presenters. Also, the smooth transitions helped the audience to stay oriented, especially when jumping between subtopics.

The pptPlex plug-in is less flexible.

A system very similar to *CounterPoint*, but provided by Microsoft as a plug-in for PowerPoint, is *pptPlex*[4]. A special background slide takes the role of the canvas in CounterPoint, the actual slides of the talk are filled into pre-defined placeholders on that background. Further nesting of slides is not possible. While users can modify these included background templates and even create their own, very little flexibility is offered when it comes to presenting: The viewport is automatically zoomed and panned to the next slide in the linear order of the original deck—changing this order or optionally branching to a different sequence are not directly supported. Overviews can be given, however, by zooming out, and the talk can be resumed at a different slide by click-

[3] www.prezi.com
[4] www.officelabs.com/projects/pptPlex

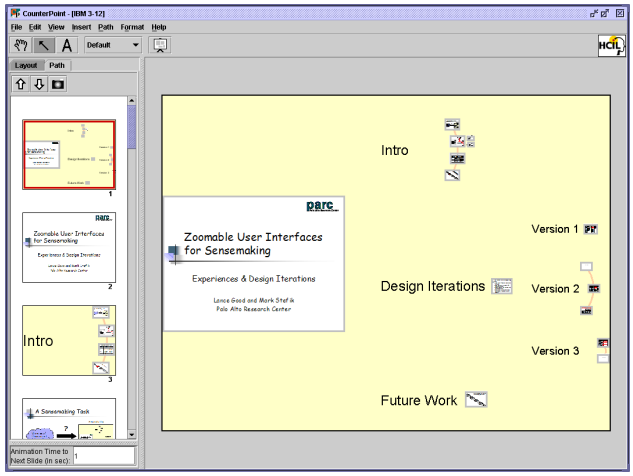

Figure 5.8: CounterPoint allows the author to arrange PowerPoint slides freely on a zoomable canvas. Additionally, labels can be placed next to single slides or clusters, and the software supports multiple different presentation paths over the same slide arrangement. Source: [Good, 2003]

ing on it. The latter, of course, requires a pointing device to be available during the presentation.

Prezi, a slide-less presentation ZUI, already embodies most of our ideas for FLY.

Prezi (formerly *ZUIPrezi*) is, as already mentioned above, a commercial web application implementing a ZUI to author and show presentation visuals. The most important point that sets it apart from *CounterPoint* and *pptPlex* is that it follows our idea of completely abandoning the concept of slides: In *Prezi*, content elements are freely pasted onto an infinite, zoomable canvas. Frames can be placed on that canvas as alignment hints for a certain viewport to be be shown in a presentation, but that is optional (Figure 5.9). Authoring a presentation consists of two steps: First, the information landscape is created by arranging the content media (text, images, videos, sketches, and alignment frames) on the canvas at different positions, angles, and zoom levels. Second, the author defines a presentation path over this information landscape by creating a series of viewport stops at the desired locations. These stops are created either by setting the viewport via direct manipulation pan and zoom to show a part of the canvas and then taking a 'snapshot' or by selecting an object or alignment frame on the canvas directly, which creates a viewport stop that is aligned with the respective bounding box. Multiple paths or branching

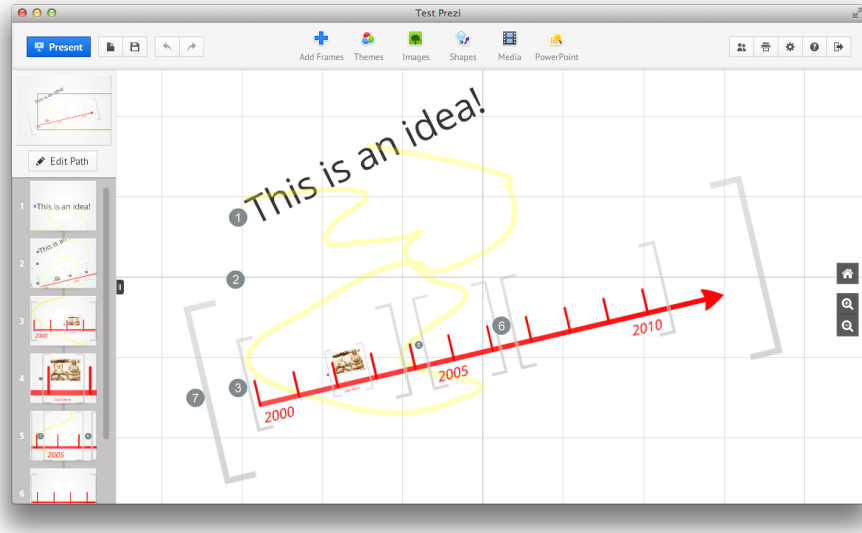

Figure 5.9: *Prezi* is a web-based presentation system that builds upon the same canvas metaphor as FLY. Information is laid out on an infinite plane at arbitrary zoom levels. In this way, authors can represent complex topics through their distinct visual gestalt.

paths are not supported. At presentation time, the author can choose to freely navigate over the presentation canvas, again via direct manipulation pan and zoom, or to advance on the pre-defined presentation path. Advancing on the path results in automatically created, smooth transition animations between the saved viewports.

Some studies were conducted using Prezi.

Parts of our own work on FLY predates the first release of *Prezi*, and some of our studies could have been conducted with either system. Because it is so close to our idea of how to structure and navigate presentation visuals, and because today *Prezi* is in widespread use, we also used it for one of our later field studies (cf. 5.3.5 "Fly Case Study—Analyzing Canvas Presentations in the Wild").

The first fly prototype was based on a less flexible tree structure.

The last system in this list of related work is the first prototype of FLY [Holman et al., 2006], our own approach at investigating a different navigation paradigm for presentation visuals. The prototype was developed to run a first formative proof-of-concept study, checking if spatially ar-

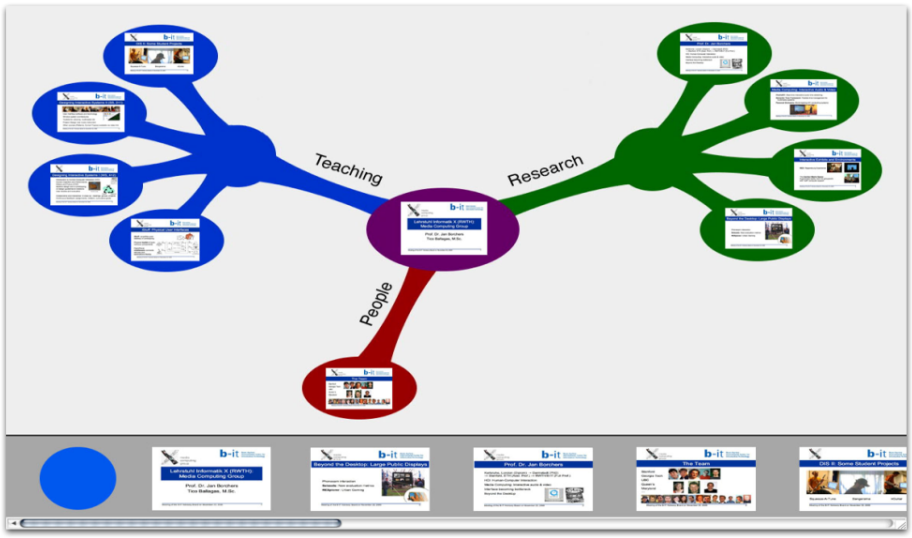

Figure 5.10: The first prototype of Fly. This version only represents a small subset of the ideas for canvas-based presentations; it was mainly used to conduct a formative study. Source: [Holman et al., 2006]

ranged presentations could be understood by the audience as well as the conventional time-multiplexed slide decks. In contrast to all later versions of FLY, the talk is always tree-based, the individual nodes are authored using slideware, and the nodes are laid out on the presentation canvas automatically (Figure 5.10). Navigation through the presentation is either direct manipulation pan and zoom or automatically animated smooth transitions between viewports that show single nodes or overviews. Holman et al. conducted a study in which they compared content retention between two groups of HCI students after attending a talk with either FLY or PowerPoint presentation visuals. While they found that FLY slightly outperformed PowerPoint on average, the results were not significant.

Content cutting can only be solved by abandoning the spatial rigidity of the slide frame.

The previous work presented here mainly addressed the problems of *time dominance* and *detail trap* by allowing to arrange slides in graph structures and by offering explicit visualizations of the context. The conceptual model for our FLY navigation additionally questions the information chunking imposed by the rigid structure and hard boundaries of the slide frame, which can lead to *content cut-*

ting. Our idea is that in breaking the linearity of the slide deck and the boundaries of the slide frame, FLY could help authors to visualize their flow of ideas, communicate the structural gestalt of complex topics, and not be discouraged from exploring emerging ideas.

5.3.3 Determine the Semantic Mapping

In the last two chapters on audio and video navigation, we always used the construct of the semantic mapping to facilitate navigation through the semantic structure, which represents the conceptual model of the medium; we did this by transforming the users' navigation gestures, which referred to the semantic structure, through the mapping into the syntactic domain. Then we leveraged the existing software infrastructure, i.e., the technical representation of the medium together with its syntactic navigation capabilities, to perform the actual navigation and presentation of the result (cf. 2.2.2 "Media Model" (p. 49), 3.2.2 "Determine the Semantic Mapping" (p. 72), and 4.2.2 "Determine the Semantic Mapping" (p. 108)). The impression for the user thus was always to directly interact with a representation of the semantic structure.

Finding an initial syntactic to semantic mapping is difficult for information canvases.

One core mechanism of this idea was that we could, in some way, construct the semantic mapping by inverting—sometimes globally, sometimes only locally—an initial mapping from the existing syntactic structure into the semantic structure. In the theory chapter, we presented different possible ways to arrive at this initial mapping (cf. 2 "Generating New Interfaces Using the Combined Model"). In the case at hand, the combination of the existing syntactic structure and the envisioned semantic structure of presentation visuals does not lend itself very well to this approach. The reason for this is twofold:

The syntactic structure with the slide sequence as the support domain is unsuited for more flexible navigation approaches.

First, since we want the semantic structure to reflect the conceptual model of the information space, the initial mapping would have to map the slide structure of a presentation document to that space; this mapping would be extremely difficult to define (cf. Figure 5.3). Automatic construction of the mapping is out of the question, and manual construction of the mapping basically requires the whole semantic struc-

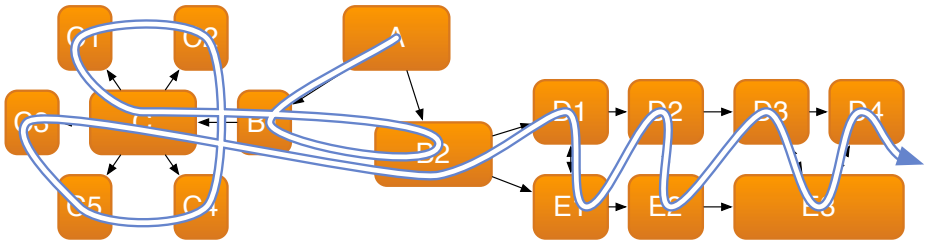

Figure 5.11: Semantic mapping in Fly. Instead of defining a semantic mapping into the syntactic structure of the medium, in FLY, we let the semantic structure be created explicitly in the authoring process. After this, it is easy to overlay a syntactic structure like the temporal order (blue path).

ture to be manually created first. Moreover, the initial mapping would be almost the opposite of being surjective—resulting in a semantic mapping that is defined only locally at sparsely distributed locations in the semantic structure. Such a semantic mapping, however, is almost useless for the semantic navigation that we intend to achieve. Also, this whole process would still leave us with the slide as the smallest navigable syntactic unit—which we have already identified as one of the things we want to change with FLY in the first place.

We want the medium to be authored in a representation of its semantic structure.

Second, we also want the authors of presentation visuals to be able to express their ideas directly in the semantic structure—ideally, the whole authoring process takes place in this space and then defines a mapping to the syntactic space of a linear presentation sequence after the fact (Figure 5.11). An explicit and tangible representation of the semantic space would also benefit the audience in terms of spatial memory utilization and knowledge conveyed by the visual gestalt of the spatial information layout.

If we author the medium in a representation of the semantic space, it can also be syntactically structured in this way.

Both of these points suggest to change the medium in a more radical way than just designing a semantic navigation interface on top of its existing representation. We thus turn the process around like outlined above; we change the representation of the medium to a direct reification of the semantic structure (Figure 5.12) and have the author of a presentation document also create the semantic mapping in its final direction—essentially following the pattern of *'manual definition of the initial mapping as integral part of the media authoring process'*, which we have proposed in the the-

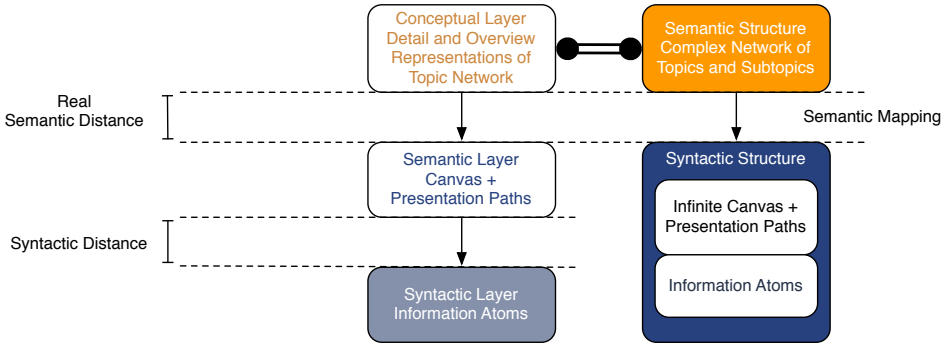

Figure 5.12: Combined navigation interface model for FLY. In this case, the medium is altered in its format to have the semantic and syntactic layers of the interface directly mirror the semantic and syntactic structure of the content.

ory chapter (2b "Generating New Interfaces Using the Combined Model"), just in the reverse direction.

Our new syntactic structure can explicitly represent the spatial and temporal factors separately.

The author can create her work directly in a higher dimensional space—as a spatial information landscape with optional semantic scaling—without any of the restrictions imposed by the slide format and then define one or many presentation sequences as a sequence of locations in this space. Thus, we can get a multitude of syntactic structures: Each of them is determined by projecting the semantic structure through one of the semantic mappings (Figure 5.11). And, more importantly, each of them can cater to the needs of a different audience or respect different time or complexity constraints on the presentation that may arise when the final talk is held by the presenter.

5.3.4 Designing the User Interface for FLY

The FLY interface should allow users to interact with the syntactic structure of the canvas and presentation paths using direct manipulation.

To ensure that the interface facilitates our goal of having the visual gestalt of the content resemble the semantic gestalt of the presentation topic, we chose FLY's underlying interface metaphor to be that of a directly manipulatable, layered collage or continuous arrangement of information atoms on an infinite canvas (Figure 5.13). We tried to avoid any syntactic—and therefore content-independent—granularity constraints, orderings, borders, or separations unless they were conscious design decisions by the author

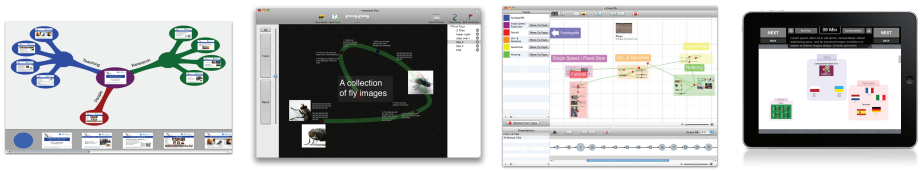

Figure 5.13: Screenshots of all four FLY prototypes. Our idea of canvas-based presentations has consequently been extended and refined through several iterations. Sources: [Holman et al., 2006; Lichtschlag et al., 2009; Hess, 2011; Bemtgen, 2012]

and were meant to communicate a fact or quality inherent in the content. The hierarchical graph structure of the first prototype (cf. 5.3.2 "Zoomable Presentation Interfaces"), which was based on concept maps [Holman et al., 2006], was therefore dropped.

FLY differs from other ZUIs by providing semantic zooming and a fixed zoom depth.

This interaction technique of pasting content—digital media such as text, images, or videos—freely onto an infinite canvas at different scales is, of course, akin to existing zoomable presentation interfaces like the ones discussed above (cf. 5.3.2 "Zoomable Presentation Interfaces"). However, two important differences set the FLY interface apart from most other ZUIs:

First, we decided against an unrestricted range of zoom levels to access the scale space of the semantic structure. Instead, similar to a city map, we created a 'ground plane', a level on which the most detailed representations of subtopics can be placed. We hoped that with this limitation we could counter the feeling of disorientation or getting lost that users often report when using regular ZUIs [Furnas and Zhang, 1998; Hornbæk et al., 2002; Bederson, 2010]. Also, allowing infinite zoom-ins might again impose the risk of the *detail trap*, which we want to avoid. On the other side of the granularity spectrum, FLY has a maximum 'overview level', which is the furthest distance from the ground plane that authors can place information on. A finite number of discrete semantic levels between those two extremes are directly accessible through the UI, the space in between can be accessed by continuously zooming in or out.

Second, FLY supports semantic zooming, a term coined by David Fox [1993], by placing different representations of information on top of each other, each at a different layer. An overview at a high level thus can reflect the shape of the topic without the distracting effect of scaled down details dissolving into indistinguishable noise. This can be compared to semantic zooming UIs like GoogleMaps™[5], where city names, which are visible when zoomed out, vanish when zooming in on the city and are replaced by streets and other infrastructure that was not visible before.

Design Goals

De-factorizing information space and presentation time allows creating different talks on the same topic.

In contrast to the slide format, which forces authors to directly project the content of a presentation onto a timeline, we reverse this principle in FLY and let the author project any number of timelines onto the content—thereby creating the semantic mappings. This allows the information landscape of the content to be created, revised, and modified until the author is content with the topic *before* any temporal order of presentation is assigned. Thus, the topic can be completely and faithfully reflected in the visualization first without any need for *content cutting*. Paths that represent the temporal component of a talks can then be defined over the canvas, independent of each other. Creating an information collage on one large topic, e.g., for a university course and then defining the individual paths for the respective lectures is a typical way of working with FLY as an author. In this scenario, it is easy to create recaps at the beginning of a lecture by first visiting the core concept of the last lecture, perhaps at a higher semantic zoom level. Also, including overviews and back-references comes naturally.

Changing the information canvas is consistent across all presentations.

Another advantage of this principle is that modifying a chunk of information on the canvas, for example to add an image or to fix a mistyped formula, causes all presentation paths or talks to become automatically updated at once. Managing separate versions of presentation documents for different audiences (which at some point will possibly be in different states of updated content) is no longer needed. To stay with the example of university courses, one could even imagine to aggregate the content of multiple courses

[5] maps.google.com

into one document, facilitating cross-references and excursus into related topics. Also, these documents could be co-authored between experts in different domains or disseminated after a talk to encourage self tutoring and exploration of additional material.

Changing the medium changes the user experiences and workflows for all user roles.

The goal of *Fly's* design is to leverage our proposed changes to the technical representation of presentation visuals and to help all three user groups in their tasks by giving them an interface that coincides with their conceptual model. Such a drastic shift in the way a medium is represented, however, has to be examined carefully in the light of all aspects of usage of the medium; helping the authoring of presentation visuals is meaningless if presenting the material or learning from a presentation becomes more difficult in the process. In the following, we describe how we envision the work processes for all three groups to look like in FLY.

Authoring with Fly

As explained above, the interplay of semantic structure, semantic mapping, and syntactic mapping allows to defactorize the creation of the content as a faithful visual representation of the topic from the creation of the presentation path. This clear separation changes the workflow of the author from the combined interaction model of slideware, where the semantics of the content and syntactics of the presentation order had to be considered at the same time, to a more structured and focused three-step-model. The first two steps are interchangeable, depending on if the author follows a top-down or bottom-up design strategy; the order in which the steps are presented here is more suitable for the latter way.

Information Collection and Arrangement

Authors start by creating a collage of information atoms on the canvas that reflects the topic's spatial gestalt.

First, the author collects information about the topic—everything that can be represented visually may be used: text, photos, video clips, images, or diagrams. These information atoms can directly be placed on the canvas where they can be sorted, clustered, laid out, and re-arranged any time. The order of presentation does not matter at that time, the author can focus on how the layout best represents the overall shape of the topic and how the relations between

subtopics can be expressed through the gestalt of their spatial arrangement. She is encouraged to try out variations and to experiment; the content items can be easily moved around via direct manipulation, while the big picture is always visible and allows to evaluate every local layout decision in the context of the whole. With no automatic layout, and the author thus being always in the loop, the canvas only reflects the design decisions of the author. This also facilitates orientation in the information landscape for the author and, later, for the audience [Buzan, 1991].

Providing Abstractions and Layering Where Needed

Authors then create semantic abstractions or details on the other zoom levels.

After the layout of the ground level is finished, the author can create the higher levels for the semantic zooming feature. More abstract or just visually less detailed versions of singular information atoms or whole groups and subtopics can be stacked on top of their detailed counterparts. When zooming out for overviews or summaries, the representation changes as a function of the zoom level; the details below gradually fade out, and the abstract visualizations become visible. If the author follows a top-down approach, she would probably begin by designing these overview levels first and then drill down into the detail levels by adding refined representations and visualizations.

Creating Presentation Paths

Lastly, authors can define multiple temporal structures by creating a number of different presentation paths over the canvas.

The last step that is needed is to finally design a presentation path through the three-dimensional scale space information landscape that is the result of the first two steps. The author can create such a path by adjusting the viewport by panning and zooming to show the desired part of the canvas at the desired zoom level; then, she takes a 'snapshot' of this view. A presentation path consists of a series of such snapshots and can be stepped through by the presenter like a regular presentation. In each step, the according snapshot view is shown to the audience. When presenting such a path, the transitions between these views are animated automatically—they are always smooth and continuous like a cinematographic camera flight over a real landscape and connect the two views in a meaningful, physically plausible way. Unlike previous presentation systems that employ the concept of paths [Good, 2003; Moscovich et al., 2004], in FLY the author can choose if there should be a visual representation of the path—e.g., a line or curve—visible for the audience during the presentation.

Presenting with Fly

Presenters start by selecting or modifying one of the authored presentation paths.

Presenting a FLY document allows for much flexibility and ad hoc adjustment of the planned course of a talk. In the general case, presenters will start out by choosing an appropriate path through the information landscape from a list before they start the actual presentation. The list of available paths ideally contains a selection of different durations, complexities, and focus topics; but that, of course, depends on what the author—who can be the same person as the presenter—has prepared beforehand.

When presenting, one can stay on the path, freely stray from the path for excursus, or branch off to a different path.

During the talk, the presenter can stick to the selected path, advancing through the path stops one at a time, or he can switch to another path on-the-fly if he finds that the time budget for the talk has unexpectedly changed or a different level of detail and complexity is better suited for the audience at that time. It is also possible to leave the path and interactively navigate through the semantic structure of the medium to offer additional explanations or points of view on a topic; this is especially helpful to react to questions from the audience. Overviews over the topic, a subtopic, or the progression of the talk can be given with little effort at any time by just zooming out; advancing to the next stop of the current path smoothly zooms back to the level of abstraction indicated for that stop.

The added flexibility and direct manipulation navigation ask for a more expressive input device than a clicker.

The seamless transition between the pre-sequenced navigation on a path, the selection of such paths, and the spatial direct manipulation navigation through the semantic scale space gives the presenter a flexibility that is impossible to achieve with the slide format. Utilizing these capabilities of the canvas format to their full capacity, however, may require additional training on the side of the presenter. Also, the most widespread type of presentation control devices, often dubbed ‘clickers’, falls short in offering adequate input methods for the direct manipulation parts of the navigation; a multi-touch enabled mobile device would be more fitting for controlling FLY presentation visuals [Bemtgen, 2012]. The screen on these devices can then alternatively show the presenter’s notes or provide a view on the information canvas with direct manipulation pan and zoom touch input.

Learning and Understanding with Fly

The calm viewport transitions and the semantic visual clustering help the audience too.

The reasons why the canvas-based presentation format may be beneficial for the audience are similar to why the format can help authors in their work: Increased expressiveness for semantic relations between clusters of information, better context visualization and overviews, and the tendency to exhibit less of the authoring slips that are too common in slide presentations are desirable properties of presentations from the audience's point of view. Abandoning the slide frame makes spatial and temporal fragmentation of continuous topics unnecessary, and the semantically motivated spatial arrangement can leverage the audiences spatial memory to make learning easier—this might be true especially for spatial learners [Good and Bederson, 2002]. Without slides, the visual result of a presentation can always be a calm and smooth 'flight' over the canvas without the 'jarring experience' [Moscovich et al., 2004] of flipping quickly back and forth in a slide deck.

Canvas-based presentations facilitate object consistency and cross-modal information delivery.

Since multiple connections and relations of content items can be expressed visually through the use of gestalt laws (cf. [Sternberg, 2002]), there is less need to have the presenter make them explicit verbally. This could reduce the cognitive load for the audience—shifting load from the verbal to the visual channel enables usage of a larger part of one's memory resources [Good and Bederson, 2002]—and exploits the benefits of cross-modal information delivery [Mayer and Gallini, 1990]. Also, making the presentation follow a path through a continuous space and allowing smooth transitions between overlapping viewports helps the audience, together with the spatial nature of the information representation, to track semantic connections without actively having to think about it, thanks to the perceptual phenomenon of object consistency [Walsh and Kulikowski, 1998]. All this applies especially in long presentations (e.g., lectures), where a narration has to be continuously followed and processed for information.

FLY Interface Prototype I

Lichtschlag implemented the first FLY software prototype.

For his thesis, Lichtschlag [2008] implemented a first software prototype, which realized a subset of the design aspects outlined above. The user interface was kept deliberately simple (Figure 5.14): the prototype is a document-based application, each window consisting of a toolbar (5.14.A), a zoom indicator column (5.14.B), the main canvas view (5.14.C), and a tree view containing the 'viewport path stops' for the projected timelines of all talks through the semantic structure of the current canvas (5.14.D).

Navigation on the canvas is direct-manipulation-based; presentation paths are created by demonstration.

The toolbar contains icons to create new information atoms—text- or image-based—on the canvas, which will then be inserted at the current position and abstraction level (5.14.E). The position of the item can be adjusted by simply dragging it on the canvas, the abstraction level can be changed via a toggle button on the toolbar (5.14.F). The remaining icons control how a talk timeline is projected onto the content: users can create a new timeline or 'route' through the information landscape (5.14.G), add a viewport snapshot or 'landmark' to the current route (5.14.H), or start presenting (5.14.I).

The ZUI controls give direct access to the two levels of semantic abstraction in the prototype.

The zoom indicator column (5.14.B) shows the current zoom level using a vertical slider. Three buttons next to the slider allow zooming directly to one of the two abstraction levels or to a full overview of all content on the canvas. The relative arrangements of these elements shows which zoom levels are associated with which semantic abstraction levels.

The inactive zoom level is blurred and semi-transparent.

To give authors guidance when designing the content for the two semantic zoom levels, we restricted the choice of font sizes in each level to what would be readable at the respective zoom ranges. Depending on the current zoom level, either the overview or the detail level is fully opaque and in focus and thus clearly visible—the other level is semi-transparent and blurred. This gives a notion of the other semantic level at all times, offering a sense of continuity, physical plausibility, and orientation, without distracting too much from the content currently in focus. Moreover, the introduction of a visual background on the 'ground level' helps staying oriented by always having a visual cue to judge the 'altitude over ground' of the current viewport.

Figure 5.14: Screenshot of the user interface in the first FLY software prototype.

Navigation and editing direct manipulation commands are similar to major graphics software packages.

The central canvas view is the main work area for the presentation author. Information atoms can be dropped onto it (as an alternative way to the toolbar button or the menu entry) or re-arranged via direct manipulation dragging. FLY also supports the interaction concepts known from other direct manipulation software interfaces, such as viewport panning, mouse-centered zooming via the mouse wheel or two-finger-scroll, as well as the standard keyboard editing shortcuts and commands (cut, copy, paste, etc.). Double-clicking an object allows to modify its content, e.g., changing the resource for an image or editing the text for a text box. Similar to the 'document in hand' navigation metaphor or Apple's 'natural scrolling' direction, the canvas is panned in two dimensions also by dragging. To avoid conflicts with dragging objects on the canvas, dragging the canvas is done with the right mouse button. This way of solving the disambiguation between direct manipulation of objects or direct manipulation of the context or scene is used in many software packages and is especially common in computer games.

Figure 5.15: Topic indicators in FLY. These semi-transparent directional hints can be optionally faded in to visualize the spatial context of the current position when zoomed in. Source: [Hess, 2011]

To help finding one's way at closer zoom ranges, FLY can optionally display *City Lights* indicator overlays.

Context visualization in ZUIs is known to be difficult [Pook, 2001]. Therefore, when navigating the canvas at closer zoom levels, the user can optionally have the context visualized by overlaying topic indicators that point to nearby semantic clusters (Figure 5.15), a technique similar to *City Lights* by Zellweger et al. [2003]. These signs become more opaque when the topic is closer and morph into the topic cluster itself when it enters the viewport (either because the canvas is moved or zoomed out for an overview). Alternative visualizations that have been proposed for displaying off-screen items in the context of maps on mobile devices include *Halo* [Baudisch and Rosenholtz, 2003] and *Wedge* [Gustafson et al., 2008] would likely work as well.

FLY offers extensive path editing capabilities in the right column of its UI.

The actual presentation paths or 'timelines' are hosted in the tree view inside the right column (5.14.D). Each timeline contains a list of 'path stops' or 'landmarks'. The presentation path interpolates these stops; a visual representation of the path on the canvas can be optionally displayed in the form of a semi-transparent spline curve. Path stops are created by aligning the viewport of the central view so

that it shows the desired content and then pressing the 'Add Landmark' button on the toolbar—a screen flash and camera shutter sound is given as feedback that a snapshot has been taken. The stops can be freely re-ordered, renamed, or deleted in the tree view. Also, a path can be navigated by either stepping through the list or directly clicking on any stop in the list.

Paths guide the presenter's progression over the canvas but can always be left to navigate freely.

When presenting, a path can be selected before the canvas is shown in full screen without the authoring tools and controls. The presenter can either navigate freely by means of normal spatial direct manipulation or advance to the next stop of the current path by pressing the arrow keys or via mouse clicks.

Being created for testing purposes only, the first prototype could not include videos as information atoms, did not support branching paths, and did not include presenter notes.

There are, of course, a number of limitations in this prototype, so that it can only represent parts of the whole FLY concept (cf. 5.3.4 "Design Goals" and 5.3.1 "Finding a Conceptual Model"): First of all, the software supports only a very limited selection of media types to be placed on the canvas as information atoms. Dynamic media, such as videos or animations, and incremental revealing techniques, such as building up a list or image, should be embeddable in a final implementation of FLY. Second, while this prototype allows authors to create any number of presentation paths to prepare multiple different talks on the same topic, having paths branch for on-the-fly decisions about optional material is not possible. Third, the prototype misses features that benefit the presenter during the talk, for example, presenter notes, presentation timing aids, an input device for easy wireless navigation (cf. 5.3.4 "Mobile FLY Interface Prototype"), or presenter-only visible previews of the next stop. These limitations do not prohibit the effective use of the FLY prototype in real world presentation scenarios—it has been used, for example, to present our work at CHI 2009 [Lichtschlag et al., 2009]—but definitely hinder it. The prototype, however, was mostly meant as a testbed for the canvas *authoring process* and thus only has to represent the author's experience of the FLY concept sufficiently well.

FLY Interface Prototype II

A more refined second prototype was implemented by Heß for our audience study.

A second iteration on the FLY prototype—done by Thomas Heß for his Master's Thesis [Hess, 2011]—addressed some of these limitations and incorporated some of the user feedback we received from Lichtschlag's authoring study (see below in 5.3.5 "Evaluation of the Authoring Process with FLY "). Apart from being a complete re-implementation that uses modern frameworks and APIs, this second version of FLY was designed to respect the established interface guidelines and UI patterns for OS X applications (Figure 5.16). As such, some ideas from the original prototype, for example mouse-centered zooming and panning using the right mouse button, which had caused some problems with test users, were abandoned and replaced by more conventional controls. This prototype also offers a larger number of user definable preferences for visualization and interface behavior.

More importantly, however, this new version also introduced some fundamental design changes:

Semantic abstractions are now explicitly linked to detail content.

- Instead of offering two separate semantic zoom levels that can be used to label subtopics and groups of information items, the software now explicitly supports subtopics; a subtopic can be given a label and a background color. Users can assign each information item to a subtopic, which makes the item inherit its color and label. The latter is then what is shown on higher zoom levels (Figure 5.16, top left).

Path stops are flagged if the originally visible content is later moved out of their viewport.

- Path stops now maintain an internal link to the information items that are visible in the stop's associated viewport. Should the items be re-arranged in a way that they would not be visible in this stop any more, the stop is flagged as needing revision. This makes interleaved authoring of the canvas and the paths much easier for authors.

The path stop UI visualizes more information, e.g. viewport areas.

- The path stop UI is no longer represented textually, as a list, but graphically, as a stack of (possibly branching) lines. The actual stops are depicted as circles with their radius dependent on the zoom level of the stop (Figure 5.16, bottom). Also, the aspect ratio of the viewport can be set separately for each path. This

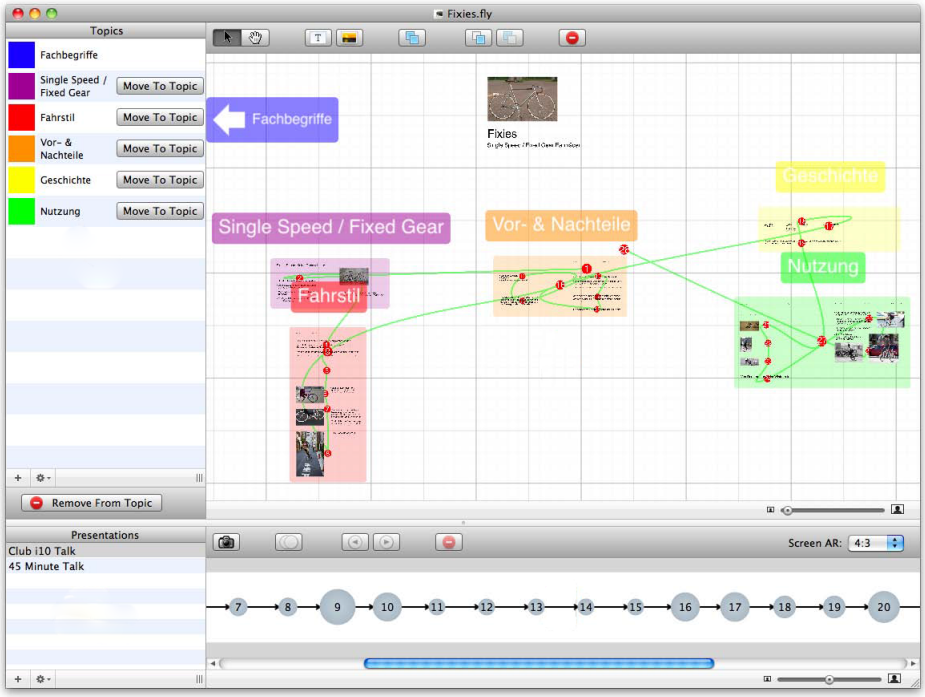

Figure 5.16: Screenshot of the second FLY software prototype. Most notably, the interface now explicitly includes topic management (left), an advanced path visualization (bottom), and is more compatible with established Mac OS X interaction patterns. Source: [Hess, 2011]

allows to easily author the same presentation for an external projector or wall display.

- Heß had planned and designed additional changes to the interface, which have not yet been implemented. Among these are support for animated builds and video, the actual ability to navigate branching paths, and an extensive list of design ideas for the presenter's screen during a presentation [Hess, 2011].

More features exist as wireframes but have not yet been included.

This second prototype was primarily created for a study that analyzes the effect of the new presentation format on the audience. Additional information on the design can be found in Heß's Master's Thesis [2011], the experiment is also described below in 5.3.5 "Understanding and Learning from Fly Presentations".

Mobile FLY Interface Prototype

To analyze the presenter's need for more expressive input, Bemtgen created a mobile version of FLY.

Our most recent FLY prototype was developed by Claude Bemtgen for his Bachelor's Thesis [Bemtgen, 2012]. Its purpose is to analyze the suitability of the canvas as the explicit representation of the semantic structure for the remaining user role: that of the presenter.

To allow the presenter to freely navigate the canvas or to select between branching path alternatives, a more expressive input device than a standard 'clicker' is required. Consequently, we decided to utilize a portable multitouch tablet: its screen can be used to display presenter notes or the canvas—or both—, while the canvas can be navigated using the standard gesture repertoire for spatial direct manipulation .

Current tablets are powerful enough to run FLY in full.

An added benefit of the platform is that current tablets, like the iPad™ used for our experiments, have enough processing power and sufficient ways to connect to external displays to make it possible to run FLY entirely on the mobile device. We are still in the process of investigating if the content creation environment of the presentation author can and should also be moved to the tablet or if it should remain centered around a classic computer.

5.3.5 Evaluate the Interface

We have to evaluate FLY with all three user roles: author, presenter, and audience.

In this fourth step of creating semantic navigation interfaces according to our interaction model 2.2.4 "Descriptive, Comparative, and Generative Power of the Combined Model", we have to evaluate if our changing the technical representation of the medium in a way that it represents the semantic structure derived from the three conceptual models of the identified user groups (cf. 5.3.1 "Finding a Conceptual Model") actually benefits all of these groups. For this analysis of the FLY idea in all three contexts—authoring, presenting, and learning—, we need different prototypes and different experiments: We will first describe the experiments that we have conducted to evaluate the implications on the *authoring process* and compare how authors work with FLY to how they use PowerPoint (cf. 5.3.4 "Authoring with

Fly"). Since the subsequent steps of presenting and learning greatly depend on the quality of the authoring process, this deserves a careful and extensive analysis. The findings will then be complemented by the results of a later case study where we have investigated how authors change their visualization and structuring strategies when they use canvas presentation systems like FLY for their real work outside a lab setting (cf. 5.3.5 "Fly Case Study—Analyzing Canvas Presentations in the Wild" below). In the second half of the section, we describe our experiments to investigate the influence of the changed presentation format on the *audience* by measuring the learning performance of students (5.3.4 "Learning and Understanding with Fly"). We have not finished our analysis of FLY's impact on the role of the *presenter*, yet. A short summary of our plans in this regard can be found in 5.4 "Conclusion".

Evaluation of the Authoring Process with FLY

In the beginning of the section, we have discussed the concept and design of canvas presentations (cf. 5.3.1 "Finding a Conceptual Model" and 5.3.4 "Designing the User Interface for FLY ") and their theoretical advantages for presentation authors. To analyze the impact of the canvas format and the associated navigation concept on the authoring process, we conducted three studies:

A basic analysis of authoring strategies was performed using paper prototypes.

The first study employs paper prototypes of both a slide-based system and FLY to collect data about how authoring strategies, resulting presentation layouts, amount and degree of communicated semantic interconnections of complex topics, and time needed for authoring differ between the two paradigms.

The results from the paper study were confirmed in a refined software prototype experiment.

The second study verifies these results by performing a similar comparison between Microsoft PowerPoint, representing the most widespread commercial slideware package, and our first FLY software prototype as described in 5.3.4 "FLY Interface Prototype I". Both studies were designed and carried out by Leonhard Lichtschlag for his Diploma Thesis [2008] under the guidance of the author, and they have been published at CHI 2009 [Lichtschlag et al., 2009]; this text is a summary of the methodology and results.

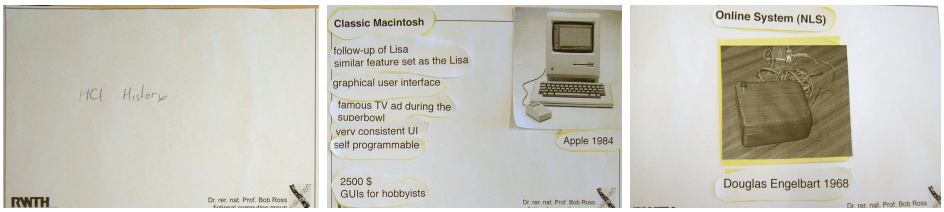

Figure 5.17: Paper prototype of a slideware presentation authoring system. The left picture shows the slide master with a centered title, the middle and right pictures show two typical finished slide layouts. Source: [Lichtschlag, 2008]

An in-the-field verification of the earlier lab-based results was conducted two years later.

The third study was conducted two years later when—due to the wide availability of *Prezi* as a public and free canvas presentation tool—we could analyze a large body of canvas presentation documents that authors had created 'in the wild', without our supervision, and sans the confounding factors always present in a lab setting. This study was a part of Thomas Heß's Master's Thesis [2011], which was co-supervised by Leonhard Lichtschlag and the author; it also has been published as a case study at CHI 2012 [Lichtschlag et al., 2012b].

Paper Prototype Study

The paper prototypes for slideware and canvas-based presentations used paper for slides or canvas and sticky notes for information atoms.

For this first study, we created paper prototypes of both a slideware system and FLY as a canvas-based presentation system. The slideware system was represented by a stack of A5 paper sheets, which already contained a pre-printed standard footer: the presenter's name and the name and logo of our group (Figure 5.17). The FLY system consisted of a large (A0) paper canvas and a rigid cardboard frame—with the presenter's name and the name and logo of our group printed on—to simulate the viewport (Figure 5.18). Holding the frame over the canvas at varying distances simulates zooming; with this simple version of the zooming feature, however, the prototype does not offer semantic zooming. For both systems, we created a large number of information atoms on sticky notes to be placed onto the slides or the canvas. Those information atoms included figures, text snippets, and bullet points; the text was scaled to resemble a font size of at least 20pt on the slides, which is a common guideline for a standard font size in presentations.

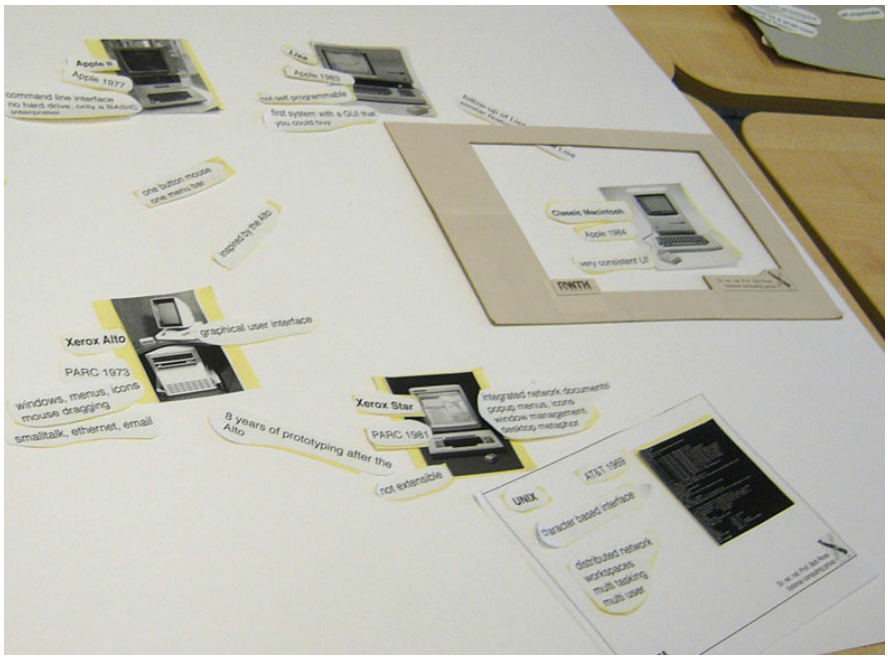

Figure 5.18: Paper prototype of FLY for the authoring study. The cardboard frame simulates the viewport that can be panned and zoomed by moving the frame or holding it at a distance from the surface. Source: [Lichtschlag, 2008]

Experiment

The participants were asked to author talks about the history of HCI.

The participants' task in this study was to create presentation visuals for two talks on the history of HCI. The two topics and the respective material were adopted from the course on *Designing Interactive Systems I* of the CS programme at *RWTH Aachen University*, held by Prof. Borchers: both dealt with the historical development of window systems and included material on Engelbart's *Online System (NLS)* [Engelbart and English, 1968], the Alto (cf. [Smith and Alexander, 1988]) and Star (cf. [Johnson et al., 1989]) systems developed at Xerox PARC, the Lisa (cf. [Birss, 1984]), Macintosh (cf. [Freiberger and Swaine, 1984]), and OS X (cf. [Singh, 2006]) systems by Apple, the early variants of the UNIX shell, a number of Microsoft Windows™(cf. [Win]) versions, and NeXTStep (cf. [Singh, 2006]) as well as a selection of applications that influenced user interface or application design (e.g., VisiCalc [Grad, 2007]), input devices (e.g., the mouse), and interaction paradigms (e.g., WYSIWYG [Myers, 1998]).

We selected two topics that were similar in terms of complexity, semantic connectedness, and size.

Both topics were selected in scope and shape to be of similar complexity; they both contained multiple semantic connections and relations. Meaningful orderings or arrangements thus included: by date, by innovation, by institution, by commercial success, by research contribution, or by inheritance. Since such a number of different layers of connections and relations are difficult to convey in a linear slide deck, these topics seemed good candidates to investigate possible benefits of the canvas-based format.

Each participant was asked to author one talk for each topic using each of the systems; the experimenter took notes and videotaped the authoring process.

At the beginning of each trial, the experimenter handed the authoring system prototype—the canvas and frame or the stack of slide templates—as well as one set of sticky information atoms—depending on the topic—to the participant. The information atom sticky notes contained content from the original course slides and were prepared by the experimenter beforehand to equalize the material collection step (cf. 5.3.4 "Authoring with Fly") between participants and keep the duration of the experiment manageable. Participants could, however, create additional items during the experiment if they wanted to. After explaining how the authoring system worked, the experimenter instructed the participants to "prepare visual aids for an upcoming talk to the best of their ability" and encouraged them to think aloud during the authoring, outlining their design decisions and how they would imagine the talk to be given later. The participants were told that the prepared visuals were explicitly not meant for later dissemination, so there was no need to include material that would not fit the scope of an oral presentation. This was important to ensure that the created visuals would not have to stand on their own but should be suitable for presentation support. Any questions the participants had during the trial regarding the topics were answered by the experimenter. Each trial was videotaped and the resulting presentation documents photographed for later analysis; the experimenter also took notes on the think aloud part of the study. After a presentation document was finished, each participant was additionally asked to fill out a questionnaire (Figure A.1 in appendix A "Questionnaires" (p. 335)) regarding their impressions of the interaction.

The experiment was a within subjects study over two conditions, FLY and *slideware*. The order of the conditions for each participant was randomized, as was the assignment of the topics to the conditions. Apart from the notes the ex-

perimenter took for the think aloud aspect of the study and the questionnaire, we measured the time it took each participant to author each presentation and determined a visualization score depending on how many relations and connections between subtopics were made explicit in the created visuals. The observed relations included:

The dependent variables were authoring time and a score reflecting the amount of semantic information made explicit in the authored presentation documents.

- *temporal relations*, e.g., in which temporal order the presented systems had been developed or published,
- *heritage*, e.g., which system was designed under a dominant influence of another system,
- *institutional relations*, e.g., groups of systems developed or sold by the same company,
- *commercial or academic success*, e.g., contrasting commercially successful and commercially unsuccessful systems.

For any instance of these implicit orderings that was incorporated visibly in the presentation document, the visualization score was increased by one point or a half point, up to a maximum total score of seven points (see Table A.1 in appendix A "Questionnaires" (p. 335)). A higher score thus indicated a more explicit visualization of the inter-subtopic connections and relations.

All participants had prior knowledge on the presentation topic.

We invited 13 participants for the experiment. Since they all needed to qualify as authors for presentation visuals in the context of *History of HCI*, we selected a number of HCI professionals and faculty as well as some graduate and undergraduate students. All participants had advanced knowledge in both topics, and four of them had held university lectures in HCI history before.

We hypothesize that canvas-based presentations can visualize more semantic relations but take more time to author.

With the assumed ability to visualize more connections and relations through the 2D arrangement, we expected the canvas visualizations to achieve higher scores. At the same time, we predicted that the unfamiliar format and authoring process would cause the participants to need more time creating the FLY presentations. The within subjects format of the study may have resulted in learning effects between the first and the second trial. Regarding the two topics, we believed, however, that they were similar in difficulty and

thus unlikely to introduce additional side effects. To formally capture these points, we formulated a number of hypotheses for the paper prototype experiment:

H1 The canvas-based presentation visuals exhibit a higher degree of connectedness between subtopics, thus reaching higher visualization scores than the slide-based presentation visuals.

H2 Authoring canvas-based presentations takes longer on average than authoring slide-based presentations.

H3 The second trial will, on average, be completed faster than the first.

H4 The presentation visuals created in the second trial will, on average, reach higher visualization scores than those created in the first trial.

H5 The individual topics are comparable in how well presentation visuals for them can be created, thus, visualization scores for the individual topics are roughly equal.

Results

The FLY documents scored higher, and no evidence for learning effects the topics differing in difficulty could be found.

The resulting visualization scores lie in the range 2–5 ($\mu = 2.85$) for slides and in the range 3–6 ($\mu = 4.62$) for FLY. A paired t-test reveals that the difference in the means of the scores ($\mu = 1.77$) across the two conditions is highly significant ($p = 0.00004$), thus allowing us to accept **H1**. We found no significant differences in the visualization scores between the two chosen topics (mean difference $\mu = 0.31$, paired t-test $p = 0.619$), which supports **H5**. Also, the visualization scores were not significantly higher (mean difference $\mu = 0.31$, paired t-test $p = 0.603$) in the second trial for each participant, rejecting **H4** and suggesting that the learning effects between the trials were small compared to the influence of the main factor (FLY vs. *slideware*).

No significant difference in authoring times could be observed.

The authoring process took slightly longer (6.37 %) in the *canvas condition* than in the *slide condition*; this difference is not significant (paired t-test, $p = 0.519$), also we found that 6 of our 13 participants were, in fact, faster with FLY. We therefore cannot accept **H2**. There was, however, an

indication of an influence of the routine, which the participants got from the first trial, on the duration of the process in the second trial; on average, the participants competed the second trial faster (mean relative difference $\mu = 12.62\%$). Although the difference is not significant (paired t-test $p = 0.081$), and we cannot accept **H4**, the influence of the learning effect on the time taken to author the visuals was stronger than the influence of the format, which is a positive result.

We used a questionnaire to get qualitative feedback from authors.

Regarding the participants' qualitative assessment of the different approaches to authoring presentation visuals, we tried to find out three key aspects in our questionnaire (cf. Figure A.1 in appendix A "Questionnaires" (p. 335)):

- Does the syntactic structuring element of the slide frame limit the author or does it provide guidance and is constructively used (questions **Q1** respective **Q2**)?
- Does the absence of any structuring element in the *canvas condition* initially inhibit—like a writer's fear of the blank page—the first authoring steps (question **Q3**)?
- Does the absence of syntactic structure lead to unorganized or chaotic arrangements (question **Q4**)?
- Do authors perceive the *slideware condition* or the *canvas condition* as a better ground for easily expressing their ideas (question **Q5**)?
- Do authors prefer using slideware or FLY (question **Q6**)?

Having evaluated the ability to explicitly visualize relations and connections between content elements in the quantitative analysis—supporting our initial arguments for breaking up the sequential linearity inherent in the navigation of slide decks—, these qualitative questions are targeted at getting insight on the authors' perception of the slide concept and its limitations and benefits for them. The questions **Q1**–**Q4** were based on 5-point Likert scales; for the last two questions, the participants could answer *Slides*, *Canvas*, or *None*.

Qualitative results were inconclusive.

The results, with the exception of **Q5**, did not reveal any statistically significant or even clearly visible trends. The questions about the slide as a fixed container for content yielded centered Likert score distributions with large standard deviations (**Q1**: $\mu = 2.54$, $\sigma = 1.20$; **Q2**: $\mu = 3.23$, $\sigma = 1.17$)—in the end, the answers seemed to be mainly influenced by the participants' personal opinion. The questions about the effects of absent syntactic structures in the *canvas condition* possibly reflect the unfamiliarity of the participants with the new format (**Q3**: $\mu = 2.23$, $\sigma = 1.30$; **Q4**: $\mu = 2.62$, $\sigma = 1.45$); the results indicate a slight tendency towards feeling lost at the sight of the empty canvas and mixed opinions on the 'messiness' of the finished visuals. Regarding the expressiveness of the two presentation paradigms, participants clearly preferred the canvas format (**Q5**: *slides*=2, *plane*=10, *none*=1). However, most did not generally prefer one over the other but stated that they would choose the format dependent on the topic to present (**Q6**: *slides*=3, *plane*=4, *none*=6).

Observations

What was more interesting, were the notes about the think aloud part of the study and the observations made by the experimenter:

In the slide condition, we saw authors cutting content or being locked in the detail trap.

In the *slide condition*, all test users tried to implement what can be described as a self-chosen set of style guidelines to keep the visual appearance of the slides consistent. Six of them were observed either revising their layout of some slides, because parts did not fit the slide frame or could not be made consistent with the style, or leaving material out of their document, because they felt that there was no space for it: both are instances of *content cutting* (cf. 5.2.1 "Content Cutting"). The typical way for slide authoring was to pick one subtopic that was deemed suitable as a starting point according to some possible ordering, e.g., the earliest of the computer systems, and then designing a slide for it. In the majority of the cases (7 participants), this slide was then considered finished and not revised—in some cases even not looked at—again during the authoring until the whole deck was completed. Material on the current subtopic that did not fit the slide any more could either 'overflow' to a new slide or was set aside and left out of the talk. Then, that process was repeated until the end. Revising and rearranging of material was only done on the slide currently

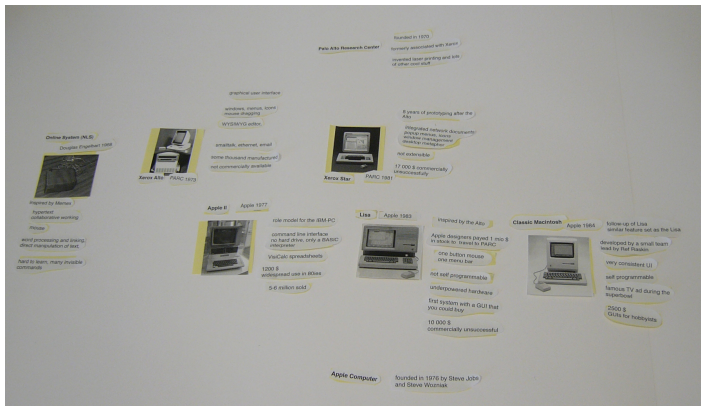

Figure 5.19: Example of a canvas layout in the FLY paper prototype study. The pictures were placed first, creating the overall semantic layout. Then, the titles and bullet points were added.

under construction; the authors exhibited the exact *Detail Trap* behavior described in 5.2.1 "Detail Trap".

In the *canvas condition*, participants typically started by sorting and clustering the material by subtopics; some did so directly on the canvas, some took up to two minutes of planning time before placing the material there. A common strategy was to start with a number of images—one for each subtopic—and arranging them in a way that their relative spatial location reflected some semantic relation between the respective subtopics. Then, other material (more images, headlines, text, figures) was grouped around these central nodes (Figure 5.19).

We also analyzed the resulting documents of presentation visuals to see if there were common structuring and layout strategies within each condition:

Slides mostly adhered to one of two structural archetypes.

In the *slide condition*, the layouts of the individual slides were mostly derivatives of one of two archetypical designs (see Figure 5.17 middle and right) from which most slides differed only marginally. Either a slide featured a picture prominently—possibly together with a heading, a caption, or both—or it was primarily a list of bullet points that was occasionally illustrated by an image. Images were often re-

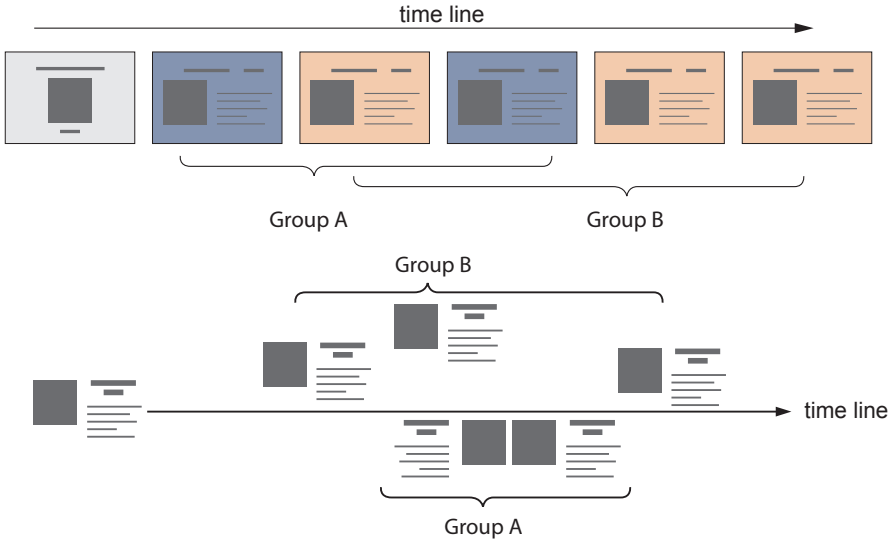

Figure 5.20: Conflict between temporal and group ordering. In the *slide condition*, participants had to decide which semantic relation they mapped to the sequential syntactic structure (top). In the canvas *condition*, such conflicts of forces could be more easily resolved (bottom). Adapted from: [Lichtschlag et al., 2009]

peated across a number of slides acting as 'icons' to graphically indicate the slides belonging to one subtopic.

Participants noticed that it was difficult to express multiple or non-linear relations.

The conflicting forces in finding a good ordering for the subtopics, which was partly provoked by our selection of highly interconnected topic clusters for the experiment, were evident to our participants and often commented on. The linear sequential structure of the slide decks required them to make a choice of either keeping groups of related subtopics together but losing any other linear ordering in the content or vice versa (Figure 5.20). Usually, they decided on one dominant linear domain, for example a timeline, along which the subtopics could be arranged, and said that they would expect other relations to be made clear verbally in the talk. Some but not all participants also created overview slides to visualize inter-topic connections (e.g., the timeline in Figure 5.21) that they felt could not be expressed otherwise. These observations together with the lower visualization scores of the slide decks support our statements on the problem of *Time Dominance* (see 5.2.1 "Time Dominance").

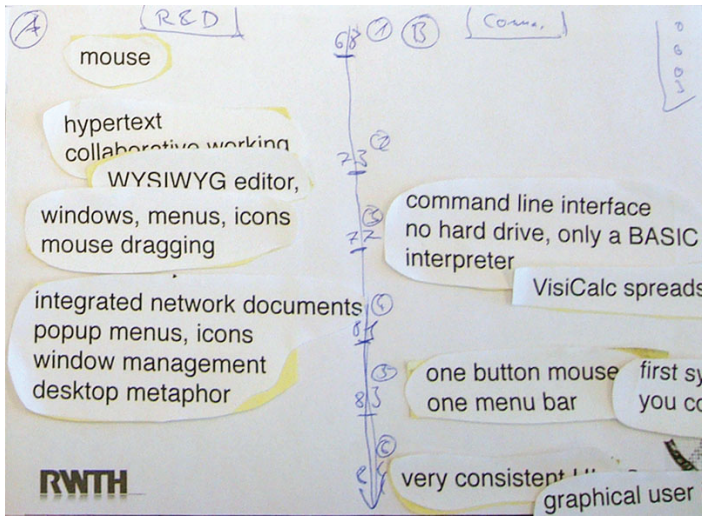

Figure 5.21: Timeline overview on a single slide. In order to resolve grouping conflicts, participants sometimes provided overviews on inter-topic relations on a single slide. Adapted from: [Lichtschlag, 2008]

In the canvas condition, participants created meaningful spatial arrangements of information.

The presentation documents from the *canvas condition* were of much greater diversity than the slide decks; they also contained more detail information, overall. Most participants made use of the affordances given by the absence of a rigid slide frame and created information clusters for each subtopic that then would be 'scanned' by more than one viewport (Figure 5.22). They also visualized comparisons between subtopics by having viewports overlap two clusters or opposing points of view. As we expected, the canvas visuals contained more relations and connections, which were often expressed through spatial configurations; the problem of conflicting forces in the ordering was thus elegantly solved using the extra dimension FLY offers. Participants often started arranging the material according to one ordering or grouping aspect and then extended and modified the layout over the second dimension to include other aspects. This strategy led to a number of very different emerging arrangements (see Figures 5.20 bottom, 5.23, and 5.24), which we all observed multiple times.

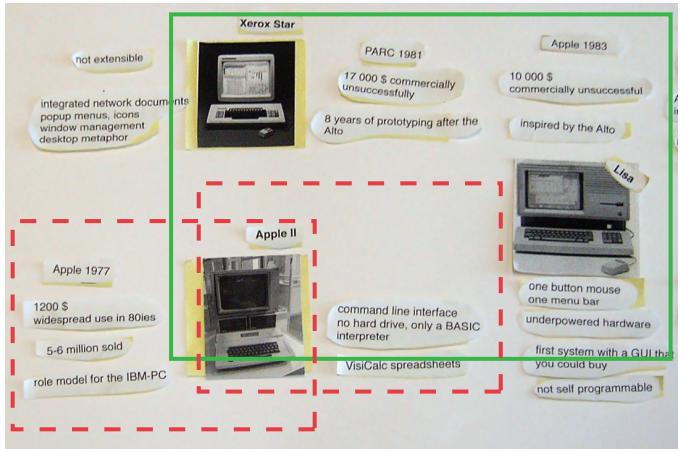

Figure 5.22: Different viewports on a section of the canvas. This participant planned an overview (green) followed by two detail views (red) that share parts of the material. Source: [Lichtschlag et al., 2009]

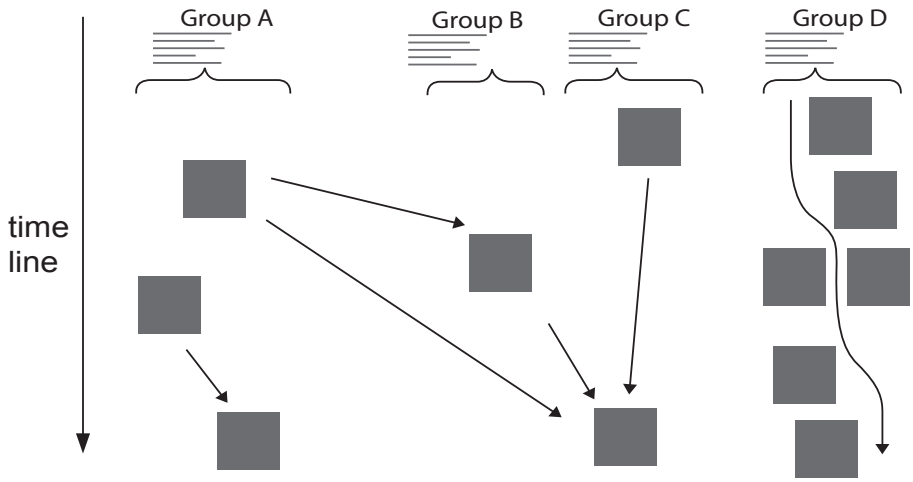

Figure 5.23: Pillar layout in the FLY condition. This layout represents time flowing vertically from top to bottom and uses arrows and the horizontal axis for other semantic relationships. Source: [Lichtschlag et al., 2009]

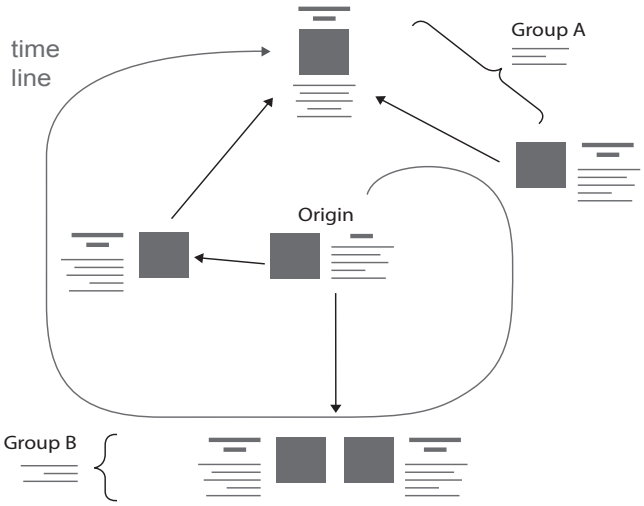

Figure 5.24: Circular layout in the FLY condition. This layout revolves around a central idea with time spiraling outwards. Other relationships are expressed through spatial proximity of the items. Source: [Lichtschlag et al., 2009]

Sometimes content would accidentally be pre-revealed, which may be a problem of the format.

An unforeseen consequence of the planar arrangement was that sometimes a viewport would, in addition to containing the items it was supposed to, overlap parts of currently unimportant, non-referenced chunks. Some participants were aware of this but stated that they did not consider it a problem if the visible unrelated snippet had already been introduced earlier in the presentation. This behavior, of course, does not occur with slides. We discuss the implications of such unrelated or pre-revealing in connection with our follow-up study in 5.3.5 "Software Prototype Study".

Users asked for a semantic zooming feature.

Additional feedback and suggestions by the participants were often concerned with the desire for a semantic zooming facility with separate abstraction levels, which the paper prototype, of course, could not offer. Also, the ability to explicitly delineate regions on the canvas, e.g., by textured or colored areas on the background, was requested. Both ideas were later included in the subsequent software prototypes (see 5.3.4 "FLY Interface Prototype I" and 5.3.4 "FLY Interface Prototype II").

Discussion

Most of our assumptions on the two formats could be confirmed.

This first paper prototype study confirmed most of our hypotheses regarding the problems of slideware and how the canvas format that is based on content aligned navigation and direct manipulation of the semantic structure can solve these problems. Overall, the FLY visualizations could represent more connections and relations of the complex topics explicitly. The different workflow allowed to keep the layout flexible longer through the process of authoring; at the same time, authors are less prone to fall into the *detail trap* and lose the connection of details with the larger context. *Time dominance* was also less of a problem with FLY, as all canvas presentation documents made use of the additional degrees of freedom to de-factor the presentation time from the semantic structures of the topics. Through the ability to define continuous content areas that could be easily shown as an overview or in detail through a sequence of partially overlapping viewports, the *content cutting* problem could be reduced: We observed no instances of discarding relevant material because of syntactic constraints, and it seemed that authors were more likely to see the content of a topic as a whole instead of a series of chunks. In the end, the FLY presentations could accommodate more material in total. A surprising result was that the authoring process in the *canvas condition* did not take significantly more time, although all participants were less familiar with the format and in some instances spent more time on planning the presentation. The clear consensus on FLY being the tool that enabled easier expression of ideas in comparison to slideware was especially encouraging.

Software Prototype Study

The results from the paper prototype study must be confirmed with a software prototype.

To verify the positive results from the paper prototype study, we developed the first FLY software prototype, which we have already introduced above (5.3.4 "FLY Interface Prototype I"). This was essential, since a software presentation authoring system is subject to a number of constraints that are not present in the paper setting: First, the cardboard canvas was always completely visible and not truncated by the boundaries of a computer screen—it was unclear how negatively this limitation of a software version would impact the authoring process and experience. Second, the paper pro-

totype was, of course, a tangible representation of the FLY system; interaction with the software would build upon the traditional desktop paradigm of single point interaction using mouse and keyboard, which can be argued to limit the authors expressiveness.

The study design was similar, the two conditions were FLY and *PowerPoint*.

For the verification study, we mostly followed the same experiment design as the paper prototype study to keep the results comparable. The two conditions for the participating presentation authors were using the FLY software prototype representing the canvas-based model or Microsoft PowerPoint 2004 for Mac representing the slideware model. The selection of PowerPoint as a control condition, of course, means that the comparison might be skewed by factors like different software quality (production level vs. prototype), participants' familiarity (most users had experience with PowerPoint, none had experience with FLY), and the vastly different feature set in the favor of the slideware. Since these points do not unfairly promote the *canvas condition*, however, they do not threaten the validity of the study.

We selected widely known topics from popular culture.

As with the paper prototype study (cf. 5.3.5 "Paper Prototype Study"), we pre-selected information material—this time in digital form—on two topics of similar size and complexity. Both topics were taken from cinematic pop culture: One revolved around the conflict of light and dark in Lucas's *Star Wars* and their representing agencies, the Jedi and the Rebellion on the one side and the Sith and the Empire on the other. The other topic was identically structured and referred to Rowling's *Harry Potter*. The material prepared by the experimenter included, like in the first study, a collection of images and text snippets. We kept the within-subjects format of the study, thus participants authored a presentation on one topic in PowerPoint and on the other topic in FLY; the order and the condition-topic assignment was randomized.

We used the same measurements as in the first study: the visualization score scheme for quantitative assessment of the expressiveness of the authoring tool and a questionnaire for qualitative analysis of the authoring experience. The questionnaire was adapted from the first study (Figure A.2 in appendix A "Questionnaires" (p. 335)).

For the second study, we recruited 18 participants: ten students, five lecturers, one engineer, one architect, and one quality assurance professional. Five of them were computer scientists, and none had participated in the first study.

Because of familiarity with PowerPoint, we expected to see less distinct results.

We predicted the overall results to be consistent with the first study—albeit less pronounced because of the unbalanced setup comparing an established presentation software with the FLY prototype. Also, we expected the resulting documents to exhibit similar layout and design strategies as we could observe in the paper study.

FLY documents also scored significantly higher in the software study.

The visualization scores for the *slide condition* lay in the range of 2–4 ($\mu = 2.78$), those for the *canvas condition* in the range of 1–7 ($\mu = 3.75$). The difference was, as we had anticipated, smaller than for the paper prototypes ($\mu = 1.97$, $\sigma = 1.96$) but still statistically significant (paired t-test, $p = 0.009$). We can thus accept **H1** also in the case of software implementations of the two presentation formats. Testing for learning effects and bias from the given topics did not reveal any significant influences from these potentially confounding factors.

Participants were more satisfied with the visuals that they had created in FLY.

When asked about how satisfied they were with the resulting documents (questions **Q7** and **Q8**), participants tended to give positive answers for both conditions. Satisfaction with their own work in the FLY condition ($\mu = 4.22$, $\sigma = 0.81$), however, was significantly higher (paired t-test, $p = 0.008$) than in the *Powerpoint condition* ($\mu = 3.67$, $\sigma = 0.91$). The questions about which tool allowed for more ease of expression and about the authors' general preferences were answered consistent with the first study: a large majority of the participants attributed better expressiveness to FLY (**Q9**: *PowerPoint*=4, FLY =10, *none*=4) and also preferred it overall (**Q10**: *PowerPoint*=3, FLY =10, *none*=5).

Participants' think aloud comments indicated that they appreciated how the canvas format helped with common authoring problems.

Again, we asked our participants to think aloud during the authoring process to help us understand their decision making and their mental model of both systems. In the *slide condition*, seven authors noted the extra work necessary to create overview slides. FLY, on the other hand, was thought by one participant to "create overviews by itself." Six participants noticed a mitigation of the *content cutting* problem when switching from PowerPoint to FLY. Most found the two semantic abstraction layers to be sufficient for the

given topics, though there are indications that more complex topics might need more abstraction layers (cf. 5.3.5 "Fly Case Study—Analyzing Canvas Presentations in the Wild" (p. 249) below). FLY helped them be more creative, said seven of the participants. All found the way how paths are defined by demonstration—through adjusting the viewport and taking a 'snapshot'—easy to understand and to perform.

Certain aspects of the UI were problematic for our users.

During the experiment, we could observe how certain aspects of the FLY UI influenced the user experience for the authors: Most of them being new to zoomable interfaces, mouse centered zooming posed a problem or was confusing for some of our users. Also, our coupling of the insertion layer (overview or detail) to the current zoom level sometimes produced mode errors. We could again observe some instances of the 'pre-revealing' of upcoming content that we first saw in the paper prototype study (cf. 5.3.5 "Paper Prototype Study"); this time, however, it was considered problematic by four of the users. They tried to circumvent the situation by widening the gaps between the affected subtopic areas, which meant that the problem had an influence on the final layout of the document—an effect we neither wanted nor anticipated and which needs further study. Interestingly, partial revealing of unrelated material that had already been covered in the talk at that time was not considered problematic by these authors.

Most of the PowerPoint presentations were strictly linear without overviews.

Similar to the procedure of the paper prototype test, we analyzed the resulting presentation documents with regard to layout, structure, and presentation sequence. From the body of PowerPoint documents, a large part (14 of 18) was structured in a strictly linear fashion, without any overviews or back-references through repetitions of material. One document was linearly structured but featured one overview slide at the beginning; the remaining three documents just clustered all content on a very small (< 3) number of slides.

We found the same document layout types from the paper prototype study and only a few sequential layouts.

From the FLY documents, in contrast, only three had a sequential structure; half (9 of 18) followed the approach of arranging the information atoms in meaningful clusters (Figure 5.25), e.g., good versus evil. Two were laid out in a way resembling the pillar design observed in the previous study (cf. 5.3.5 "Paper Prototype Study"), and two used a circu-

Figure 5.25: Typical layout in the FLY condition showing semantic spatial clusters. Source: [Lichtschlag et al., 2009]

lar layout, positioning semantic groups of items on a spiral around a main character of the movie. The remaining two canvas presentations resembled semi-structured collages, expressing semantic relationships through proximity alone and not implying any linear order of the material.

Zooming was used for creating overviews but less for semantic structuring.

While the majority of the FLY presentations (15 of 18) used the zooming feature to produce meaningful overviews at several places in the talk, only one of them made use of the explicit overview level to overlay semantically abstracted information representations. Although this feature had been asked for multiple times in the paper prototype study, in the software study, most participants seemed to prefer a strictly planar layout. Possible reasons for this include the limited scope of the topics as well as the unfamiliarity with the concept of canvas presentations and the FLY software tool itself.

The software prototype study confirms the positive effects of the canvas format on presentation authoring.

In conclusion, the software prototype study confirms the encouraging results from the paper prototype study, thus supporting our claim that the canvas presentation paradigm with its explicit representation of the semantic structure of the medium has, indeed, advantages over the syntactic-structure-based slideware for presentation authors. In both experiments, we could show that the three main problems of modeling presentation visuals after the

slide metaphor—*content cutting*, *time dominance*, and the *detail trap*—were markedly mitigated or could be entirely avoided. One has to keep in mind, however, that both experiments are subject to a number of limitations: the controlled environment of the lab, the pre-defined selection of presentation topics, and the participants' unfamiliarity with the tools are just a few examples. To make sure that our results have not been skewed by these factors, a field study with 'real world' presentation visuals is needed.

Fly Case Study—Analyzing Canvas Presentations in the Wild

Prezi became widely available and was very similar to FLY in most regards.

After the first two experiments on the effect of the canvas paradigm on authoring, which we presented above (5.3.5 "Paper Prototype Study" and 5.3.5 "Software Prototype Study"), the commercial zoomable presentation web tool *Prezi* (cf. 5.3.2 "Zoomable Presentation Interfaces") had been released. This tool implements a concept for presentation visuals that is, in large parts, congruent with our ideas for direct manipulation navigation in the semantic structure of the medium. In fact, it functions almost identical to FLY with some exceptions: First, *Prezi* is a classic ZUI and neither includes semantic zooming nor does it limit the zooming space, unlike FLY which provides a 'ground plane' to offer guidance and orientation and allows information to be placed on top of each other in different levels of semantic (and visual) abstraction. Second, *Prezi* allows, in addition to the direct manipulation zoom and pan navigation over the canvas, to also rotate the viewport around the visual axis. FLY does not support viewport rotations; when we designed the software, rotations were difficult to represent as true direct manipulation interactions. While this has changed with today's ubiquity of devices with multi-point input—where two points can define any arbitrary affine 2D transformation—we have still deliberately left rotations out, because they can impede the mental formulation of spatial information maps [Schacter and Nadel, 1991] and might therefore diminish the advantage of arranging information in map like structures instead of sequences. Third, *Prezi* supports only a single projected presentation timeline for each information landscape while FLY allows an arbitrary number of such presentation paths. *Prezi* consequently does

not offer branching paths either, which is a feature we have planned and already designed for FLY the next iteration of our current software prototype (cf. 5.3.4 "FLY Interface Prototype II").

Prezi being widely available, created an opportunity to study canvas-based presentation documents outside the lab.

With *Prezi* gaining popularity and seeing much use, a large body of presentation documents that have been authored using this web platform has become available. And with the relative similarity of the media navigation and structuring concepts of *Prezi* and FLY, there was an opportunity to analyze 'real world' presentation visuals for the emerging structure and layout strategies we could observe in the lab setting. Specifically, we were interested to see if the tendencies to visualize multiple orderings and relations of connected subtopics had manifested in regular use of canvas presentations and if the overall structure was visualized more explicitly by overviews and the spatial shape of the arrangement.

The work in this section was done mainly by Thomas Heß in the context of his Master's Thesis [Hess, 2011] under the guidance of the author and Leonhard Lichtschlag. A summary of this work has been presented by the author as a case study at CHI 2012 [Lichtschlag et al., 2012b].

We examined 50 publicly available presentation documents.

For the study, we examined 73 presentation documents that were drawn from the 'popular' section of the public online repository of *Prezi* presentations[6] on July 1st, 2010. When using a free *Prezi* account, every authored presentation is publicly accessible in this repository—for paid accounts, private documents are possible, but public is still the default setting. From the selected documents we excluded any that were clearly unfinished, not meant as visual presentation support, or were instructional presentations on how to use *Prezi*. We analyzed the remaining 50 documents with regard to four aspects—two that represent similarities between *Prezi* and FLY, and two that represent important differences:

- *Layout Strategies*: how the spatial arrangement is used to communicate the semantic structure of the topic or if the layout is governed by other forces, for example aesthetics.

[6] http://prezi.com/explore/popular/

- *Overviews*: if and how often the zooming capability of the system was used to create overviews over the current subtopic or the whole talk.
- *Scale*: how the infinite and continuous scale space of the ZUI was used to structure the material of the presentation.
- *Rotations*: how often and in which cases authors used viewport rotations.

We could observe a multitude of different canvas layouts in the field.

Every of the 50 analyzed documents exhibited a unique overall layout; this is in contrast to slide presentations where recurring patterns and established deck structures are commonly used. Consistent with our earlier observations of FLY, the authors had used viewport transitions in all three dimensions to highlight element groups of varying sizes, from showing overviews down to single information atoms.

Layout Strategies

The layouts could be clustered into three types.

We observed three primary, and fundamentally different, classes of strategies according to which the information was laid out on the canvas, which we call *Sequential Layouts*, *Structural Layouts*, and *Decorative Layouts*. The first two classes were what we expected to see as they had also been found in the FLY experiments (5.3.5 "Paper Prototype Study" and 5.3.5 "Software Prototype Study"); the third one was new.

Most documents exhibited *structural layouts*.

The set of documents that employed a *structural layout* was the largest; in 32 of the 50 presentations, the spatial arrangement and shape of the content items reflected the structure and shape of the topic. This was either achieved by clustering the information atoms to form spatially coherent *topic areas* (29 documents, see Figure 5.26 for an example), which is consistent with our observations from the lab studies, or by repeatedly nesting content using deep hierarchies to facilitate *incremental idea development* (3 documents).

In one type of structural layout, subtopics are explored top-down, one after the other.

Presentation paths in documents structured through *topic areas* typically followed a characteristic pattern: First, a topic area or cluster would be shown as an overview, then, the area would be explored in a top-down fashion. After the cluster had been exhausted, another overview—either

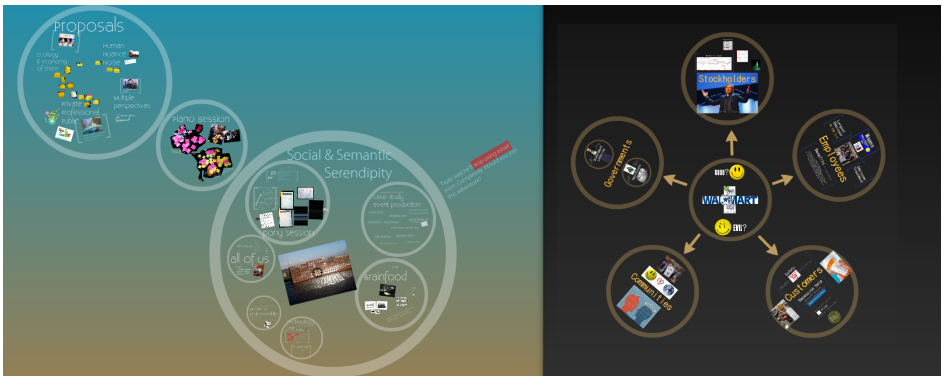

Figure 5.26: Two Prezi documents showing structural layouts of the type *topic area*. Source: [Lichtschlag et al., 2012b]

a recap of the last or a preview on the next cluster—would be shown before moving on the next topic area. This pattern was often used recursively for hierarchies of topics and subtopics.

Another type uses the zoom feature extensively to present the development of an idea incrementally.

The mechanic of the documents following the *incremental idea development* approach worked just the other way around. Those would usually start at the most detailed scale and progress by zooming out step-wise to develop a complex idea in a bottom-up way (Figure 5.27). Their structure was deeply nested, such that higher level content would be incrementally revealed by the continuous zoom-out—a strategy that is especially facilitated by ZUIs without a limited zoom range. To be discernible, content appearing later in the presentation has to be presented at larger scales, the presentations end with an overview of the whole canvas. Because of the large scale differences of current and past content, these documents use fewer overviews.

Sequential layouts resembling slide decks were the minority.

Only a small part of the overall number of documents (4 of 50) used *sequential layouts* with structures resembling that of slide decks. These presentations were restricted to a very small range of zoom levels while traversing a linear arrangement of material. Little or no overviews and little or no overlaps between viewports were also characteristic for this layout strategy. Judging from the content of these presentations it seemed that they were rather used to support strictly ordered narrations, such as telling a story, than for explaining a topic.

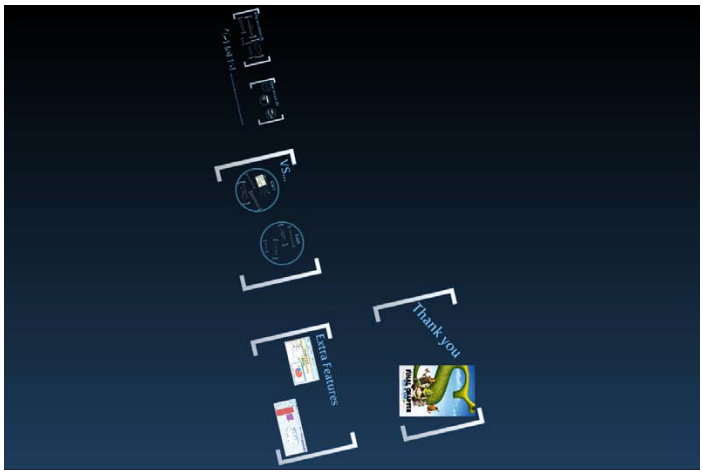

Figure 5.27: Prezi document showing a layout of type *incremental idea development*. Source: [Hess, 2011]

We did not encounter *decorative layouts* in any of our prior studies.

We could observe a new layout strategy that was used in roughly one third of the documents (14 of 50), which we called *decorative layout*. These presentations consist of a large graphic that covers the whole of the canvas with all other content elements placed on top at the same lower scale (see Figures 5.28 and 5.29). The arrangement of these content elements follows the visual shape of the background graphic into which they are embedded. This strategy produces an overall aesthetic information collage; expressing semantic relationships through the spatial placement of the individual items, however, seems more an occasional side effect than the primary concern of the authors that follow this style.

Overviews

More than two thirds of the presentation documents used zoom-outs for overviews.

Confirming our results from the FLY studies, a majority of the analyzed *Prezi* documents (34 of 50) made use of the zooming capabilities to produce overviews in their presentations. Half of those (17 of 34) created overviews for both recap of past subtopics and preview of new ones; of the rest, 16 zoomed out to preview upcoming content, and only one exclusively employed overviews for recapitulating parts of the presentation.

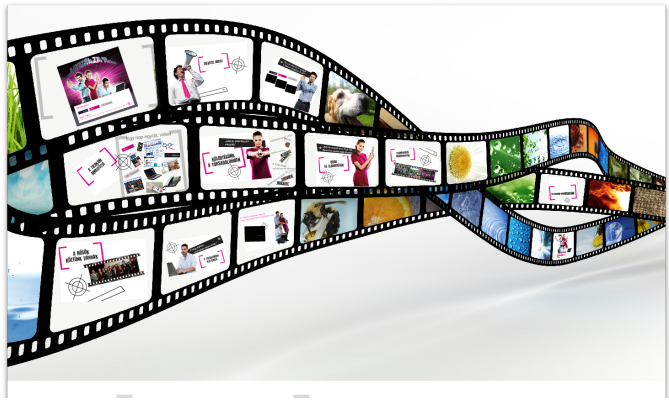

Figure 5.28: Example for a *decorative layout* as observed in our Prezi study. Source: [Lichtschlag et al., 2012b]

Figure 5.29: Example for a *decorative layout* as observed in our Prezi study. Source: [Lichtschlag et al., 2012b]

Zooming

Zooming was otherwise used for framing important information or hiding optional items.

The authors used zooming together with viewport positioning to bring the currently relevant part of the arrangement—usually, single elements or small clusters of items—into focus. Also, viewport zoom was used to emphasize important points by focusing on single words in text blocks or to illustrate details of larger graphics by zooming in on parts of diagrams or screenshots. One application of the zoom feature, which is again typical for

ZUIs, was to hide optional or marginal content (e.g., citation sources) by scaling it down to be almost invisible. Such information could then be revealed by a dramatic zoom-in.

Even with unrestricted zoom depth, most authors only used a moderate number of distinct zoom levels.

As explained above, *Prezi* differs from FLY in that it does not limit the zoom depth. Nevertheless, most of the examined documents (36 of 50) made only moderate use of this feature and placed content on at most three distinct zoom levels. The majority of the remaining ones (12 of 50) still stayed at or below six levels; only two presentations—both of which followed the *incremental idea development* layout strategy—went above six zoom levels.

Rotations

More than 60% of the documents contained viewport rotations, but in a vast majority these were only decorative.

Another difference is *Prezi's* ability to rotate the viewport along the optical axis; we found this feature being used in 32 of the 50 documents. Using rotations in a meaningful way that communicates some semantic aspect or property of the content seems to be hard: For three of the documents, which used circular layouts, it helped to underline the structure. For the remaining 29, rotation was mainly a means to achieve a decorative effect—especially in conjunction with decorative layouts where it helped to align items with the background—or provoke impressive viewport transitions. This latter case often meant that the camera would be rotated by 90° or more (Figure 5.30); if the resulting effect helps the audience to retain the context or to understand the structure is debatable (cf. [Schacter and Nadel, 1991]).

Discussion

Overall, we could observe similar document layouts as in the lab studies, which supports the generalizability of our earlier results.

The analysis of the 50 *Prezi* documents (Figure 5.31) largely confirms the results from both lab studies: The de-factoring the semantic spatial structure of the content from the syntactic temporal component of the talk gives the affordance for authors to move away from imposing linear, sequential structures onto their content and rather let the layout directly mirror their conceptual model of the topic. A majority of the examined documents were created using such *structural layouts*. However, the added expressiveness can also be (ab)used for purely aesthetic purposes, demonstrated by the relatively large number of *decorative layouts* we observed. We also found support for our claim that canvas format and navigation paradigm facilitates generating overviews—most documents included previews of upcoming material, recapitulated material already covered, or

Figure 5.30: Example for a Prezi document that makes extensive use of *viewport rotation*. Source: [Lichtschlag et al., 2012b]

both. Offering a way to explicitly model semantic abstractions, as we have in FLY, proves useful; authors of the *Prezi* documents mimicked this capability with regular ZUI features, especially so in the cases where we found *incremental idea development* layouts. Rotations, which are not supported in FLY, were mainly used for eye-catching rather than to inform about the content. Considering that spatial knowledge acquired from maps is not robust against orientation manipulation [Schacter and Nadel, 1991], and canvas presentations were designed to follow a mental model similar to that of maps, the decorative benefit of viewport rotations seems not worth the potential cost in cognitive load for the audience and the risk to lose much of the advantage of spatial memory.

Closing Remarks on the Authoring with FLY

FLY helps to avoid the three authoring problems of slideware.

Shifting the navigation domain of presentation visualizations away from the temporal progression of the talk and basing it on the semantic structure of the content is the main idea behind the FLY project. We investigated the implications this idea has for the authoring of presentation docu-

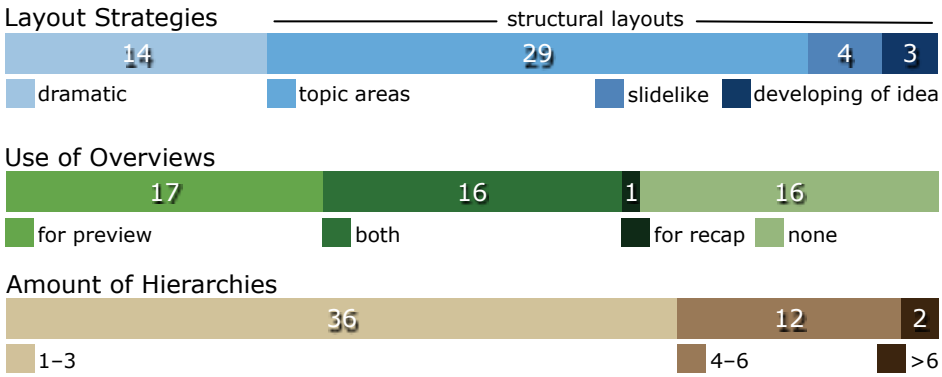

Figure 5.31: Summary of the study results. We can see that for most canvas-based presentation documents, authors make use of structural layouts, overviews, and a small number of zoom levels. Source: [Lichtschlag et al., 2012b]

ments in three independent studies with consistent results. Canvas presentation tools, such as FLY or *Prezi*, provide affordances that help authors to avoid the three common problems of slide-based presentation authoring identified in the beginning of the section: *content cutting*, *time dominance*, and the *detail trap* [Lichtschlag et al., 2009].

Canvas-based presentations relieve authors of the temporal and spatial chunking forced upon them by the slide format.

The first is addressed by eliminating the restrictions that the rigid syntactic structure of the slide frame imposes on the form of the content. The slide frame was meant to spatially partition content for presentation in discrete, sequential chunks, catering to the technical limitations of physical transparencies. With these limitations long gone, we can allow authors to design continuous structures with meaningful shape for their content. The decision whether some piece of information should be included in the talk now depends less on the available space in or visual balance of a rectangular region and more on the question if the information can be embedded into the visual gestalt of the topic. Also, overviews can evolve naturally through continuous zooming, possibly aided by carefully designed semantic abstractions, and do not have to be constructed artificially inside a slide frame.

De-factoring the temporal domain of the talk from the spatial domain of its content has proven a successful way to break the *time dominance* of slide decks. By adding an ex-

The temporal and spatial separation of presentation order and topic gestalt allows more efficient visualization and change management.

tra dimension to the design space of presentation visuals, it has become much easier to visualize semantic relations and connections between subtopics through the use of structural layouts and spatial gestalt laws. Traditional slide decks only offer temporal adjacency to express grouping, thus requiring orthogonal relations to be laboriously expressed through repetitions and temporal interleaving. FLY presentations, in contrast, facilitate visualizing multiple connections at the same time. Since the timeline is authored separately from the 'information landscape' of the canvas, it becomes trivial to include optional content or to define multiple talks on the same subject with different focuses or for different audiences. Likewise, repetitions or overviews can be included at any time without changing the content of the document. This also makes syntactic duplication of content for the purpose of authoring slightly different presentations obsolete, thereby reducing the risk of introducing inconsistencies.

Semantic zooming and a limiting ground plane help to retain the context.

The planar arrangement of information atoms and the continuous zooming capability of FLY provide context at all stages of the authoring process. This is critical in avoiding the *detail trap* and in creating visually and semantically coherent presentation documents. For the author, it is easy to step back at any time to get an overview over the 'big picture' she has designed so far—or to incorporate that overview into the talk—regardless of following a top-down or bottom-up approach.

From the three aspects of the presentation visuals' lifecycle, we have now analyzed the way authors can create presentation documents. This still leaves the process of actually giving the talk and the implications on how the audience can understand and learn from it to be investigated. While the evaluation of the first of these two remaining aspects is still in its early stages, and presenting will be thoroughly examined in the future, the remaining aspect of presentations—as a medium to transfer knowledge—is the subject of the next study.

Understanding and Learning from Fly Presentations

After having shown that the change from digital slide presentations—a format that is dominated by its syntactic structure—to canvas-based presentations—a format, which allows an explicit representation of its semantic structure—can help authors to create better presentation visuals, we now have a closer look at the role of the audience regarding this medium. As we have discussed above in 5.3.1 "Understanding, Learning, and Enjoying—Audience", the presentation visuals should help the audience to understand and learn the presented topic as well as give them guidance to refer to the contents of the talk for questions.

We need to ascertain that FLY has no negative influence on learning or understanding.

We therefore conducted a user study to find out if these assumed advantages for the audience exist to a scientifically measurable degree; at the very least we must make sure that our proposed changes in the presentation medium, and the resulting changes in the way presentations are authored, do hot have any adverse effects on the understandability and learnability of the presentation content. For this purpose, we measured *recall of facts* and *recall of the topic macrostructure* as well as the *subjective assessment* of two groups of presentation attendees under two conditions: traditional slide presentations with PowerPoint and canvas-based presentation visuals with FLY.

The influence of media on learnability is disputed.

We have already explained in 5.3.1 "Understanding, Learning, and Enjoying—Audience" that such studies are regarded as somewhat controversial: Experiments that try to evaluate how a change in medium influences learning and understanding are very difficult to design, and it has been debated extensively if these effects can be shown—or even exist—at all. Also, it is possible that people have fundamentally different learning strategies and that not all strategies may benefit from a spatialization of information as we advocate with FLY; presentation attendees who are not spatial learners might even be overburdened by the shift of information load to the visual channel. This variable, which is difficult to control for, can skew the results of such learnability studies.

Learning Study Using FLY Prototype II

We tested the retention of facts and macrostructure for participants exposed to either a FLY or a PowerPoint talk.

To be able to describe the effects of the canvas-based presentation format with respect to a baseline, we compared the learning performance of an audience after being exposed to each of two instructional talks, one given with the second FLY prototype 5.3.4 "FLY Interface Prototype II" and one with Microsoft PowerPoint as a control. Both talks were kept to a short time frame of 15 minutes to avoid fatigue, and both covered different, unrelated topics. Since learning effects between the two conditions across different topics are unlikely, we used a within-subjects study design for greater leverage. This study was designed and carried out by Thomas Heß for his Master's Thesis [Hess, 2011], the work was guided by Leonhard Lichtschlag and the author.

Based on the discussion on potential benefits of the canvas format for the audience above, we formulated a number of hypotheses [Lichtschlag et al., 2012a]:

- H1 Fact retention will be similar for both the canvas and the slide deck conditions.
- H2 Regarding recall of the macrostructure of the talk, the audience will perform better in the canvas condition than in the slide deck condition.
- H3 The canvas visuals will provide the audience with a better orientation of talk progression.
- H4 The attendees of the talks will find the structure of the canvas presentation easier to understand.
- H5 For the audience, the amount of information shown on the screen at a time will be perceived as more adequate in the canvas condition.

Participants

Our sample groups of 13 participants each were stratified according to spatial learning ability.

For the study, we recruited students from an introductory course on HCI at RWTH Aachen University. To avoid bias, students who were not native speakers (the experiment talks were given in German language) and those who had prior knowledge or experience in the presented topics were excluded from participation, leaving us with 26 students (23 male, 3 female) aged 23–35 (median 27). The participants

were divided into two balanced groups of 13, which were stratified depending on their spatial cognitive ability. We measured the latter using a card rotation test [Ekstrom et al., 1979].

Setup

Four test talks were authored by an external professional, two topics as each a FLY and a PowerPoint presentation.

We had two independent talks prepared for the experiment: one on *Fixed-Gear-Bicycles* and one on *Convergent Evolution*; the materials for both are available for download[7]. It was important to homogenize the quality of the visuals for both talks and in both conditions; ideally, both should contain the same information and exploit the respective capabilities of the formats to a maximum. With such different formats, this is a very difficult task. All four presentation documents were thus authored by an externally hired, uninvolved presentation author (age 31) who was an expert on both content domains. As he had, of course, less experience with FLY than with PowerPoint, we supported him in the process by explaining the software and answering any questions. We allowed the author to interleave the creation of the presentations with both tools, so that he could transfer visualization ideas from one format to the other if desired. This also mitigated any learning effects on the author's side that might potentially have resulted in the second documents for each topic containing better visualizations, structuring concepts, or layouts.

The results of the authoring step were as predicted by our earlier studies.

An informal analysis of the four presentation documents confirmed most of the findings of the authoring studies in 5.3.5 "Evaluation of the Authoring Process with FLY ": The FLY documents were overall more verbose and used a large number of different viewport layouts, while the PowerPoint documents mainly contained the typically small variety of common slide arrangements (e.g., bullet points plus one image). Overviews in the canvas format were spatially structured and showed context and detail in parallel, while overviews in the slide format tended to be more text-based and linear (Figure 5.32). A small number of visualization constellations were unique to FLY, as they could not be reproduced on slides; for the convergent evolution talk, for example, the historical evolution and present day geographical distribution of marsupials was shown as a viewport-spanning annotated timeline with a parallel se-

[7] hci.rwth-aachen.de/fly

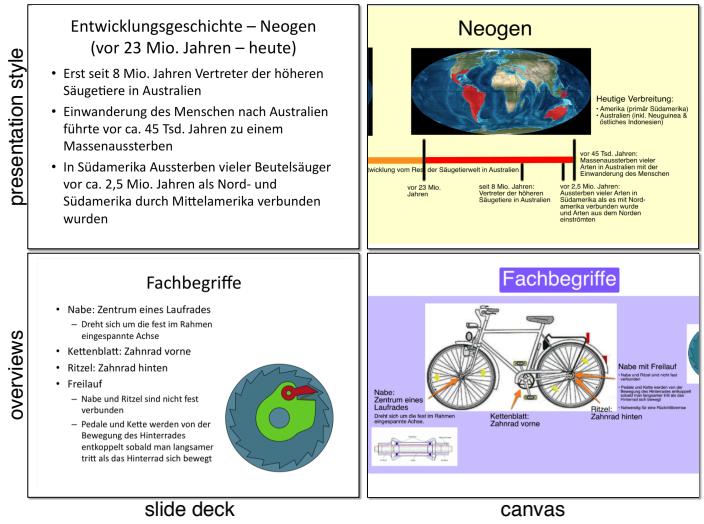

Figure 5.32: Comparison between PowerPoint slides and a FLY canvas for a subtopic from the convergent evolution talk. The slides were mostly text with an occasional image, while the canvas presentations offered a higher degree of contextual integration. Source: [Lichtschlag et al., 2012a]

ries of geographical area overviews. In PowerPoint, the same content had to be presented sequentially, interleaving textual explanations with geographical maps on separate slides each.

Both conditions shared the same pre-recorded audio material for their narratives.

For an audience, the perceived quality of a talk depends to a large extent on the performance of the speaker [Lichtschlag, 2008]. Controlling for this multivariate quantity is extremely difficult; we therefore tried to eliminate this influence altogether by playing back pre-recorded narration together with the presentation visuals for all talks. As it has been shown that learning performance is not affected by the choice of live or recorded audio in presentations [Ellis and Mathis, 1985], this is an effective measure to minimize bias by speaker performance, personal affinity, speaker-audience-interaction, and similar factors. For the voice track, we hired a professional broadcast speaker to ensure a well-pronounced and engaging narration. The style of language was chosen to be simple and informal, matching that of a normal academic presentation.

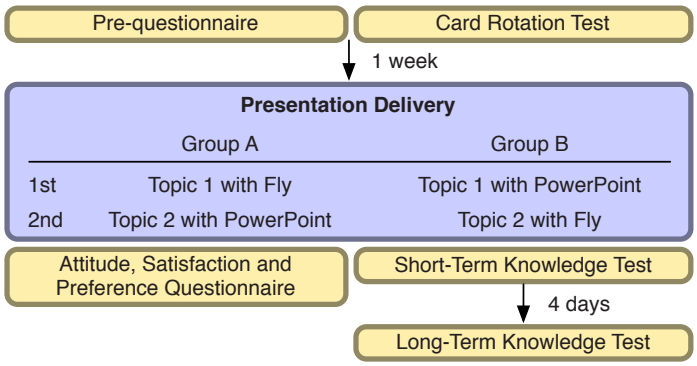

Figure 5.33: Schedule for the different parts of the FLY audience study. After the students were sorted into two groups according to their spatial learning ability, both groups were exposed to one talk of each delivery condition, *PowerPoint* and FLY. Subsequently, they were given a questionnaire and were tested for short-term fact retention. Another long-term retention test was conducted four days later. Source: [Lichtschlag et al., 2012a]

For each topic, both the slide deck and the canvas conditions shared the same audio material to ensure consistency. This way, we could control and balance the exact amount of time and detail in which any single aspect of a topic was covered by the vocal explanations. We did, however, rearrange the order of some spoken paragraphs to better fit the presentation visuals.

Procedure

The procedure of the experiment is summarized in Figure 5.33. Both groups attended one talk on one topic held with FLY and the other talk on the other topic held with PowerPoint. Learning effects between the topics were unlikely, so the order of topics was the same for both groups of participants. The assignment of presentation format to topic was counterbalanced.

Both groups had to sit short-term and long-term retention test exams.

After each talk, the participants immediately took a short-term knowledge test, filled out a questionnaire regarding their personal preferences, and could freely comment on their experience with the presentation. A second knowledge test on the topic, to asses long-term knowledge acqui-

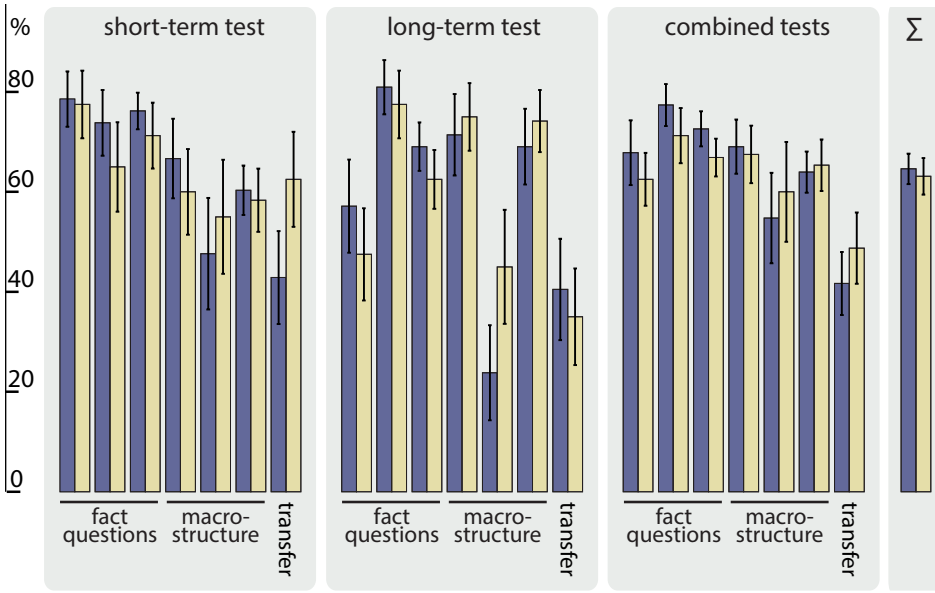

Figure 5.34: Results for the short-term and long-term fact retention and problem solving tasks, individually and combined. The chart shows the percentage of correct answers by presentation method (canvas: dark purple, slideware: light yellow, error bars ± 1 SE). Both techniques performed equally well in terms of retaining facts, structure, and transferring knowledge. Only the short-term knowledge transfer question shows a significant difference. Source: [Lichtschlag et al., 2012a]

sition, was performed four days later. Both knowledge tests included *retention* questions on content and macrostructure facts as well as *problem solving and transfer* questions to check content understanding. The questionnaires for the qualitative part of the study contained mostly Likert scale questions (see Table A.2 in appendix A "Questionnaires" (p. 335)).

Results

The results of the retention and transfer question tests are summarized in Figure 5.34.

For the *short-term* fact retention questions, the students scored consistently higher in the canvas condition. These results are, however, not statistically significant. For the questions regarding the macrostructure of the talks, no clear trend is visible. The problem-solving questions show higher

mean scores for the slides condition (paired t-test, p=0.029), but that trend is not visible in the long-term test.

For the *long-term* fact retention questions, the students scored consistently higher in the canvas condition. These results are, again, not statistically significant, nor are the results for macrostructure and transfer questions.

Comparing the results of the long-term and short-term tests, we can find a significant deterioration of the scores for group A in the slide condition (paired t-test, p=0.003). Both topics were designed to be equal in terms of content and difficulty—consequently, there were no significant differences between the mean scores for each topic in short-term, long-term, or total.

The participants' subjective assessment of the different formats showed a significant preference for FLY in questions A4 ($p = 0.047$), S5 ($p = 0.048$), S6 ($p = 0.006$), and S7 ($p = 0.001$). All other results were not significant. More details can be found in [Hess, 2011] or [Lichtschlag et al., 2012a].

Although the groups were stratified samples with regard to sex, age, and spatial or verbal learning types, we observed some interactions of group membership with other results: Members of group B scored consistently higher by a small margin in both short- and long-term questions. Members of group A found the presentation visuals significantly less distracting overall (paired t-test, p=0.043).

Regarding the spatial ability of our participants, we did not observe any interactions between this characteristic and any of the test scores or any preference for either of the presentation formats. Interestingly, however, higher spatial ability correlated with the Likert score on question S2, dealing with the perception of the amount of content being shown on screen. Spatial learners showed a significant tendency to judge the viewport as being too crowded ($\mathrm{pearson'sr} = 0.468$, $p = 0.003$).

Discussion

Overall, there was very little evidence that our proposed change of technical representation and navigation for the presentation visuals—which is primarily a change for the

The influence of the presentation format on retention and problem solving was mostly non-significant.

author and presenter, as the audience does not actively participate in the non-linear navigation—has any influence on the performance of the audience with regard to content retention. Given that we properly controlled for all other factors, which we took great care to do, we thus can accept **H1**, meaning that a change of format from PowerPoint slide decks to FLY canvases does not hurt the learnability of the content. Similar results for macrostructure understanding mean that **H2** is not supported; while people *felt* that the structure of a topic was easier to understand in FLY (S5, see below), there was no measurable evidence that this also manifested in better test scores. Either format thus seems to be equally suited to transfer information to the audience.

The FLY talks were rated significantly better in terms of orientation and ease of understanding.

When it comes to the subjective experience of the participants, however, there were significant differences. Spatial (**S6**) and temporal (**S7**) orientation were both rated significantly higher for the canvas-based talks than for the slide deck-based ones (related samples Wilcoxon signed rank tests, $p_{S6} = 0.006$, $p_{S7} = 0.001$); we thus can confirm our hypothesis **H3**.

The answers to the question if the structure of a talk was easy to understand (**S5**) were also distributed differently across the two conditions: While understanding the structure of the slide deck was perceived to be neither easy nor difficult (median = 3), participants found the structure of the canvas presentations very easy to understand (median = 1). This difference was significant (related samples Wilcoxon signed rank test, $p = 0.048$). This is interesting because, as mentioned above, we could not measure any objective improvement on the retention of the talks' macrostructures.

The participants' opinion on the amount of content being shown on screen at the same time (**S2**) was, on average, favorable for the FLY presentations. Although this supports our hypothesis in that regard, the difference was only significant to the 10% confidence level (related samples Wilcoxon signed rank test, $p = 0.058$), therefore we cannot regard **H5** as confirmed. Still, it can be viewed as an indicator that the audience's experience of a talk is influenced by the authoring problems of slideware we had identified earlier. Too much content on screen is a typical result of *content cutting* and the *detail trap* discussed above 5.2.1 "Common Problems of Slideware"; FLY presentations suffer much less from

these problems, allowing for a better balance of the amount of content in the viewport at any time.

Closing Remarks
We analyzed the learning performance of the audience for both slide deck and canvas-based presentation formats by measuring short-time retention and long-time retention of content and topic macrostructure as well as understanding of the topic through problem-solving transfer questions. Consistent with a large number of earlier studies [Hoyt, 1999; Russell, 1999; Ramage, 2002; Joy II and Garcia, 2000; Clark, 2001], we could not observe a significant effect of changing the media structure and navigation on the learning success of two groups of students. The preference of the audience, however, lies with FLY and the canvas presentation format; our participants found it significantly easier to stay oriented and understand the structure, and they preferred the amount of content shown at a time.

Differences in familiarity and the controlled test environment could have had an influence on the results.

We have to keep in mind that while the results of the experiment are encouraging, the study has a number of limitations. FLY presentations, and the canvas format in general, had been new for all of our participants, with the exceptions of a very small number of students who had known *Prezi* before the study already. The difference in familiarity between this visualization concept and the traditional slide deck may have had an influence on how the audience perceived and processed the material. Also, the participants knew that they would have to take a test after the talks; this and the related Hawthorne effect [Adair, 1984] are known risks of skewing the results of such studies. The short length of the talks, which weakens the applicability to, e.g., the case of a full time lecture, and the focus of the presentations, which were informative rather than motivational, emotional, or persuasive, are additional sources of possible confounding effects.

5.4 Conclusion

Overall, we could show that our underlying idea of offering navigation through the medium not only along the syntactic structure of its technical representation but through

its content-defined semantic structure has immense benefits for some of the relevant usage scenarios. For the medium of presentation visuals, we have extended the technical representation to directly reflect the semantic structure and gave authors the power to define multiple syntactic timelines through this structure. The new canvas-based representation of the visuals also means a change from a discrete, one-dimensional to a continuous, three-dimensional support domain, which made the established concepts of spatial direct manipulation applicable for navigation in the medium.

Authors are the user role that may be influenced most from the canvas format we introduced with FLY.

While for presentation visuals we have to respect at least three different usage aspects and, consequently, three different user roles—authors, presenters, and audience—, it is the author who we can consider the primary 'client' of the navigation interface. Authors are the users that have the strongest need for the rapid generation of design alternatives and the quick reversal of layout micro-decisions the most. They therefore can benefit best from the realignment and the direct manipulation capabilities. In the usual case, the role of the presenter is much more passive when it comes to navigation in presentation visuals—she will mostly follow the projected timeline through the content and occasionally make navigation decisions when showing optional material, answering a question, or adding extra overviews for recapitulation. The audience, finally, has no direct but only indirect influence on the navigation of the medium; it is still extremely important to investigate the effects of canvas presentations on this role, because they are the main clients of the process that is supported by the visuals.

For authors, FLY reduces some problems of slideware and offers a more flexible workflow.

So far, we have analyzed the concept of canvas presentations in the context of only the first and the third aspect of usage: that of the author and that of the audience. For authors, where the impact should be the biggest, we could demonstrate that with systems like FLY or *Prezi* some common problems that can regularly be observed in the context of traditional, slide-based presentations are mitigated or even eliminated. Visualizing complex structures of related topics comes more easily using a canvas, and the results are more powerful in expressing connections and relations between subtopics. The authoring process itself becomes more flexible, supporting both top-down and bottom-up

approaches and allowing rapid exploration of design and layout ideas. No special training is needed to reap these benefits from the changed paradigm; a sample of *Prezi* documents that were created on the web without supervision showed very similar traits to those created with FLY in the lab.

For the audience, we found only few significant effects on learning performance but a positive subjective assessment.

As mentioned above, without a verification that introducing canvas presentations does not hurt the experience of the audience, we cannot claim the concept to be an improvement—no matter the advantages for the author. In a carefully prepared study with two groups of 13 students, we could not find any significant influences on the learning performance with the exception of a negative impact on the scores of a short-term transfer questions test 5.3.5 "Learning Study Using FLY Prototype II". Although troublesome, this result could not be repeated or confirmed in the subsequent long-term test. This negative result is contrasted by a range of positive significant results regarding subjective preference of the participants.

Overall, we are confident that canvas presentations are a format that has the potential to solve many of the problems of slide deck presentations. The encouraging results we obtained from our experiments on the authoring and audience roles are confirming this. An investigation on how this presentation paradigm affects the presenter, however, still remains to be done. We plan to conduct such a study in the future using our iPad prototype of FLY (cf. 5.3.4 "Mobile FLY Interface Prototype").

Chapter 6

Non-time-based Media: Source Code

As the last of the four different kinds of media, and as an example for the static nature of non-time-based media, we discuss *source code* in this chapter. Similar to our discussion on three preceding media types, we will show that changing the interaction for navigation in the medium according to our proposed model—from the syntactic structure of source code to a content-defined semantic structure—has significant benefits for the users who work with source code.

Source code has explicitly encoded semantics in its syntactic structure.

Source code is especially interesting for media navigation studies, because it has a property that sets it apart from the other types of media we have discussed so far: Its underlying semantic structure can be algorithmically derived, because, in fact, the semantics of the code are expressed syntactically. This is necessary, of course, because otherwise, it could not be compiled and run by a computer. Source code thus is an example for the fourth class of medium when it comes to navigation along content-defined structures. In audio, video, and presentation visuals we had to construct a mapping from the semantic, content-based structure into the syntactic structure manually, through heuristics, or explicitly through the creation process of the medium (see 2 "Generating New Interfaces Using the Combined Model"); we will see that the rigid structure of source code allows au-

tomatic generation and easy reversal of this initial mapping.

The efficiency of source code navigation has great economic impact.

Apart from the interesting structural properties of source code, there are economical reasons to investigate new methods to navigate this medium: Of the total costs of any software project, up to 70% [Pressman, 2010] are spent just on software maintenance. Typical maintenance tasks comprise fixing bugs, adding features, or refactoring parts of the code to be used in other projects. Source code navigation is an integral part of these tasks: the code has to be read and understood, places for modifications in the code have to be found, and side effects of changes have to be identified and tracked through sections of the program. Consequently, impressive numbers regarding developers' time spent on—and relative costs of—code navigation have been reported: Erlikh [2000], for example, found evidence that 60–90% of the costs of software development in general involve reading and navigating source code as part of maintenance tasks. Ko et al. [2005] found that navigating code dependencies and investigating task-irrelevant code account for 35% and 46% of developers' time, respectively.

Semantic navigation could help developers to move more quickly and intuitively through code.

These figures have been published relatively recently, and they are the results of studies that use modern IDEs with the current standard syntactic navigation capabilities of file hierarchies, full-text search, and similar features. As with the other types of media described in the preceding chapters, however, offering only syntactic navigation leaves the users with the problem of continually mapping their conceptual model of the source code with respect to their tasks to the syntactic structure (cf. 2.2.3 "Combined Model" (p. 50)). Therefore, our goal is to reduce the time and cost factors introduced by standard syntactic source code navigation interfaces and offer more efficient ways by applying our concept of elevating the dominant navigation paradigm from the syntactic to the semantic structure of the medium.

We focus on the Objective-C programming language.

It should be noted that different programming languages call for different representations of source code and, consequently, may be predisposed for different navigation paradigms. To disambiguate this situation, we will restrict our investigations to statically typed, C-style languages (cf. [Kernighan, 1988]) in general and the Objective-C language in particular. The reason for this choice of language is twofold: Firstly, it is becoming an increasingly popular

language; iOS apps as well as applications for the relatively widespread OS X platform are programmed almost exclusively using Objective-C. The language follows the predominant object orientation paradigm, thus representing the—currently—most relevant class of programming languages. Secondly, Objective-C is a language with features that greatly help the aforementioned automatic deduction of code semantics. Apart from the class introspection capabilities that come with the language, excellent tools for static analysis of Objective-C source code are available as part of the standard toolchain with the LLVM[1] compiler and its Clang[2] parser frontend.

This chapter is structured as follows: After a brief recap of the relevant characteristics of the medium, we describe a selection of the known conceptual models for navigating source code and show that most of the widely used integrated development environments (IDEs) for source code lack suitable interaction techniques for navigating the semantic structures derived from these models. We then introduce STACKSPLORER, our code navigation plug-in for Apple's Xcode IDE, using the same four-step-approach 2.2.4 "Generating New Interfaces Using the Combined Model" as in the preceding chapters. With the results from our studies on STACKSPLORER, we were able to refine our understanding of the conceptual model for code navigation, leading us to develop BLAZE, a similar plug-in that follows this model more closely. We close this chapter with some suggestions for future work on code navigation.

This chapter focuses on the first generative step: finding a conceptual model for the navigation.

Describing the last of our four exemplary navigation techniques, this chapter focuses on the first generative step of finding the users' conceptual model 1 "Generating New Interfaces Using the Combined Model" of code navigation for typical code maintenance tasks. Formulating such a model needs to build upon the related research areas of code understanding, code maintenance strategies, and behavioral modeling of programmers or developers. The first step therefore consists of a survey of the related literature in these areas as well as a summary of our own formative study on developers' navigation behavior and their own perception thereof.

[1] http://llvm.org
[2] http://clang.llvm.org

EXCURSUS: OBJECTIVE-C SOURCE CODE:
Technical Representation
Objective-C source code as a purely text-based medium has perhaps the simplest technical representation of all media investigated in this thesis. The code for an application or library written in this language is contained in a number of header and implementation files, which follow relatively strict syntactic rules. The contents of these files, however, are pure text and are interpreted as such; any textual formatting, for example by means of indentation, line breaks, or comments, is purely optional and has no direct functional effect.

Syntactic Structure
The support domain of source code is two-tiered: on the top level, the code is usually organized in pairs of files—header and implementation—, and on the bottom level, the individual statements are located at certain text positions inside these files. These positions often correspond to the line structure of the text file, but this is rather a convention than a strictly enforced syntactic rule. This simplicity of the syntactic structure and the fact that there is very little connection between this structure and the location of code statements or even elements of higher level patterns—with some exceptions, e.g., that some statements are illegal in header files—also suggests that the medium is relatively old and that it should be possible to find better ways of navigating source code than through this syntactic structure.

Semantic Structures
Object-oriented source code represents the structure and relations of and between objects; examples for possible semantic structures are therefore the classes with their methods, the static class hierarchy, calling relations, or the system of roles in standard software design patterns such as Model-View-Controller (MVC) [Krasner and Pope, 1988]. For most tasks, these structures are much better suited for navigation and orientation in the source code than the standard syntactic representation.

6.1 Source Code Navigation in Standard Interfaces

Navigation through source code is important for code understanding and therefore for every maintenance task.

Successful modification of software—for the purpose of fixing bugs, adding features, or refactoring the code—involves three phases [Boehm, 1976]: the existing source code has to be understood in parts or in whole, locations for the modifications have to be identified and the modifications applied, and the changes have to be revalidated in the context of

the full code base. The first phase of code understanding[3] may be the most important of the three, because the time spent on understanding code and locating features represents a significant proportion of maintenance, debugging, and reuse of code [Storey et al., 1997; Ko et al., 2006; Revelle et al., 2010]. Also, the result of this first phase is largely responsible for the success of the other two. The most important aspect of code understanding, in turn, is the ability to navigate through the code and analyze the relationships of its different parts [Tilley et al., 1996].

Current code representations are purely file and text-based and cumbersome to navigate.

Unfortunately, the technical representation of source code is still the same as it was decades ago: code usually is distributed over simple ASCII files which can be viewed and modified using standard syntactic text editor interfaces. Navigation in such a file is mostly accomplished by full-text search or by manually scrolling through the code; this is acceptable if the task is to read every statement in every source file in order (Figure 6.1). For content-based navigation, however, syntax highlighting and commands for directly jumping to a method or function by name are often the only tools that are available.

The fragmented, linear structure of the medium facilitates a high locality of changes, which is often suboptimal.

In order to understand a piece of unfamiliar code, there clearly exist better navigation strategies than just reading each file from top to bottom—but that is exactly what may be implied by the text file representation. Similarly, the fact that usually each class has its own header and implementation file only marginally helps in tracking down bugs or finding the best location for a code change. On the contrary, the high locality of changes that is encouraged by the syntactic structure of 'text-file collection' source code often leads to poor choices of change locations [Robillard et al., 2004].

Developers quickly adopt even very simple navigation aids.

Another hint at the inadequacy of standard text file navigation for source code is that even very simple UIs that offer content-related navigation are preferred by developers over the standard interfaces. One prominent example is Code Thumbnails by DeLine et al. [2006]. It gives developers a miniature overview of the current source code file that acts like a focus-plus-context view or a graphically enriched scrollbar, thus allowing to leverage spatial memory

[3] In the related literature, both *code understanding* and *code comprehension* are used interchangeably.

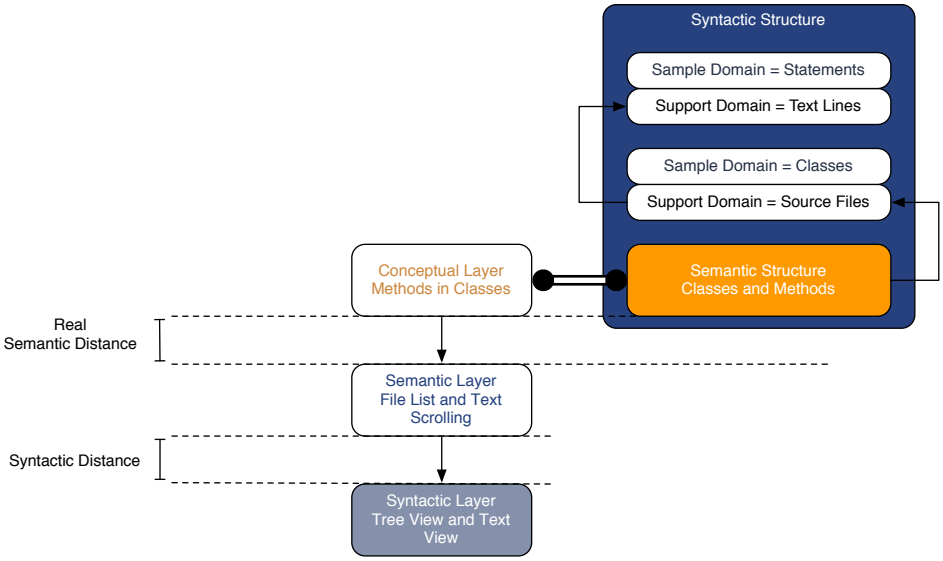

Figure 6.1: Combined navigation interface model of an IDE with standard text and file navigation capabilities. If the task is to select source files and read the statements from top to bottom, the overall syntactic and semantic distances remain small.

for quicker intra-file navigation (Figure 6.2). This concept is not unlike *Fly* (cf. 5 "Hybrid Media: Presentation Visuals" (p. 185))—although individual items or text cannot be made out, the shape of the content and its visual landmarks are used for orientation—but with one important difference: with *Fly*, this visual gestalt of the content is explicitly defined by the author and reflects its semantic structure, while *Code Thumbnails* has to rely on characteristic visual patterns of the syntactic textual representation of the source code. While this method of giving a visual overview over a piece of source code is not new—a similar system, *Seesoft* (Figure 6.3), was already proposed in 1992 by Eick [1992]—Deline et al. were the first to utilize the concept to aid in code navigation. *Code Thumbnails* also includes an additional higher level syntactic overview, the *Code Thumbnails Desktop*, which is a user arrangeable collage of code thumbnails of individual source code files (Figure 6.4). This is meant to allow quick inter-file navigation, again by recognizing the visual shape of the code.

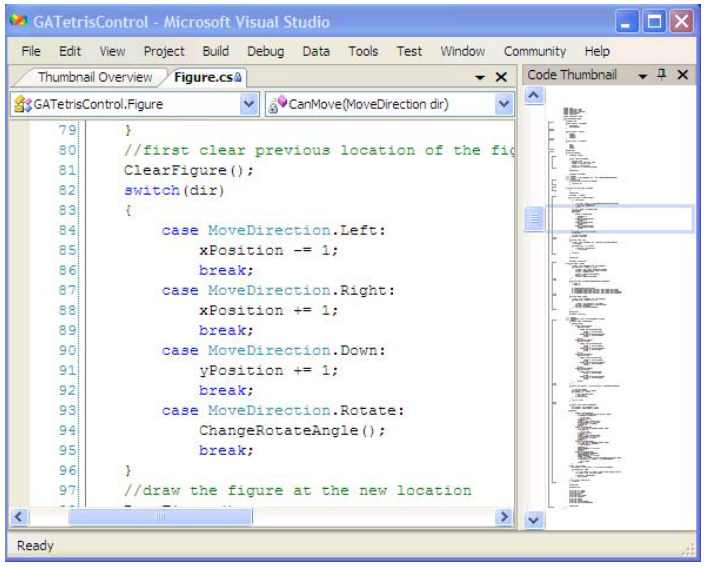

Figure 6.2: Screenshot of *Code Thumbnails*. The scrollbar of the code editor window is augmented with a miniaturized view of the actual content. Developers can thus identify specific locations inside the code through their visual gestalt and quickly navigate there. Source: [Deline et al., 2006]

Even such simple augmentation of the standard syntactic navigation in source code was much appreciated by developers in a user test. They quickly adopted the technique so that *Code Thumbnails* accounted for roughly 40–90% of all intra-file navigation [Deline et al., 2006]. The *Code Thumbnails Desktop* idea was similarly well received. These results support our claim that better alternatives to the standard text-file interfaces for source code navigation are needed.

Modern IDEs offer rudimentary semantic navigation that often lacks a consistent conceptual model or is not well integrated with the code editor.

Many modern IDEs, of course, already support some ways of semantic navigation in source code. For object-oriented languages like Java, C++, or Objective-C, it could be argued that a representation of the semantic concept of classes and methods is common. The pervasiveness of syntactic navigation, however, is again evident in a similar way as with the example of slide decks (cf. 5 “Hybrid Media: Presentation Visuals” (p. 185)): classes are commonly associated with individual files, which allows to use the traditional syntactic navigation capabilities of file lists for a crude form of se-

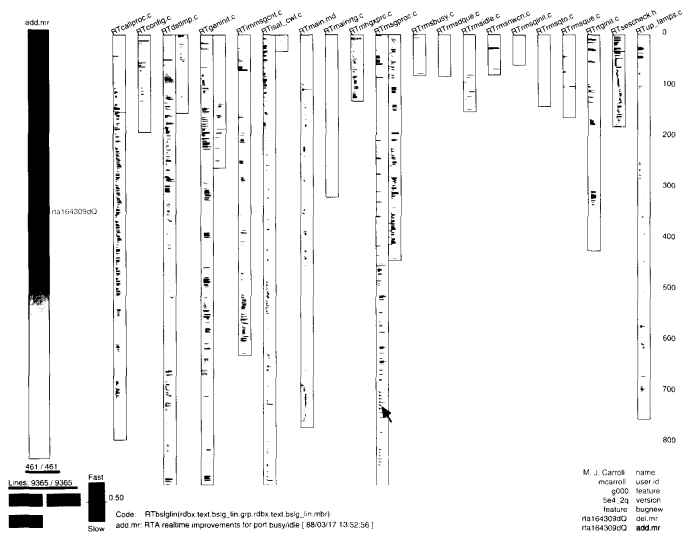

Figure 6.3: Screenshot of *Seesoft*. Each source code file is represented by a miniaturized view of its content. The individual code lines are colored according to a number of selectable filter criteria. Source: [Eick et al., 1992]

mantic navigation (Figure 6.5). Likewise, methods are associated to certain line ranges in files and often mapped to linear, table-of-contents-styled lists (Figure 6.6). Overall, in most existing IDEs, supporting navigation of the semantic structure of the code seems more like an afterthought than a central interaction concept, if it is offered at all. Class hierarchy browsers or tree views of caller relationships (Figure 6.7) are widespread but often clumsily hidden in submenus, difficult to invoke, and not well-integrated with the code editor itself. Programmers thus sometimes rely on full-text search to navigate to semantically related parts of the code, which is a rather indirect and error-prone (name ambiguities, etc.) method and requires extensive knowledge in the head about the code, its structure, and its naming schemes.

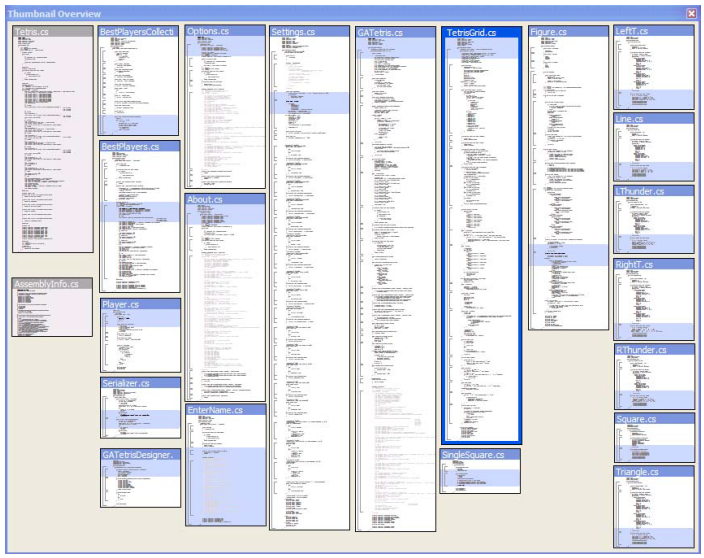

Figure 6.4: Screenshot of the *Code Thumbnails Desktop*. Similar to *Seesoft* (Figure 6.3, miniature views of all code files can be shown and arranged on a desktop. Clicking in one of the thumbnail views takes the developer to that location in the source code.) Source: [Deline et al., 2006]

6.2 Two IDE Extensions for Semantic Source Code Navigation

To create a system for semantic source code navigation, we again follow our four-step-approach as outlined in the theory chapter 2.2.4 "Generating New Interfaces Using the Combined Model", explaining the design rationale for our STACKSPLORER and BLAZE navigation tools along the way. The emphasis in this chapter lies on the first step of finding a conceptual model and deriving a suitable semantic structure from it. Similar to the other chapters, the remaining three steps are covered in less detail for the sake of brevity.

Additional information on the implementation and evaluation of both STACKSPLORER and BLAZE can be found in a number of publications, some of which also contain parts of the material presented in this text: The original idea and design of STACKSPLORER was conceived by the author and

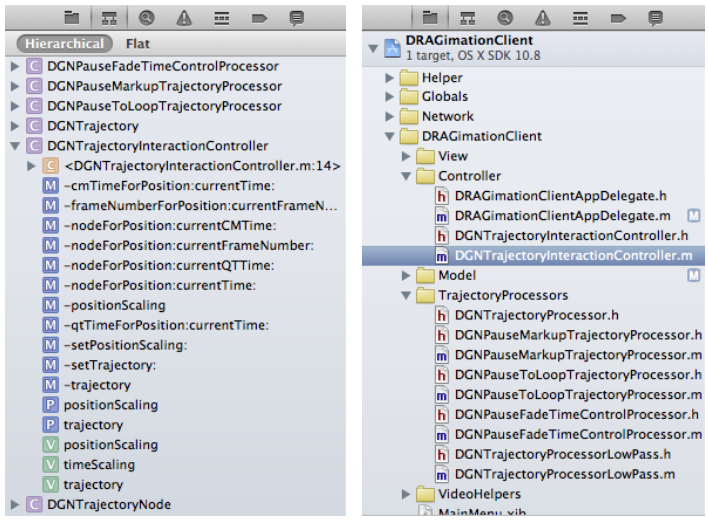

Figure 6.5: Class and file navigator views in Xcode. While the class navigator (left) allows for some kind of semantic navigation, most developers prefer the file navigator (right), because it can be better tailored to their conceptual model of the code by sorting the files into groups with descriptive names.

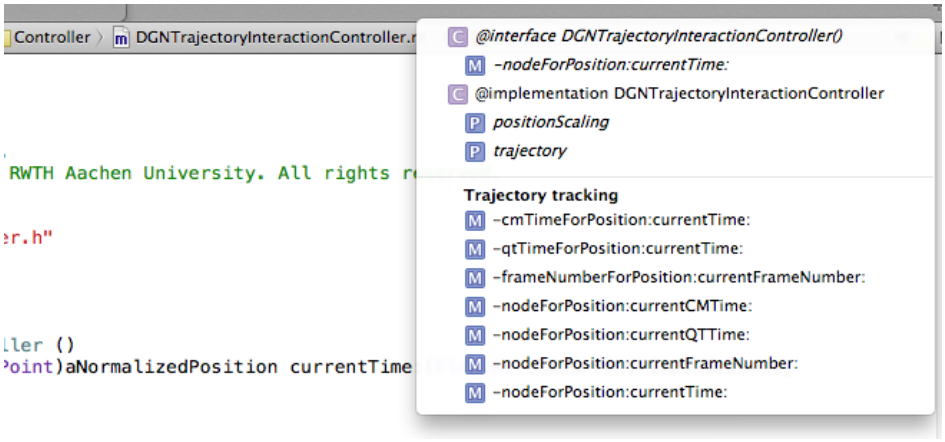

Figure 6.6: Method navigation popup in Xcode. The list is populated with the methods defined in the current file in their order of appearance in the source code. Any structuring elements, such as the section title and divider bar, have to be manually created using special `#pragma` statements.

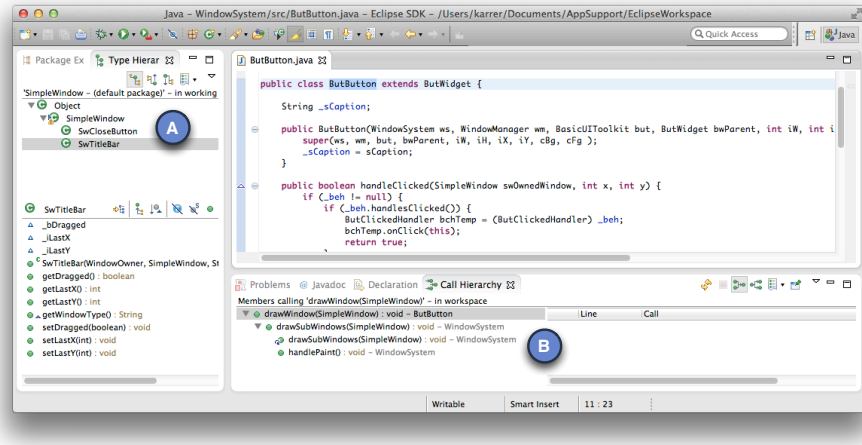

Figure 6.7: Class hierarchy browser (A) and call hierarchy tool (B) in Eclipse. Both are not contextually linked to the source code editor and have to be specifically invoked for a symbol.

then implemented and evaluated by Jan-Peter Krämer for his Diploma Thesis [Krämer, 2011] under the guidance of the author. The implementation has been presented at UIST 2010 [Krämer et al., 2010] as a poster and published together with an evaluation as a full paper at UIST 2011 [Karrer et al., 2011]. BLAZE, an adaptation of STACKSPLORER, which is focused on supporting certain navigation behaviors, was created by Joachim Kurz for his Bachelor's Thesis [Kurz, 2011] under the guidance of both Jan-Peter Krämer and the author. We published an extensive analysis of the influence of navigation interfaces in the IDE on the navigation behavior of developers at CHI 2013 [Krämer et al., 2013].

6.2.1 Finding a Conceptual Model

There exists an extensive body of related work on code understanding, conceptual models, and code navigation.

To find out which semantic structures would be ideal to align navigation possibilities with, an analysis of how developers understand, think of, and navigate through source code is necessary; we need to know which objects of interest are involved, which subspaces inside the source code contain the developers' search paths to their goals, and along which paths they move through the code when they are

trying to understand it or to apply changes. For this purpose we will first revisit some of the early related work on code understanding and conceptual models before we look at more recent work on semantic code navigation including our own formative study.

Code Understanding

Investigating how programmers understand source code has been an active area of research since the 1970s.

Scientific investigation of how programmers understand unfamiliar source code already started in the mid-70s. The goal at the time was rather to help the creation of new programming languages and to find out which intellectual skills were relevant for writing code [Shneiderman, 1976] than to inform the design of code navigation techniques and tools, and the individual results have lost some of their relevance in the context of contemporary code bases, which are much larger and use more refined and better structured languages and programming paradigms. Yet, they already developed a number of theories or *cognitive models* for how developers formulate *conceptual models* of source code.

These theories can be divided into three groups:

- *Bottom-up Model Formulation*
  Proposed by Shneiderman [1976], this theory claims that the developers' conceptual model of source code is formed by first reading it completely—an observation also made earlier by Gould [1975]—and then decoding the syntax into semantic chunks. In accordance with his *syntactic/semantic model* [Shneiderman, 1979] (cf. 2.1.4 "Division of Semantic and Syntactic Layers" (p. 29)), he proposed that these chunks are then iteratively aggregated into a smaller set of new, more abstract chunks that represents a higher semantic level. The outcome of this process, at the top of the hierarchical structure, is the conceptual model of the code.

Shneiderman and Gould say that developers read the code first, then chunk it semantically with increasing abstraction.

Programmers' conceptual models depend mainly on control and data flow abstractions.

  Pennington [1987] later supported the theory by showing in two experiments that the abstraction level of how programmers think about code increased over time. She found that the evolving conceptual models are a combination of *data-flow abstractions* on the

one hand and *functional abstractions* on the other hand. The control and data flow relations between code fragments, however, dominate programmers' conceptual models of source code rather than the functional relations that correspond to the purpose or goals of such fragments.

- *Top-down Model Formulation*
  An alternative theory as to how conceptual models of code are formed has been proposed by Brooks [1983]. According to this theory, programmers work top-down by iteratively mapping levels of problem domain knowledge to the code in decreasing order of abstraction. They start by formulating initial high-level hypotheses, which are then refined through verification and rejection of sub-hypotheses based on evidence from the source code. This evidence often comes from 'beacons', salient code features such as control structures or symbol names.

  According to Brooks, programmers start with abstract problem domain knowledge, which is then formulated as hypotheses and checked against the code.

  Soloway and Ehrlich [1984] observed that programmers used this mechanism to orient themselves in familiar code. They proposed that the conceptual model resulting from such top-down understanding is a combination of *structure* and *conventions* represented by a hierarchy of 'programming plans' and 'rules of programming discourse', respectively.

  Understanding happens top-down, combining the structure of the code with the conventions of plans and programming rules.

  A study on problem solving strategies of expert programmers by Koenemann et al. [1991] supports the top-down theory for the initial understanding of the code. They observed that when faced with a modification task, developers use code beacons to quickly classify parts of the code into those that fit their high-level hypotheses about the problem and those that do not. The former are then more closely examined, albeit in a bottom-up fashion, and the latter are mostly ignored.

  Developers have been observed to classify code top-down using beacons to verify their hypotheses.

- *Other Approaches*
  A third theory suggests that the question between bottom-up or top-down model creation may be less relevant. Several studies found developers to use both approaches depending on the code and the task [Letovsky, 1987] or that there may be a shift in weight over time between the two [Corritore and Wiedenbeck, 2001].

  Other work claims that the approach depends on the code and the task.

Another possible code understanding strategy distinction is between structured and opportunistic approaches.

They also argue that a much more fundamental distinction of different strategies to understand code is that between *structured* and *opportunistic* approaches. Where the former usually includes following the control flow of the code *prior* to planning and executing any modifications, the latter consists of localizing likely places for the change and then only understanding those [Littman et al., 1986]. These two behaviors have also been observed in other studies [Soloway et al., 1988; Robillard et al., 2004], with the structured approach mostly being more successful in solving the code modification tasks.

A better way to find a conceptual model is to actually observe developers navigating through source code.

Whatever the mechanics of formulating it, we still need to find a conceptual model that we can exploit for our goal of offering semantic navigation in source code. Singer et al. [1997] were one of the first to suggest that the knowledge of the cognitive model—the model of how and in which way a conceptual model is created [Storey et al., 1997]—does not automatically reveal a conceptual model and, hence, does not help to create better tool support for working with source code. They proposed a different approach; to derive such a model, they actually observed the navigation behavior of developers during code maintenance tasks and tried to find recurring patterns. In a long-term experiment with a large code base, they found that the work practice of the participants showed such a pattern: The developers seemed not to be able at all to build a conceptual model of the whole code base and retain it for long. Instead, they performed searches until they found a location of interest followed by an exploration phase around this location. This exploration mainly included navigation along the *semantic structural relationships* of the source code, giving a hint about a possible conceptual model.

Analyzing Navigation Behavior

This approach of observing developers during different code maintenance tasks and then mapping their navigation behavior onto different possible semantic structures to find the best fit has become an accepted and widely used tool:

Developers who navigate along semantic structural relationships are more successful.

Robillard et al. [2004] found that developers who try to understand unfamiliar source code by navigating in a structured way and along semantic structural relationships, for example inheritance or caller-callee relationships, are more successful at certain maintenance tasks than developers who just browse the code in an opportunistic way along the syntactic structure, for example scrolling through a class implementation file.

Similarly, developers tend to follow the control flow when navigating.

A similar semantic structure is suggested by the results of a study by Sim et al. [1998]. They analyzed the search behavior of developers and found that their participants followed the control flow to understand the code. Consequently, the most common search targets resolved to places where a function that was called from the location a participant was currently investigating was defined (*callee* relationship) and to places where the current function was used (*caller* relationship).

Code locations are explored through semantic relationships.

A more specific description of a similar behavior was found by Ko et al. [2005; 2006] in a study of the navigation behavior of JAVA-developers using the Eclipse IDE for code maintenance tasks. Their participants started out by collecting code fragments or locations in the source code into a mental *working set* and then mainly navigated along the semantic relationships inside the combined set. Only after they had developed an understanding of the working set, they set out to modify the code. An interesting observation was that 27% of these navigations followed a 'glance' pattern, following a structural relationship and then immediately navigating back. This could mean that the neighborhood of the semantic structure induced by structural relationships is important for developers to gather context information for potential modifications.

Navigations are often just quick, glancing referrals directly followed by a navigation back to the last place.

Questions asked by developers suggest a local conceptual model with a focus location and a semantically connected neighborhood.

More evidence for a conceptual model that consists of a set of *focus locations* together with a semantically related neighborhood has been uncovered by Sillito et al. [2008]. They investigated the questions that developers asked when tasked with making modifications to relatively large (20K SLOC and >1M SLOC[4]) code bases. The questions could be clustered into four categories that roughly outlined the developers general approach:

[4]SLOC: source lines of code

1. Find a set of focus locations to begin investigation.
2. Explore the context of these focus locations by traversing the neighborhood in star-shaped or sequential patterns.
3. Understand the structure of the induced subgraph of the call graph around each focus location.
4. Understand the relationships between these subgraphs.

The frequency of the questions in these clusters already shows that being able to navigate the immediate call graph neighborhood of a focus location is important. Also, it establishes the call graph as a promising candidate for being part of the conceptual model for many code maintenance tasks. This is further supported by the fact that the three most asked questions in the experiment were all concerned with relationships along the call graph.

Some of the semantic relations can be reduced to reachability questions.

LaToza et al. [2010a; 2010b] extended this idea by suggesting that certain questions can rather be reduced to *reachability* questions. Reachability is a much more precise concept than call graph connectedness, because the call graph does not model more dynamic methods of control flow, such as notifications, callbacks, or event posting mechanisms, which all cannot be traced by static analysis. Their paper contains a good summary of call graph limitations in this regard. To facilitate up- and downstream searches in the reachability graph, they created *REACHER*, a tool to graphically explore the answers to a wide range of reachability questions.

Parts of the navigation behavior of developers can be modeled by an information foraging process.

Rather a model for developers' navigation behavior itself than for the underlying conceptual model, Lawrance et al. [2007; 2008; 2010] and Piorkowski et al. [2011; 2012] conducted studies that support the idea that navigation in source code follows patterns known from *information foraging* [Pirolli and Card, 1995]. They argue that with modern large code bases, complete conceptual models of the source code cannot be built by the developers, anyway. Instead, they are forced into foraging strategies, evaluating possible navigation and exploration alternatives by *information scent*, the perceived likelihood of proximal cues—symbol names, for example—leading to the prey.

Following the theory of the top-down cognitive model, Lawrance et al. [2008] assumed that developers create high-level hypotheses and then look for information 'prey' to refute or confirm them. Since these hypotheses must be linguistically related to the words in the maintenance task description, for example a bug report, they computed the information scent for their *PFIS* foraging model as the word similarity between this description and the proximal cues given by method names and other identifiers. The underlying topology that connects the nodes of information or code locations was determined by the affordances of the IDE; if the IDE offered a way to get from one code location to another in one click, these locations were considered as being connected.

The foraging approaches suggest semantic relations to be a combination of call graph neighborhood and linguistic similarity.

They found that this foraging model was able to predict the aggregated behavior of 12 professional programmers better than each of the individual traces and better than both a similar model without the topology graph and a similar model without the information scent. An extensive analysis of the performance of such multi-factor models in comparison with a number of single factor models, for example *recency*, *linguistic similarity*, or *forward call depth*, performed by Piorkowski et al. [2011] produced similar results but was based on the navigation transcription of a single developer only.

Foraging models have some success in predicting actual navigation behavior.

Formative Study on Navigation Behavior

The studies mentioned above already give first directions towards finding a suitable conceptual model and semantic structure for source code navigation. Most of the related work agrees on the importance for developers to follow the control flow when trying to understand unfamiliar code, often coupled with another factor like linguistic models or recency measures. One thing to keep in mind, however, is that the majority of these experiments were conducted in a JAVA and Eclipse environment. To gain a better understanding of the behavior of Objective-C developers and to confirm that the underlying mechanics of understanding and navigating source code are similar enough to those of JAVA developers, additional studies are required.

We verify the relevance of the existing theories for Objective-C developers in our own study.

We conducted such a formative study with experienced Objective-C software developers using the Xcode IDE; this was done in the form of a contextual inquiry and a questionnaire. The study was designed and carried out by Jan-Peter Krämer for his Diploma Thesis [2010], whose work was guided by the author. Parts of the contents of this section have also been published at UIST 2011 as a full paper [Karrer et al., 2011].

We conducted a contextual inquiry with Objective-C developers working on a variety of projects.

Six participants were observed during their maintenance work on different code bases, comprising both large and smaller projects and a range of degrees of familiarity with the sources. All had studied or were studying computer science, five were male, one female, and their average age was 26.2 years.

The developers all rated their expertise with the programming language to be above average (Median = 2, 5-point Likert, 1 was best) and had at least six months of experience with Cocoa, Apple's main framework for GUI applications, and Objective-C ($M = 1.22$ years). Most of them had worked with C or similar languages before ($M = 6.92$ years of experience); all were exclusively working on Objective-C projects for OS X or iOS at the time. The tasks they were performing during the contextual inquiry covered a large part of the spectrum associated with code maintenance: three were adding features to existing applications, two were fixing bugs in larger code bases, and one was performing refactoring tasks.

Contextual Inquiry

During the observation, developers explained the reason for each navigation action.

In the contextual inquiry, we asked the developers to think aloud while performing their work and to shortly name or explain their reason each time they moved their locus of attention to a different part—at the function or method granularity—of the code. Additionally, we were allowed to take screen recordings and annotated them later to determine the characteristics of each navigation step with regards to

- the structural relation between the navigation source and its destination, e.g., from variable assignment to variable definition or from method call to the implementation of that method,

- the tool or command used to perform the navigation step, e.g., the shortcut for switching between header and implementation file or <ctrl>-clicking on a symbol to jump to its definition in the code,
- and the time taken to perform the navigation step, e.g., from the beginning of using the scroll wheel up to the moment where the destination was visible and scrolling stopped.

We clustered the navigation events into seven distinct categories.

To help clustering the different kinds of navigation the developers performed during their work, we defined two groups of six basic navigation classes that are typical for working with Objective-C code [Krämer et al., 2010]: The first group comprises the navigation types that are not based on semantic structural relationships of source code entities.

N1 Navigating to a specific location in the source code—either in the current file or elsewhere, as long as the location of the destination can be specified as and reached by a pre-planned sequence of syntactic navigation steps.

N5 Navigating between the interface part and the implementation part of classes—usually this means switching between a header file, which contains the interface of a class, and an implementation file containing its implementation.

The second group contains those types of navigation that are defined in terms of semantic structural relationships.

N2 Navigating along the call graph—either the source calls the destination part of code (through a standard function call or the invocation of a method via an Objective-C message) or vice versa.

N3 Navigating between instances of access to a variable—source and destination are both statements in which the same variable is read or written.

N4 Navigation between poster and recipient of a notification—for both explicit notification posting or

implicit notification posting via key-value-observing in Cocoa.

N6 Navigating between classes participating in one instance of the *delegation* software design pattern—apart from the relatively recently introduced possibility of closures in the language, this pattern is commonly used to replace callback mechanisms in Objective-C, so that this navigation type also applies to navigating between 'normal' C-style callback functions and their invocation locations.

We collect all other types of navigation that do not fall under any of the aforementioned classes or groups under a separate label, **N7**.

We were interested in a frequency ranking of the different kinds of semantic navigation.

From our own experiences, and from Ko's [2006] analysis of the maintenance work of Java developers, we expected to see a large number of navigation events of types **N1** and **N5** and hoped to observe a clear order between types **N2**, **N3**, **N4**, and **N6**. Apart from the fact that the participants had a general knowledge about the code they were working on and thus could build knowledge about the locations of interesting code segments, navigation that does not rely on semantic structural relationships is what is needed to explore sections of code, asking for **N1** navigations.

In Objective-C, header and implementation files need to be kept consistent according to a set of rules defined by the language. This requires manually synchronizing these files and, hence, produces a fair amount of navigations of type **N5**, almost regardless of the developer's task. If a developer needs contextual information about a certain segment of code or if a change spans multiple methods or even classes, we expected to see navigation along semantic axes through the code. This was, of course, the class of navigation events we were especially interested in to identify the most useful semantic structures along which we could design new navigation techniques.

As the observations are influenced by the affordances of the IDE, we added a questionnaire.

Such a contextual inquiry is naturally limited in what can be observed [Wixon et al., 1990]: Navigations that are not afforded by the set of tools in the IDE may occur less frequently or are mapped to other navigation types. Further inaccuracies can be anticipated because of the relatively

short time span and small number of participants, which both pose the risk of over-weighting observations of sporadic non-typical behavior. For these reasons, we decided to back up the study with an additional questionnaire, which we asked our participants to fill out after the contextual inquiry session.

Questionnaire

The questionnaire assessed how frequently the seven navigation types are used and how well they are supported by the IDE.

The web-based questionnaire was composed of five blocks of questions: demographic information and expertise assessment (**Q1–Q11**), frequency of use of the six navigation types in the context of different maintenance tasks (**Q12–Q14**), overall ranking for the navigation types independent of the task (**Q15**), asking how well the developers thought that the different navigation types are supported by the tools the Xcode IDE offers (**Q17–Q18**), and how important each of these tools was for them (**Q19–Q21**). Details and a re-print of the actual questionnaire can be found in [Krämer et al., 2010].

We expected to hear about missing tool support for the semantic navigation types.

We hoped for the results from the questionnaire to support and refine our observations from the contextual inquiry. Consequently, we expected to find an overall higher frequency of use for the syntactic navigation types and to identify a clear ordering of the importance of the different semantic navigation types. At the same time, we presumed that the existing tool support for such semantic navigation would be rated as insufficient.

Observations and Results

Developers who preferred structured navigation often relied on semantic relations to explore the code.

In the contextual inquiry, we could observe that there were two distinct strategies for how developers navigate source code: One group relied mostly on syntactic navigation using the file browser and conventional text scrolling to rapidly scan large parts of the code until they found a part they deemed important or worthwhile investigating. The other group employed a much more structured approach—sometimes even sketching a plan of approach to the problem with pen and paper before touching the code—and exploited the semantic relations between segments of the sources to a much greater extent. This observation is consistent with Robillard's [2004] descriptions of developers using either *structured* or *opportunistic* code browsing strategies. In our study, opportunistic browsing behavior was correlated with higher familiarity of the developer with the

Opportunistic navigation was employed by developers who already had a sound understanding of the code.

code at hand, which seems logical, because a good general knowledge about the structure of the code is required for this strategy to be successful. Structured browsing behavior was observed more often if the developers were less familiar with the code but also with the developers that were the most experienced with programming in Objective-C overall. We may hypothesize that this strategy can both help to incrementally build up a mental model of the code—making it a better approach for unknown or unfamiliar code bases—and allow to work more efficiently if more possibilities of semantic relationships in the language are known.

Navigation along the call graph was used to gather context information around locations of interest.

Apart from these two distinguishing global strategies, we could also observe that the use of 'advanced' navigation tools that operate along semantic relationships depended on the concrete subtask: When trying to find a place in the code to start with a task, developers used mostly basic tools, like the file navigator and scrolling. When looking for contextual information of a certain part of the code, which was believed to be relevant to the task, the participants increasingly used the *jump-to-definition* shortcut and the *project-wide search* to explore semantically connected code sections, independent of their syntactic location of files and text position inside a file. Of special interest was the observation of some users using the project-wide-search window to explore parts of the call graph (cf. 6.2.1 "Navigation along the Call Graph") starting from some method of interest.

In the questionnaire, we asked our participants how often they thought that they performed each of the six (plus one extra 'other' category) types of navigation. The results are summarized in Figure (6.8).

Syntactic navigation was still dominant, especially for known sections of code.

As expected, the developers reported to navigate between header and implementation files most often, followed by general syntactic navigation. It seems that developers are able to build solid mental models of how the parts of the code are laid out when working with familiar sources and then use syntactic navigation together with that knowledge to navigate the code. A possible reason for this could be the universal applicability of (and, consequently, the high degree of training with) syntactic navigation methods in text-based media.

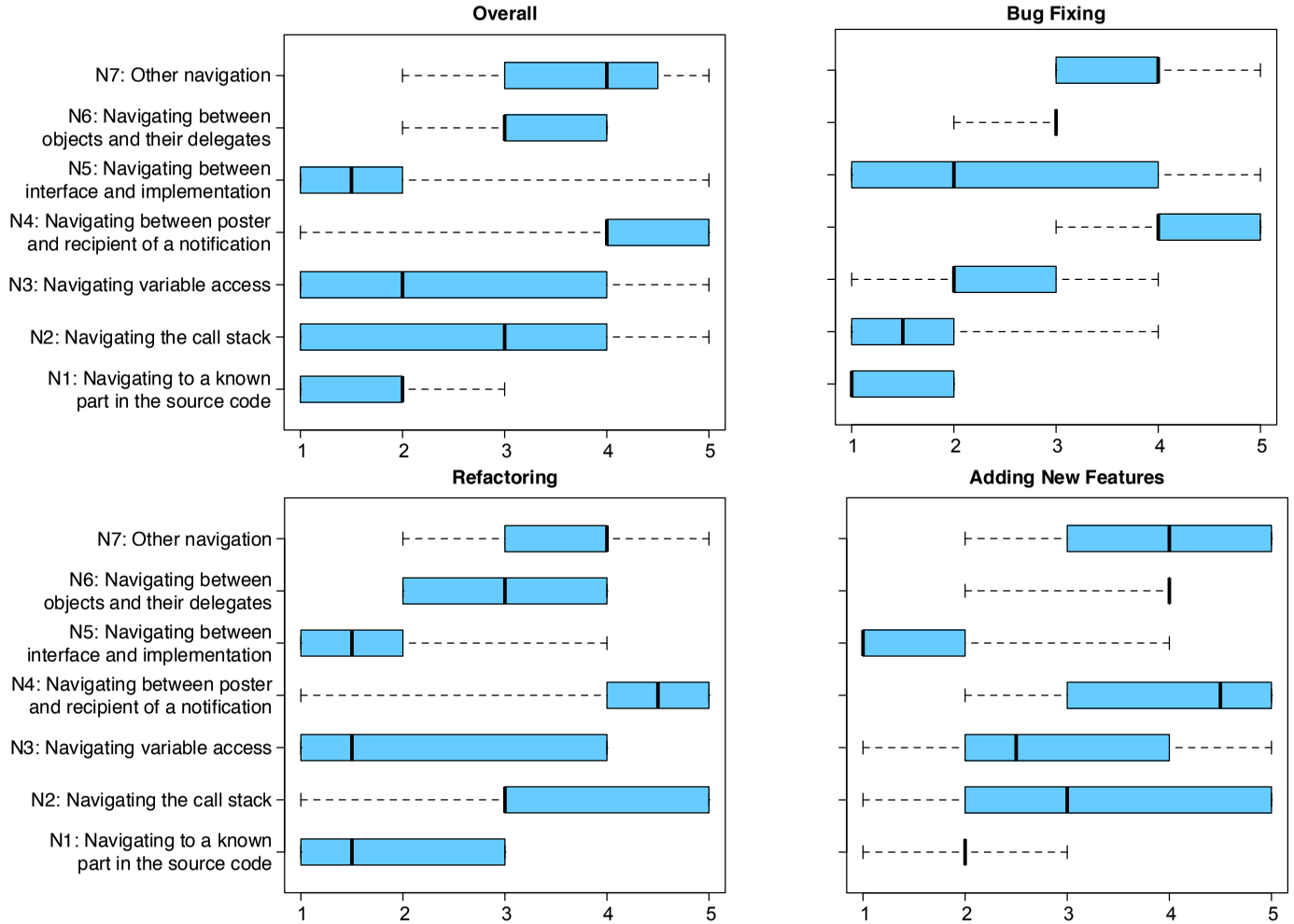

Figure 6.8: Self-assessment of our test users how often they need to perform certain types of source code navigation. "Overall" shows all of the users' responses, independent of the task. The Likert scale was always labeled with 1=frequently, 4=seldom, 5=never. Source: [Krämer et al., 2010]

Variable-based data flow and call-graph-based control flow were the most-navigated semantic relations.

Regarding the navigation types that are based on semantic relations, our participants reported that they navigated most frequently between instances of access to a variable and along caller-callee relationships. While the latter was more associated with bug-fixing tasks, the former was believed to be more important in conjunction with refactoring. Two things, however, have to be kept in mind: First, in more modern Objective-C code, instance variable access—barring global variables, the only variable access that crosses method boundaries—is almost exclusively encapsulated by accessor methods, which may even be automatically synthesized. In the current recommended style of writing Objective-C code, navigating access to these variables is thus in most cases equivalent to navigating the call graph. Second, the remaining cases of access to block local variables naturally exhibit a much higher degree of locality; in many instances, they will be visible on screen at the same time. Under these circumstances, we argue that a transient visual marking scheme to highlight the parts of the code where those variables are accessed might be more appropriate than a full-fledged navigation technique. Xcode even offers such a marking mechanism (Figure 6.9); the visual highlight, however, seems to be not prominent enough or its rendering delay too long; most of our participants had never noticed this feature.

There was agreement across all participants that call graph navigation is among the most important types of navigation.

Taking all of our participants' top-five-lists (**Q15**) and looking at the intersection, only call graph navigation remains, next to both the syntactic navigation types. Navigating the delegation design pattern or Cocoa notification was, overall, considered to be of use less often. The extra navigation category of 'others' also was believed to be invoked less frequently, even though this included navigating to graphical interface definitions (xib files) or switching to the documentation. This shows that results from such self assessments need to be taken carefully; in contrast to what users believed, we could observe them to, e.g., access the documentation much more often than they reported in the questionnaire.

Tool support for syntactic navigation was deemed best.

When being asked how well they thought that the different types of navigation were supported through the tools supplied by Xcode, the developers clearly felt that the tool set was better geared for syntactic navigation (Figure 6.10). From the navigation types that are based on semantic rela-

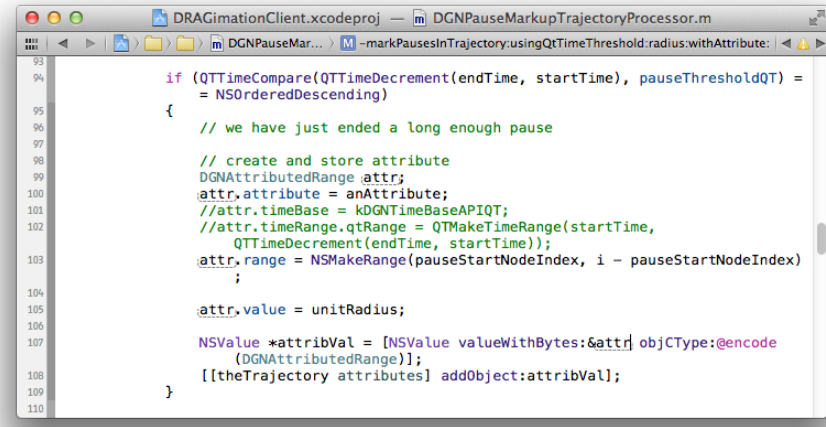

Figure 6.9: Transient highlighting of local variables in Xcode. When the caret is inside a variable name, this and all other occurrences of the variable are underlined. This also allows the name to be changed in all occurrences simultaneously.

tions, call graph navigation was still perceived as being supported best. It can be navigated, albeit only in the caller-to-callee direction, by using the jump-to-definition tool; for the other direction, navigation can only be achieved through a manual workaround using the project wide search feature. The developers' subjective ratings of the importance of different tools in Xcode mirror these findings (Figure 6.11).

Semantic call graph navigation was supported well in one direction only.

The questionnaire also contained free form fields (**Q18** and **Q21**) to collect suggestions for improvements regarding code navigation in Xcode. Most notably, the participants mentioned that they would find it beneficial if they had a way to retain the current work context when exploring related sections of code and a clear way to get back from where they started the exploration. A second common request was to have graphical overviews of semantic structures including the call graph, notification channels, and delegate relationships.

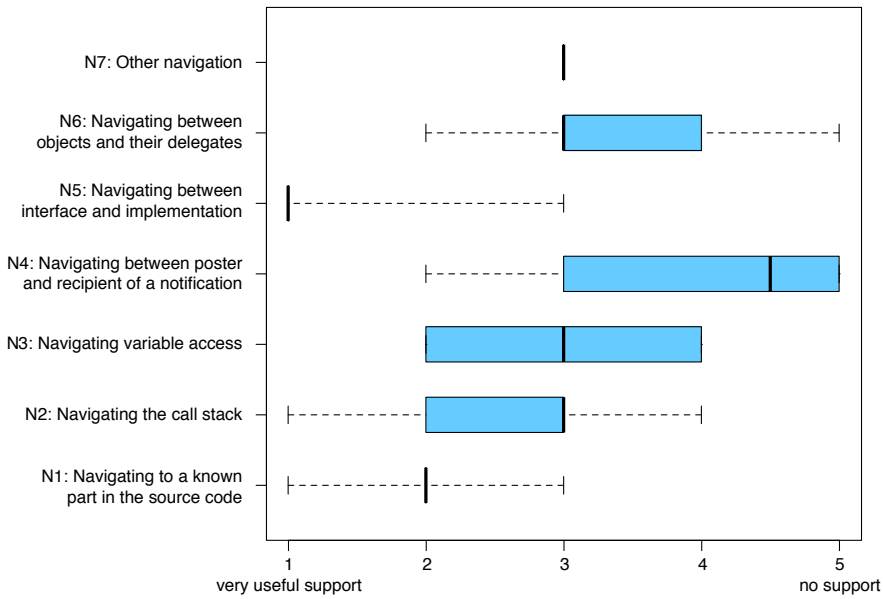

Figure 6.10: Users' assessment of the level of tool support in Xcode for the different types of navigation. Source: [Krämer, 2011]

Navigation along the Call Graph

The call graph, or a substructure thereof, is a promising candidate for a semantic structure of source code.

In our formative study to find a suitable semantic subspace for source code navigation, we could observe that a large part of the navigation events where the source and destination have a semantic structural relationship happen along the edges of the call graph. This result is consistent with the majority of the navigation studies outlined above and supports the findings by Ko et al. [2006], who showed the importance of call graph navigation for Java developers. Also, this kind of navigation appears to be the most important type when context information is required while working on a certain method, and it has only partial tool support in Xcode through the jump-to-definition tool. When we accept the call graph as the semantic structure, we can immediately see that standard text navigation methods—as they are offered by virtually every IDE—and even bookmark-style navigation helpers, like method lists of the currently open file (cf. Figure 6.6), again suffer from a large overall se-

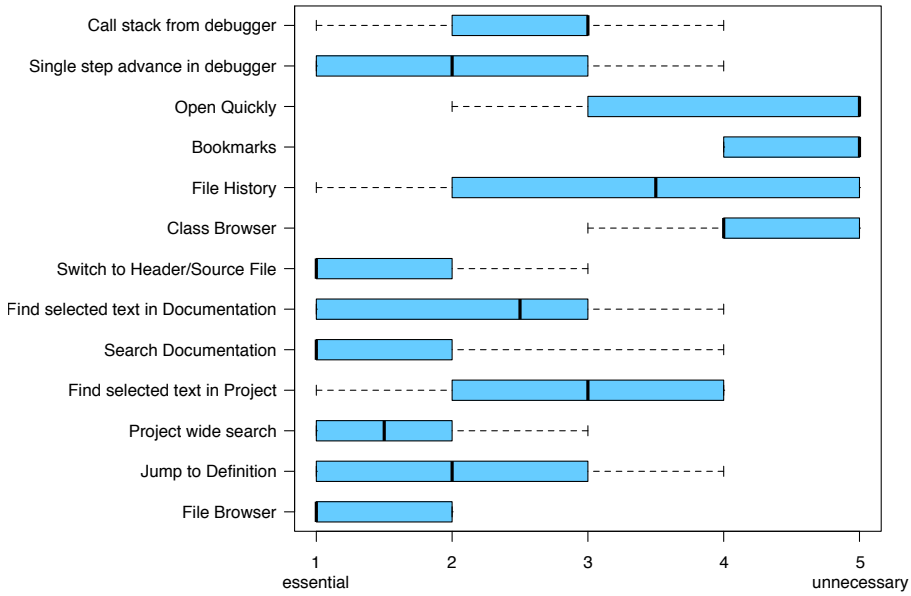

Figure 6.11: Users' rating of the importance of different tools in Xcode. Source: [Krämer, 2011]

mantic distance (Figures 6.12 and 6.13). Similar to the other media types in the preceding chapters, the tools for navigation can thus be potentially improved by directly offering semantic navigation techniques: in our case, this would be a direct access method to the call graph or parts thereof.

More complex semantic structures have been proposed but may be too complicated to be a good conceptual model.

Of course, there are studies that advocate more complex structures be used for the semantic structure. The work on code navigation as an information foraging process [Lawrance et al., 2008, 2010; Piorkowski et al., 2011], for example, suggest that some form of information scent should be used to overlay a measure of preference on the call graph topology. And Chen et al. [2000] propose that feature location in source code can be performed using the *abstract system dependency graph*, a graph-based structure that not only contains the calling relationships between methods and procedures but also the data flow inside of these entities together with additional levels of abstraction. While these approaches may be able to mathematically predict developers' current navigation behavior better than using only an

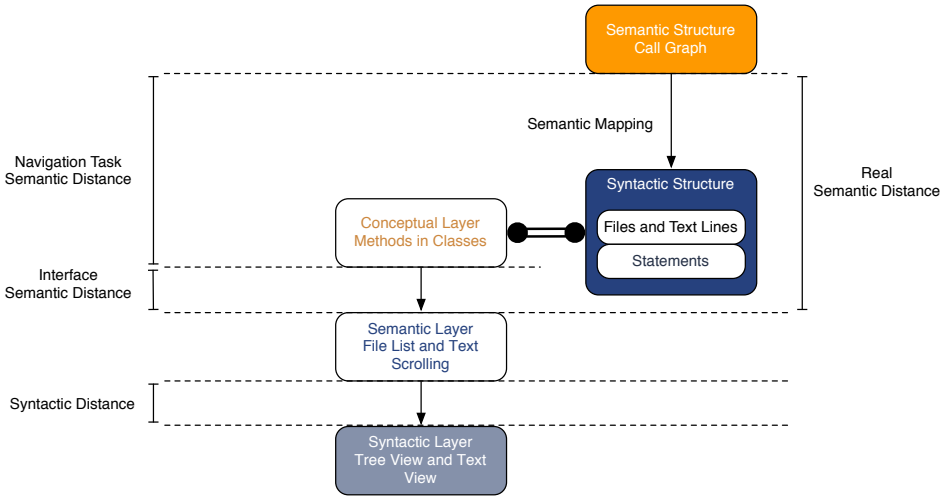

Figure 6.12: Combined navigation interface model of an IDE with standard text and file navigation capabilities. To navigate along the call graph, a user would have to find the caller or callee method by selecting the appropriate file or class from a list and then finding the method through scrolling, search, or a method list pop-up.

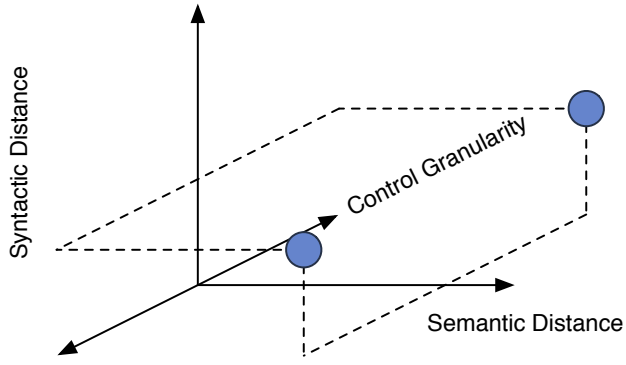

Figure 6.13: Design space showing semantic call graph navigation with standard IDE file and line browsing tools. Apart from the large semantic distance, the control granularity is either too coarse (file-based) or too fine (line-based).

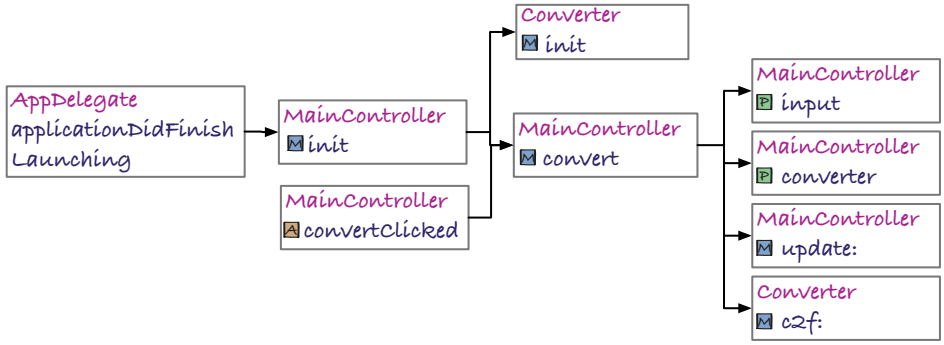

Figure 6.14: Call graph of the source code for a very simple currency converter application. The method `convert`, for example, is a *callee* of `init` and a *caller* of `udate:`.

unscented call graph [Piorkowski et al., 2011; Krämer et al., 2013], none of them represents a conceptual model that is transparent and understandable to the user. Our strategy, however, is to offer navigation along semantic structures that are derived from the way developers think about source code, and that they can grasp easily as a consequence. Thus, we decided to design a semantic navigation technique for source code along the semantic structure that is defined by the call graph of the program.

In this context, we formalize the call graph as the directed graph

$$G = (V, E_u, E_d)$$

with V being the set of all functions and methods in the source code, and $E_u, E_d \subseteq V \times V$ being the sets of 'upgraph' and 'downgraph' edges, where

$$\begin{aligned} & (A, B) \in E_d \\ \iff\ & (B, A) \in E_u \\ \iff\ & B \text{ is called from the implementation of } A. \end{aligned}$$

In this constellation, we call A the *caller* of B and B a *callee* of A. An example is depicted in Figure 6.14.

Having an interface that makes navigation in the call graph explicit and allows to traverse it interactively in both directions (Figure 6.15) has a number of benefits:

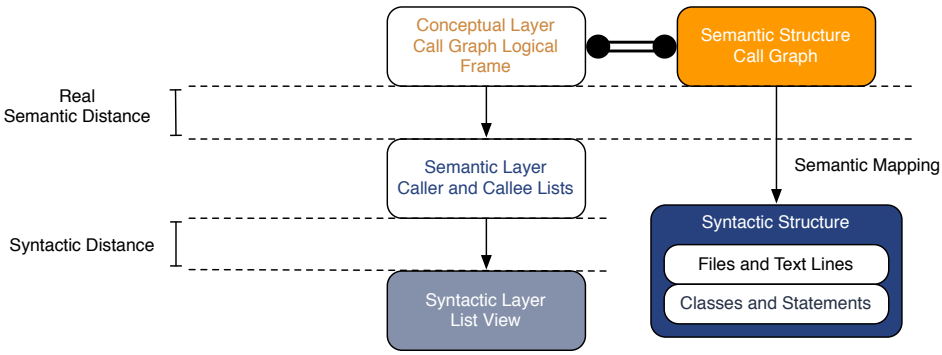

Figure 6.15: Combined navigation interface model for a call-graph-based semantic tool. Users can directly navigate through a representation of the call graph or parts thereof. The semantic distance in the interface is significantly reduced compared to standard navigation interfaces (cf. 6.12).

- Learning how a certain aspect of a program works and understanding the code that implements this aspect can be easier, because the responsible call path through the code can be directly followed. Such exploring of unfamiliar source code was one activity where we could observe developers following structural relationships, especially the call graph.

- Estimating possible side effects and assessing the scope of changes developers make to a method can become easier, because the upgraph edges at the current method directly represent the context in which that method will ever be called.

- If each navigation action along an edge of the call graph is quick to invoke and to backtrack, the notion of what is closely located in the developer's mental model of the code becomes similar to what is closely related in the semantic structure. This would allow better context retention when exploring the semantic neighborhood of a piece of code (cf. [Ko et al., 2006]), especially compared to when the developer has to use syntactic navigation like, e.g., scrolling to the target method or searching for it.

6.2.2 Determine the Semantic Mapping

Our general approach to create semantic navigation techniques—finding a sufficiently well-defined initial mapping from the syntactic domain to the semantic domain and reversing that mapping as discussed in chapter 2 "Generating New Interfaces Using the Combined Model"—is much simpler for source code than for the other media we have discussed so far: For audio, the semantic structure had to be extracted manually (see 3.2.2 "Determine the Semantic Mapping" (p. 72)), and for presentation visuals the form of the medium was altered in a way that the semantic structure was mirrored in the syntactic structure and thus explicitly defined in the authoring process (see 5.3.3 "Determine the Semantic Mapping" (p. 215)). For video, it was necessary to derive—or rather approximate—the semantic structure automatically through computer vision techniques, because this structure is not explicitly encoded in the technical representation of the medium at all (see 4.2.2 "Determine the Semantic Mapping" (p. 108)).

For source code, the initial inverse semantic mapping can be algorithmically derived.

Most instances of source code are different in that regard; our designated semantic structure, the call graph, can theoretically be algorithmically recovered in full from the syntactic representation. The reason for this is obvious: source code is nothing but a syntactic encoding of a program's semantics. As such, all semantic relations and the semantic structures they can induce, including the call graph, must be explicit in the technical representation of the medium—if they weren't, source code could not be compiled and run by a computer. We can thus take the fourth and last of the approaches for constructing the initial mapping, *"Automatic generation of the initial mapping in semantically structured media"*, from the list that we compiled in the theory chapter (see 2d "Generating New Interfaces Using the Combined Model").

A full Objective-C call graph can only be recovered at runtime.

We concentrate our studies, as discussed above, to sources written in the Objective-C language in connection with the Cocoa API. Being a dynamic language, some of the aforementioned statements on the full recoverability of structures like the call graph are thus only applicable to some degree because of features like dynamic binding: in Objective-C, the full call graph is only determined at runtime and can

even change over the course of a program's execution. In a majority of programs, however, a good approximation to the call graph can be extracted by static analysis methods directly from the sources.

We use a static analysis to create an approximation to that structure.

In the case of STACKSPLORER, we settled with a conservative heuristic to construct a superstructure of the actual call graph. Since the STACKSPLORER prototype is designed as a plug-in for Xcode 3, we can utilize the internal data structures and undocumented SPI of Xcode's lexer, tokenizer, and syntax autocompletion engine for our purposes. The full procedure is explained in Krämer's Diploma Thesis Krämer et al. [2010]; here, we only summarize the important steps:

The list of symbols and their location in the source code are extracted from Xcode's internal project index.

First, we create the set V by asking the internal project index data structure of Xcode for a list of all methods. Each method is encapsulated in an object, also providing information about the class it belongs to and its location in the source code. We then proceed by iterating over the methods and finding all *callees* for each of them. This is a multi-step process that starts out by traversing the syntax tree generated by Xcode's own lexer and collecting all expressions and sub-expressions that are either a Smalltalk-style message send or a property-style call by means of a 'dot'-expression.

For the messages, the bracket expression is parsed for the name of the selector; this selector is then fed to the autocompletion engine together with the context of the encapsulating expression to have the engine output the class that the selector belongs to. With these two pieces of information—selector name and according class—the method called in the message sending expression can be identified.

The process for the 'dot'-expression works in a similar way but has to include checks for it being used as an lvalue or rvalue: in the case of the former, the expression must resolve to a call to the property setter method; in the case of the latter, the expression represents a call to the getter. Again, the code completion engine is used to reveal the class for the accessor name, which identifies the method.

In the next step, the call locations are associated with the identified method objects, and the method objects are made

sure to point to the definition site of the method and not the declaration site. This is followed by the final step of adding the inverse *caller* relationship to each of the *callee* edges that we have found.

The resulting structure is—as explained above—only an approximation to the true call graph. Some spurious edges may be detected, thereby adding additional possibilities for navigation in the semantic structure that do not have a semantic correspondent. This, however, can be easily spotted by the developer and thus is not critical for our use case.

Our call graph contains some spurious edges, which are unproblematic.

6.2.3 Designing the User Interface for Semantic Source Code Navigation

After we have identified a suitable conceptual model for the users' task and facilitated functional access to the according semantic structure of the medium by establishing the semantic mapping, we can now define the interface and integration into the Xcode IDE to allow users to semantically navigate source code. Designing such a navigation technique that allows easy and efficient browsing of the call graph of a program is challenging. An initial idea might be to visualize the call graph as a whole; the number of methods in a program—and therefore the number of nodes in the graph—may become exceedingly large, however, and call graphs often also have very high degree nodes. This makes graphical representations of the graph infeasible both computationally and in terms of readability (Figure 6.16). A possible solution is to only make relevant parts of the call graph directly accessible. In our STACKSPLORER and BLAZE projects, we have investigated two different strategies to do so.

For the actual navigation interface, visualizing the full call graph is not possible.

For both designs, we first identified a number of key requirements from unstructured interviews with the participants of our formative study (cf. 6.2.1 "Formative Study on Navigation Behavior"), our own experience, and a thorough review of the related work on other source code navigation systems that are integrated into an IDE [Oman, 1990; Čubranić and Murphy, 2003; Janzen and De Volder, 2003; DeLine et al., 2005b; Singer et al., 2005; Kersten and Mur-

We explored two different designs that each make a different part of the call graph accessible.

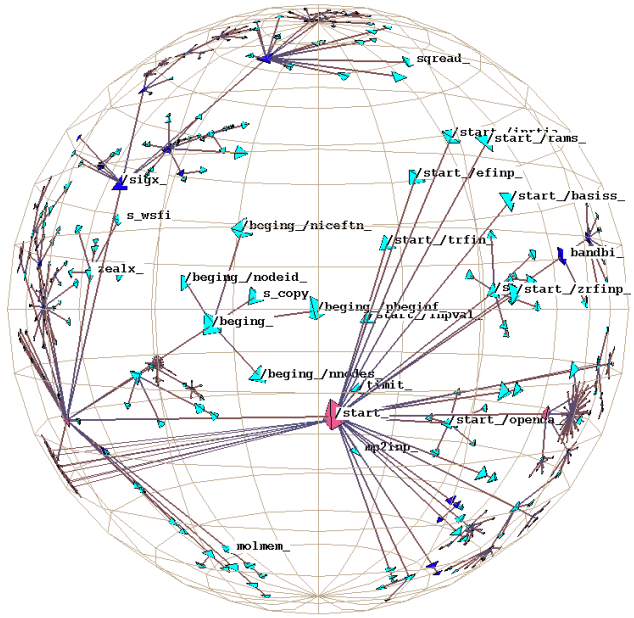

Figure 6.16: Even for small code bases, call graph visualizations quickly become difficult to parse and read. Source: [Munzner, 1997]

phy, 2005; DeLine et al., 2005a; Deline et al., 2006; Hill et al., 2007]:

- The parts of the call graph that are accessible in our interface should include and be centered around the *focus method*, which is the method that is currently edited in the source code editor. This is consistent with how Herman et al. [2000] propose incremental exploration of graph structures; the accessible part of the call graph corresponds to their concept of a *logical frame*, the focus method corresponds to the *focus node* in the logical frame [Huang et al., 1998] (Figure 6.17).
- Navigation and exploration of the call graph as the semantic structure should work by selecting a new focus node from the logical frame, thereby re-defining the latter. If another form of navigation is performed that changes the focus method, the focus node and logical frame should be updated automatically (Figure 6.18).

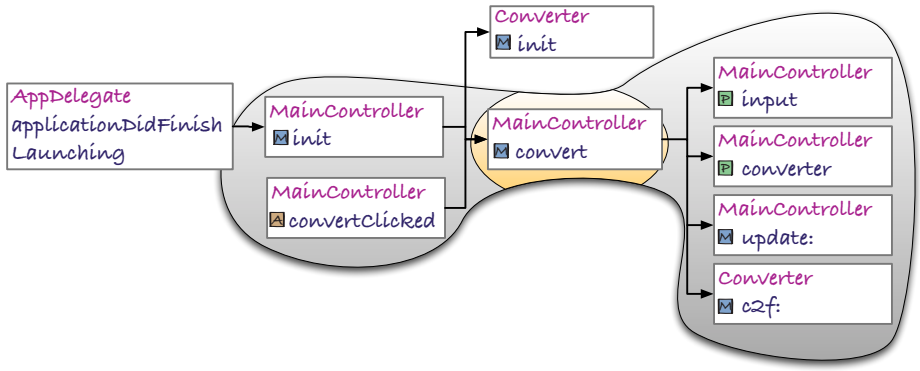

Figure 6.17: Logical frame inside the call graph. The *focus method*—the method currently being edited in the source code editor view of the IDE—defines which other nodes belong to the logical frame. Source: [Karrer et al., 2011]

This ensures that the logical frame is always current and relevant.

- The visual part of the UI should be integrated with the source code editor as closely as possible to keep the developers' locus of attention at the code and not at a 'tool'. Unlike suggested by the model of *instrumental interaction* [Beaudouin-Lafon, 2000], we try to allow users to directly interact with the objects of interest from their conceptual model of the task at hand (see 2.2.1 "Interface Model and Design Space" (p. 42)). Therefore, our goal is to make our interface be perceived as part of the code instead of an instrument.

STACKSPLORER

We need a suitable topological layout for the logical frame.

The first decision we have to make to design a UI for STACKSPLORER in accordance with the requirements detailed above is to define the topological layout of the logical frame, the part of the semantic structure that should be directly visible and accessible. Obviously, there is a trade-off between the number of methods that we can visualize as navigation targets and the available screen real estate inside the IDE; we want to show as much information and allow as much navigation interaction as possible without overload-

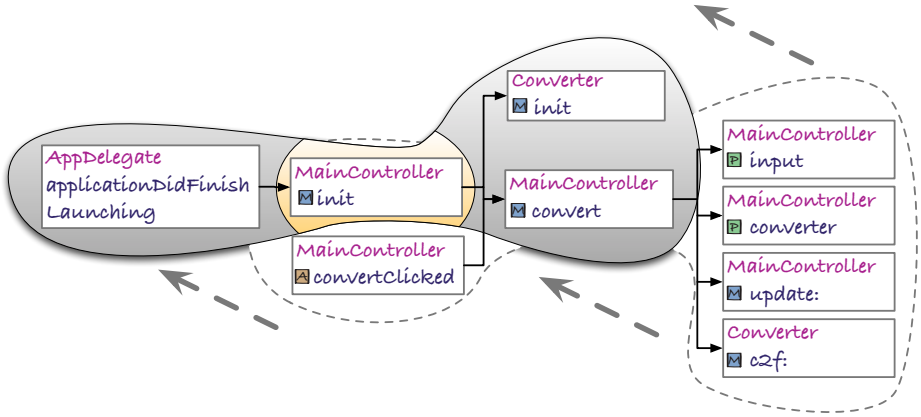

Figure 6.18: Moving the logical frame over the call graph. When changing the focus method, the logical frame and thus the part of the call graph that is being visualized changes with it. Source: [Karrer et al., 2011]

ing the UI or drawing attention from the source code. Thus, we start our search for a suitable choice for the logical frame by reducing the complex topology of the call graph.

We first restrict the full call graph to a dual tree structure of a caller tree and a callee tree, both rooted at the focus method.

Since one of the reasons for our initial choice of the semantic structure was that developers follow the control flow of a piece of source code in order to understand it [Pennington, 1987; Sim et al., 1998], we can first prune all edges from the call graph that can only be reached by reversing the direction of traversal when viewed from the focus method. The resulting structure can be modeled as two independent trees that both have the focus method as their root: one represents the *caller* tree and contains only edges in caller direction, the other analogously represents the *callee* tree that contains only edges in callee direction 6.19. In this way, the structure precisely contains all possible call-paths that travel through the focus method. This dual tree structure obviously depends on the current focus method and has to be re-built from the call graph every time the focus method changes.

For STACKSPLORER, we decided to include the full direct neighborhood of the focus method into the UI. This allows us to keep the number of navigation targets and their visualization manageable while still supporting the 'glancing'

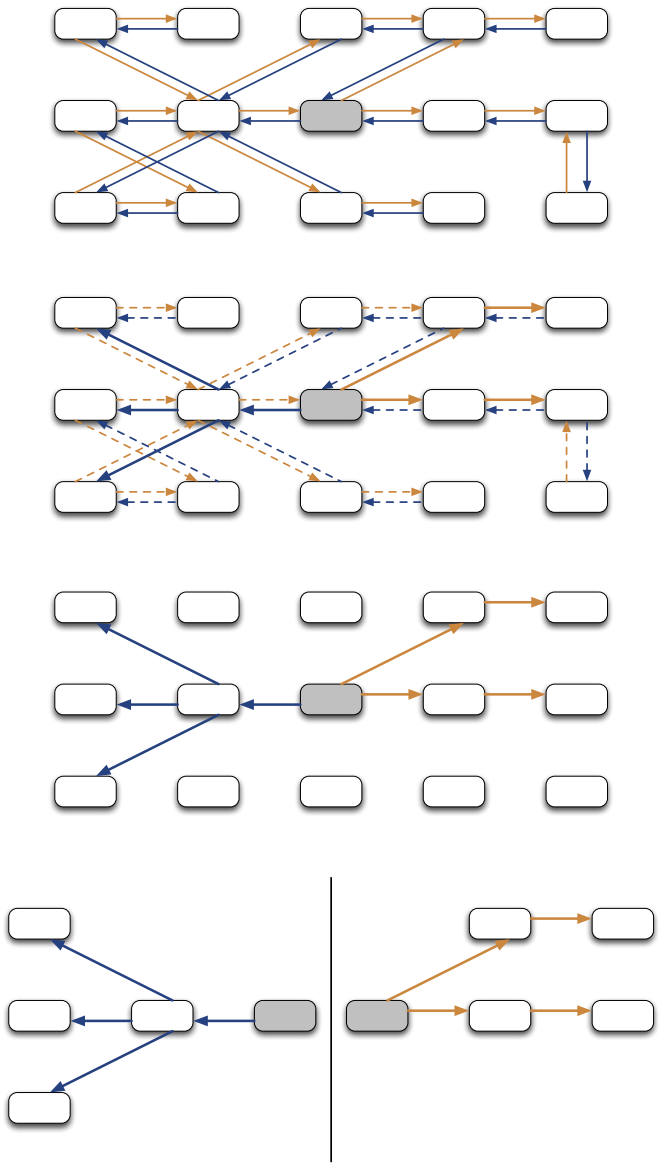

Figure 6.19: Pruning of the call graph to only include all possible call-paths through the current focus method. From the initial full call graph with possible recursions unrolled (top), all methods that are not reachable from the focus method by a sequence of only caller or callee steps are removed (middle). The remainder of the graph can be modeled as two trees, both with their roots at the focus method.

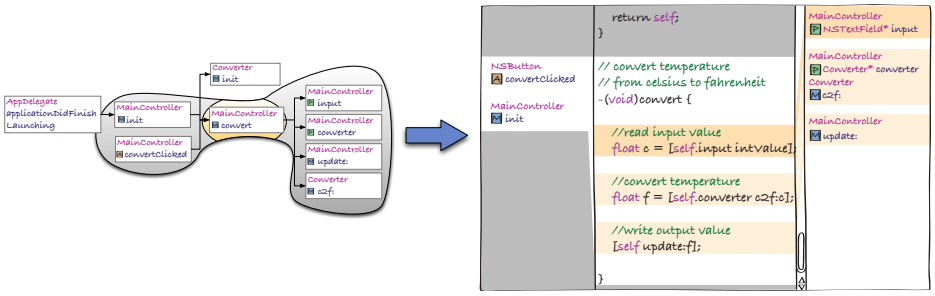

Figure 6.20: Sketch of the STACKSPLORER user interface. The direct neighborhood of the focus method in the call graph is accessible through the side columns next to the source code editor.

In STACKSPLORER, the direct neighborhood of the focus method forms the logical frame; these methods are accessible through two side columns in the interface.

navigation pattern observed by Ko et al. [2006]. The methods in the two partitions of this subgraph, upstream and downstream of the calling direction, are then displayed in two side columns to the left and to the right of the main source code editor view inside Xcode (Figure 6.20). In this way, we can keep the presentation of the syntactic structure of the code in the usual way—as a text file, scrolling vertically from top to bottom—while at the same time adding our semantic structure on the horizontal axis 6.21. The resulting interface could also be interpreted as a fisheye view Furnas [1986] for the semantic structure: The focus method is completely visible with its source code, the neighborhood only through the abstraction in the side columns, and the rest of the call graph is faded out.

Clicking on a method in one of the side columns shifts the logical frame over the call graph and then updates the UI.

To navigate the call graph in the STACKSPLORER UI, a developer can simply click on one of the methods in the side columns next to the editor; this causes the clicked method to smoothly move to the central view, which then shows the file in which the method is implemented, centered on the method. The former focus method moves over to the left and becomes an entry in the *caller* column. The *callees* of the new focus method move in from the border of the screen, replacing the old right hand side column.

The logical frame is always synchronized to the edit location in the code editor.

As soon as the cursor in the editor view is positioned inside another method, the STACKSPLORER UI is automatically updated. Also, scrolling through a lengthy focus method causes the entries in the side column to move as well, al-

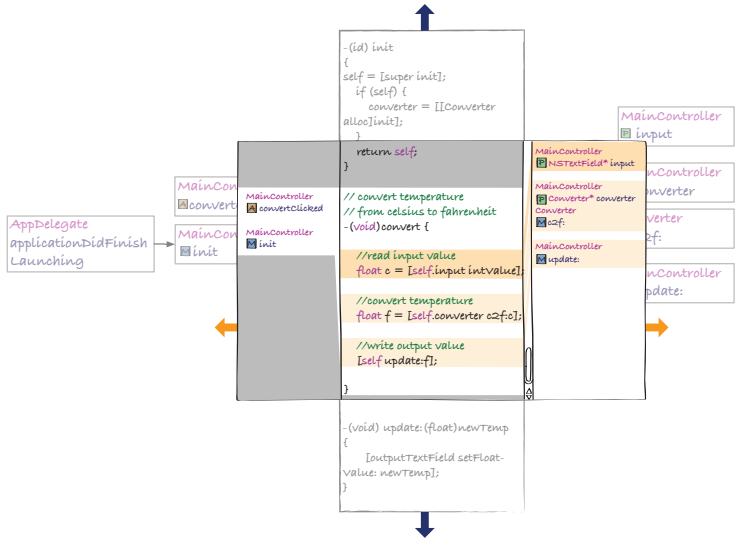

Figure 6.21: Semantic navigation concept in STACKSPLORER. The well-known syntactic navigation through source code files is preserved on the vertical axis, while the semantic navigation through the call graph is mapped to the horizontal axis.

ways minimizing the spatial distance between their location in the side column and the occurrence of the respective call in the source code editor. This simplifies grasping the context of the focus method [Karrer et al., 2011]. It also makes the right hand column provide an abstract overview over the content of the focus method; during our evaluation experiments, we could observe some developers who tested the prototype to often refrain from reading the code of a method in full but relied on the summary they could gather from the callee method list.

For the actual implementation of the STACKSPLORER prototype (Figure 6.22), we also refined the design in a number of places to make the UI cleaner and to provide information scent to the developer:

Firstly, following the idea of *wear-based filtering* [DeLine et al., 2005b], the history of method nodes that have been visited is visualized by adding a color-shaded background to the representation of the methods in the side columns.

STACKSPLORER supports 'glancing' by highlighting visited methods.

This again facilitates the glancing navigation pattern as well as it helps the local graph exploration patterns observed by Sillito et al. [2008]. The shading intensity gives a hint about the recency of the last visit to the method.

Frame overlays visually associate entries in the side columns with locations in the code.

Secondly, to facilitate quick association of the side column entries with the source code in the center view, we offer optional frame overlays that graphically link the method calls in the code with their occurrence in the side column. Also, we allow to optionally emphasize the horizontal semantic navigation over the vertical syntactic navigation by visually toning down the rest of the code in the center view that is not part of the current focus method.

Methods can be tagged and color-coded to document call paths.

Thirdly, STACKSPLORER allows to attach tags to individual methods. These can be used to mark important call paths, for example the full path of the methods involved with a screen re-draw in a graphics application. They therefore fulfill a double role, acting both as self-generated information scent for one developer and as a form of documentation between different developers working on the same code base. Tags consist of a name and color, which can be defined in a separate editor window 6.23. In the main UI, they are visible as a colored icon next to the tagged methods in the side columns. If the graphical overlays linking the code and the side columns are enabled, the tags are instead visualized by drawing the frames in the respective color.

BLAZE

For BLAZE, we tried to tailor the topological structure of the logical frame to larger navigation patterns.

For BLAZE, we defined the topological layout of the logical frame in a different way. Instead of catering to breadth-first-search exploration of the focus method's local neighborhood inside the semantic structure, as we did with the STACKSPLORER design, we tried to focus on larger navigation patterns. In particular, our aim was to explicitly support the navigation strategies observed by Sillito et al. [2008] and ourselves [Karrer et al., 2011], which are characterized by alternating phases of linear searches through the code and local cluster exploration.

BLAZE therefore makes a different local neighborhood of the focus method accessible; it consists of a single linear call

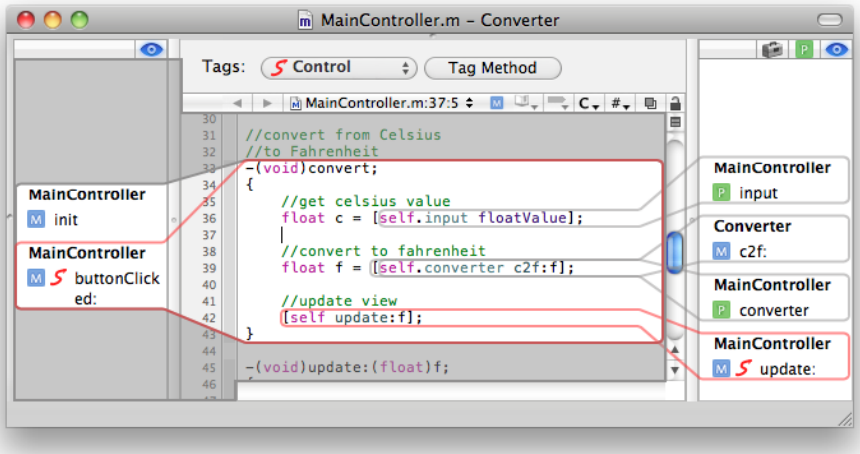

Figure 6.22: Screenshot of the STACKSPLORER user interface. In this example, the overlay frames that connect the method entries in the side columns with the actual location of the methods in the source code are enabled. The chain of methods that is framed in red has been tagged before and given a common label (cf. Figure 6.23) Source: [Karrer et al., 2011]

path through both halves of the dual tree structure (see 6.2.3 "STACKSPLORER " (p. 306)) and, naturally, through the focus method (Figure 6.24). Thus, the common behavior of exploring and backtracking along one special call path is supported very well and can be accomplished very easily [Kurz, 2011].

The logical frame contains one full path through the call graph in both directions.

There are a number of subtle side effects that this change of the topology of the logical frame brings. First, while there is only one direct neighborhood of the focus method at any time, there are multiple—and possibly many—paths through the focus method. This means that the logical frame is not determined by the choice of the focus node alone but depends on a vector of parameters, representing a hierarchy of choices between subtrees. The length of this vector is equal to the length of the path minus one, and each entry can take as many values as the respective method's fan-in or fan-out degree. This makes the parameter space for the logical frame rather large. Second, as switching the focus method via the semantic navigation interface is limited to the logical frame, we cannot navigate the whole

This kind of logical frame has more degrees of freedom, which have to be controllable in the UI.

We need separate ways to relocate the logical frame and use it for navigation.

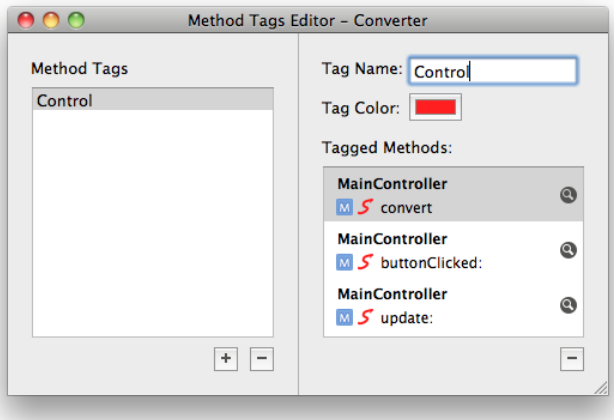

Figure 6.23: Screenshot of the STACKSPLORER tag editor window. Methods can be tagged to signify that they belong to certain special call paths. This can be seen as a quick and lightweight form of documentation: developers can, for example, tag all methods that are responsible for screen redraws, helping them and others to later find these locations in the code again.

call graph with this technique. We must therefore allow to reconfigure the logical frame without changing the focus method; in other words, each of the parameters mentioned above must be settable in the interface. Third, we must be careful when designing the behavior of the interface for when the focus method changes. Since a new focus method basically invalidates the current parameter vector, we have to guess which path through the call graph and new focus method to set the logical frame to. And last, there must be a way to navigate through the logical frame without changing it to reap the benefits of investigating the code along the path and always being able to backtrack to a safe location. This means, of course, that the focus method must not auto-update to the current location in the code for these cases. As a result of these requirements, the interface for BLAZE is somewhat more complex than that of STACKSPLORER.

We decided to represent the currently selected call path as a stack of methods in one single side column next to the

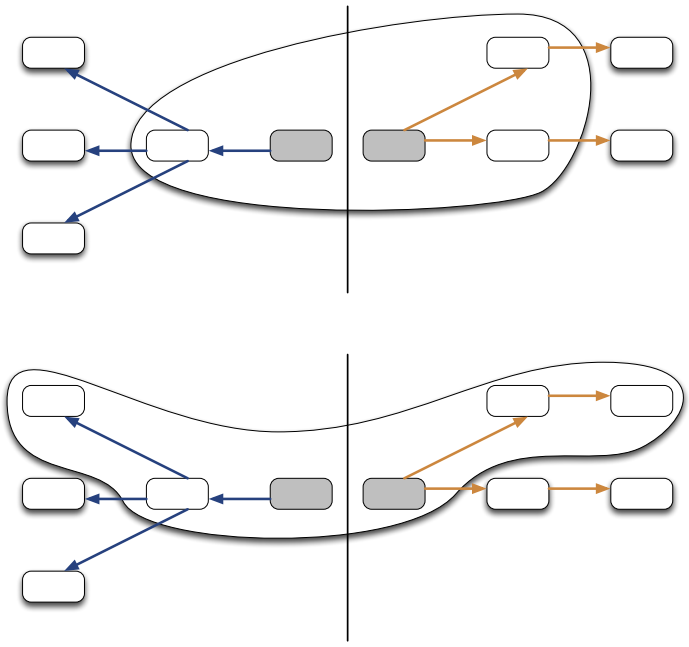

Figure 6.24: Different topologies of the logical frame in STACKSPLORER (top) and BLAZE (bottom). While in STACKSPLORER only the direct neighborhood of the focus method in the call graph is accessible, BLAZE shows a single full path through the call graph.

The logical frame is visualized in one side column as a double-stack around the focus method.

editor view in Xcode 6.25. The focus method is drawn visually highlighted, with the calling part of the path above it and the called part below. To configure the displayed call path, each of the method nodes can be swapped with one of its siblings through two arrow buttons. The interface thus is similar to a series of interdependent picker widgets or a combination lock. Alternatively, clicking on the boundary between two methods, opens a transient HUD menu that lists all child nodes of the higher level one (Figure 6.26); this menu allows a quicker selection of the subtree at any node in the path than the left/right arrow keys.

To separate navigation and relocation, the focus method is lockable.

The focus method is usually in a 'locked' state, meaning that a click on one of the methods in the path visualization takes the developer to the implementation of that method but does not change the focus method. This allows the path

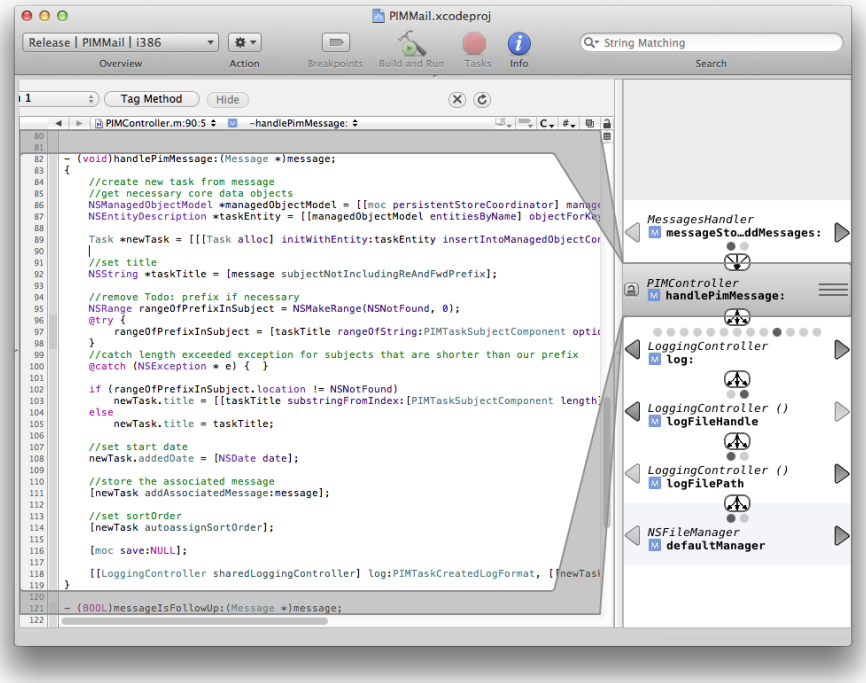

Figure 6.25: Screenshot of the Blaze UI as a plug-in for the Xcode IDE. The right hand side column shows the current focus method in grey, the incoming call path above, and one outgoing call path below.

to be explored in depth while always offering access to each node in the path and a direct jump back to the focus method. To change the focus method, it can be 'unlocked'; in this situation, every click on one of the methods in the path makes that method the new focus method.

Having the focus method unlocked facilitates the first phase of navigating, locking the method facilitates the second phase.

The navigation model of BLAZE thus embraces the two-phase navigation pattern of developers [Krämer et al., 2010; Karrer et al., 2011]. First, the developer searches for a location that is relevant for the maintenance task at hand. This is done with the focus method unlocked where BLAZE supports quick, depth-first exploration of the call graph. In the second phase, the developer gathers context information in the semantic neighborhood of the found location. This is done with the focus method locked where call paths to and from the location of interest can be explored and modified,

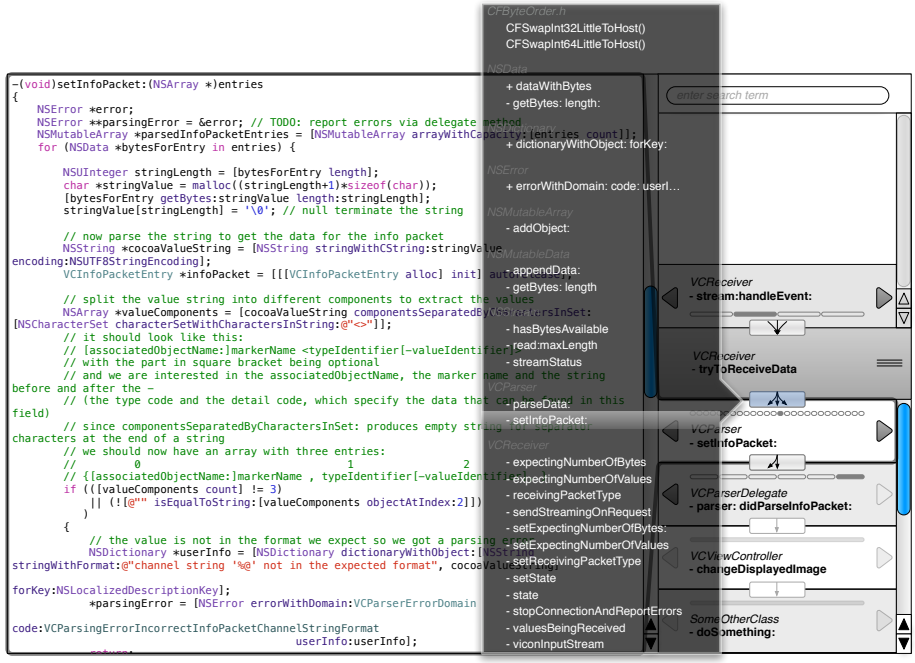

Figure 6.26: Sketch of the BLAZE user interface showing the branch selection menu. This is the menu design that was picked for the final software prototype. Source: [Kurz, 2011]

while always offering backtracking and direct navigation back to the focus method.

6.2.4 Evaluate the Interface

To evaluate our choice of conceptual model and semantic structure for source code as well as our different interface ideas of STACKSPLORER and BLAZE, we conducted a comparative study between a regular, unmodified Xcode installation and both of our semantic navigation interfaces[5]. As an additional control condition that represents the same se-

We compared both of our navigation techniques with the call hierarchy tool and an unmodified Xcode IDE.

[5]Parts of this evaluation have been published in Jan-Peter Krämer's Diploma Thesis [Krämer et al., 2010], as a full paper at UIST 2011 [Karrer et al., 2011], in Joachim Kurz's Bachelor's Thesis [Kurz, 2011], and will be published as a full paper at CHI 2013 [Krämer et al., 2013]

mantic structure, we also included a version of Xcode that we modified by adding a *call hierarchy* view (Figure 6.27).

The call hierarchy is a navigation tool that already exists in the current versions of Eclipse and Microsoft Visual Studio and that allows access to the call graph from a focus method through a hierarchical tree view interface. Similar to BLAZE with the focus method locked, the call hierarchy keeps its state and does not auto-update when the developer navigates through the code. It can, however, be manually updated at any time through a menu or keyboard shortcut.

We had participants find a change location in the *BibDesk* source code and assess possible side effects of the change.

In this experiment, we measured the success rates and task performance times for two code maintenance tasks. We used the open source project *BibDesk*[6], a BibTEX-based bibliography management tool for OS X that is written in Cocoa, as the code base for our study. The two maintenance tasks were designed to test the influence of the different navigation interfaces on *identifying a suitable location and change set* for implementing a suggested feature and on *assessing the side effects* of this change at a designated code location.

The tasks concerned BibDesk's *Autofile* feature, which automatically sorts PDF documents of publications into dedicated folders and renames them according to a user-definable naming scheme. In the first task, participants were asked to prepend the string "TRIAL" to every generated PDF file name. In the second task, we asked the participants to identify side effects that would occur if the change from the first subtask was implemented in a specific method [Karrer et al., 2011].

The tasks were considered successful when the correct location was found or when one correct side effect was identified, respectively.

To solve the first task, a participant had to perform the code change in a way that it would have achieved the desired effect as stated in the task description. For the second task, the participant was required to list as many changes (apart from the intended change, of course) as possible in the functionality of the application that would result from the code modification. Identifying one correct side effect was enough for the second task to be considered as successful; consequently, the time recoded also reflects the time until this first correct solution.

[6] http://bibdesk.sourceforge.net

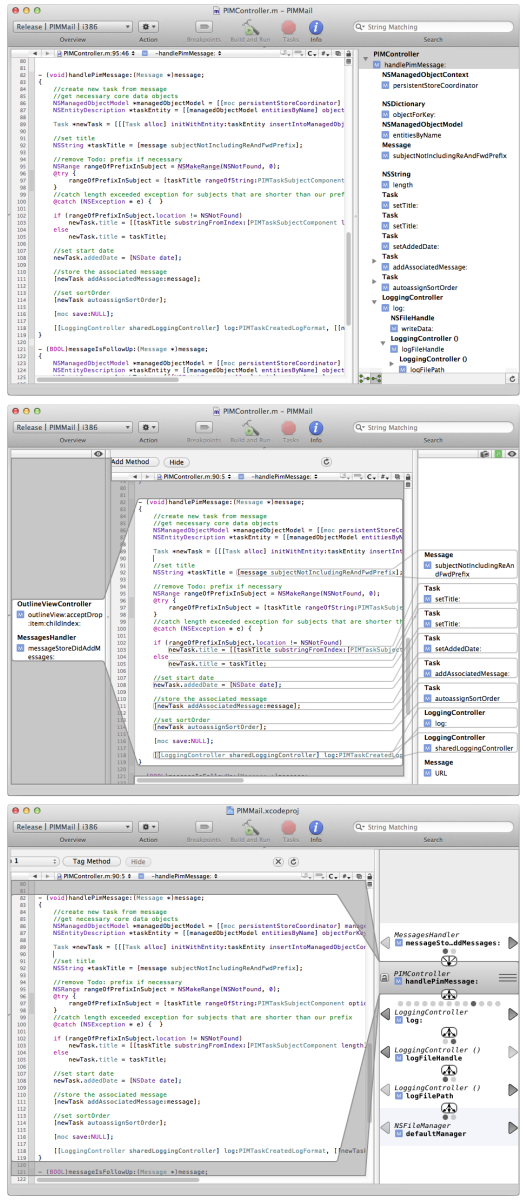

Figure 6.27: Screenshots of the three non-standard user interfaces for source code navigation along the call graph: call hierarchy (top), STACKSPLORER (middle), and BLAZE (bottom). Source: [Krämer et al., 2013]

After the time cap, the task was considered unsuccessful.

The time for each trial was capped at and limited to 25 minutes for the first and 15 minutes for the second task. If a participant could not find a correct solution in the allotted time for each task, we regarded this attempt at the task as unsuccessful and recorded the maximum time.

Quantitative Analysis

We hypothesized that both maintenance tasks will have a higher success rate but require less time when our semantic navigation tools are available.

To evaluate both our interface designs as well as our underlying navigation model, we formulated four hypotheses:

H1 More developers can find a correct solution to both tasks in the allotted time if the IDE offers a tool for semantic navigation with the call graph as the semantic structure.

H2 Developers overall take less time for both tasks if the IDE offers a tool for semantic navigation with the call graph as the semantic structure.

H3 More developers can find a correct solution to both tasks in the allotted time if they are using STACKSPLORER or BLAZE in contrast to the call hierarchy as a standard call graph navigation tool.

H4 Developers overall take less time for both tasks if they are using STACKSPLORER or BLAZE in contrast to the call hierarchy as a standard call graph navigation tool.

The first two of these hypotheses reflect our claim that semantic navigation in general and navigation that uses the call graph as its semantic structure in particular is superior to syntactic source code navigation, although the latter is still the dominant design paradigm in modern IDEs. The remaining two hypotheses are meant to reflect the influence our particular interface designs for semantic source code navigation—that of STACKSPLORER and that of BLAZE—have on the effectiveness and efficiency of common code maintenance tasks.

The experiment was performed in two parts—the first one with STACKSPLORER and the unmodified Xcode and the second one with BLAZE and Xcode with a call hierarchy

plug-in—using the same code base, tasks, and experimental setup. In the following, we will evaluate the aggregated results of both parts of the experiment; the exact descriptions of the methodology, participant demographics, and experiment setup can be found in the respective prior publications [Krämer et al., 2010; Karrer et al., 2011; Kurz, 2011; Krämer et al., 2012, 2013].

The success rates for the combined tasks were significantly higher when using call graph navigation tools.

We first look at the task success rates for all four conditions: *call hierarchy*, STACKSPLORER, BLAZE, and *Xcode*, with the latter as the control. For this, we considered a participant's trial as successful only if both tasks, the identification of a suitable location and change set for a modification and the assessment of possible side effects of a suggested change, had been solved successfully by the participant. The results are shown in Figure 6.28. We compared the success rates for the combined tasks using a one-sided Fisher's exact test, which showed that the success rates for call-graph-based navigation were significantly higher ($p = 0.021$) than for *Xcode* alone, thus confirming **H1**. If we look at both tasks individually, we can see that the difference in success rate is only significant for the second task ($p = 0.026$, Fisher's exact test). The reason for this could be that the first task required both down- and upgraph navigation, and the former can be accomplished in Xcode by using the 'jump-to-definition' tool. The second task primarily depended on upgraph navigation, which is not explicitly supported in Xcode.

The success rates between the call graph navigation tools did not differ significantly.

Following up on this positive result, we compared the success rates of the three call-graph-based navigation tools only, with the *call hierarchy* acting as the control. Although the success rate for the call hierarchy was always below that of both our interfaces, the result of a Fisher's exact test was not significant ($p = 0.35$); therefore we cannot confirm **H3**.

Both of our call graph navigation tools led to significantly lower task completion times compared to Xcode.

After analyzing the effectiveness of the four tools, we also examined their efficiency in the form of task completion time. A two-way independent measures analysis of variance (ANOVA), with the two factors being *task* and *condition*, indicated significant effects for both factors individually (task: $p < 0.001$, $F(1, 55) = 81.877$, condition: $p = 0.01$, $F(3, 55) = 4.125$). The significant effect of the task was expected, since the first was designed to take longer than the second task, which is also reflected in the maximum allotted times (25 vs. 15 minutes). We followed up on the sig-

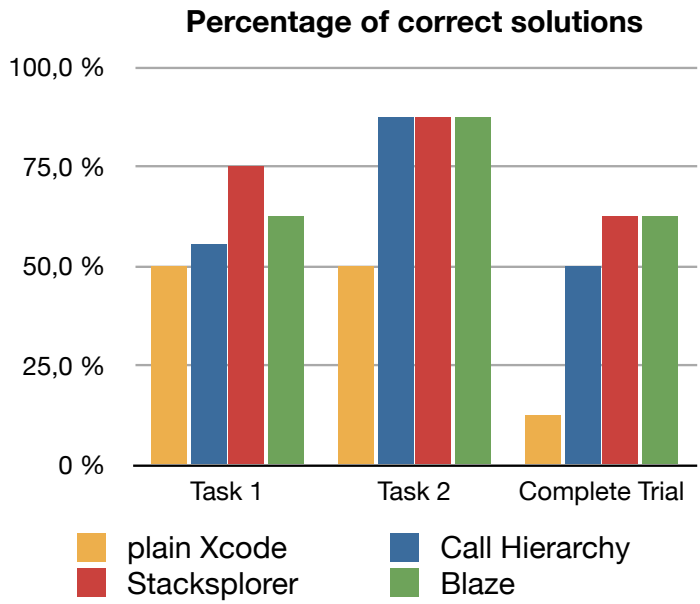

Figure 6.28: Task success rates for all four conditions in both tasks and combined. For the second task and for both tasks combined, the success rates are significantly higher when offering call-graph-based navigation in the IDE. Source: [Kurz, 2011]

nificant effect of the condition using a post-hoc one-tailed Dunnett's test with the *Xcode* condition as the control. The result shows that for both our call graph navigation methods the tasks could be completed on average significantly faster than with the unmodified *Xcode* IDE (Xcode vs Stacksplorer: $p = 0.024$, Xcode vs Blaze: $p = 0.045$); the difference in task completion time between the unmodified Xcode and the call hierarchy version was not significant (Xcode vs CH: $p = 0.807$). **H2** therefore is too general and cannot be accepted.

STACKSPLORER and BLAZE were significantly more efficient than the call graph hierarchy.

For comparing the efficiency between the three different user interface designs for call-graph-based navigation, we conducted Tukey post-hoc tests. The results were significant (call hierarchy vs. STACKSPLORER: $p = 0.037$, call hierarchy vs. BLAZE: $p = 0.046$), allowing us to confirm **H4**. Comparing STACKSPLORER and BLAZE did not reveal any statistically significant differences in either effectiveness or

efficiency.

We created a framework to analyze the navigation behaviors of developers.

In oder to find out what causes these differences between the three call graph navigation tools, we proposed a framework that helps to model, quantify, and compare the navigation behavior of developers. The idea is to transform individual navigation trails—sequences of navigation events that are classified and labeled—into a seven-dimensional feature space with axes that represent well-known micro-navigation patterns. Numerically, each entry in the feature vector represents the prediction accuracy of a simple mathematical navigation model. The transformation can thus be imagined as a filter bank where each filter resonates with a different micro-navigation pattern. A detailed description of this analysis framework will be published in [Krämer et al., 2013].

Both of our tools promote semantic navigation better than the call hierarchy.

We used this framework to analyze the navigation behavior of each participant in each trial. This required manually annotating all navigation trials—we determined timestamp, kind of navigation, used tool, start location, and end location—and then calculating the prediction accuracies of all seven navigation models. The results show that developers performed significantly more navigations along the call graph (prediction accuracy of the *forward call depth* model) when using STACKSPLORER or BLAZE compared to Xcode (one-sided Dunnett's t-test, Stacksplorer: $p = 0.003$, Blaze: $p = 0.037$) and also compared to the call hierarchy (Tukey's test, Stacksplorer: $p = 0.003$, Blaze: $p = 0.044$) [Krämer et al., 2013]. Similarly, STACKSPLORER and BLAZE caused developers to perform longer navigation chains along the call graph than they did when using Xcode alone (one-sided Dunnett's t-test, Stacksplorer: $p < 0.001$, Blaze: $p = 0.015$); this was interestingly not true for the call hierarchy (call hierarchy: $p = 0.562$) [Krämer et al., 2013]. These results, together with the parallel increase in efficiency, are a strong indication that the call graph is a suitable choice for the conceptual model of source code, that both our subgraphs make for useful semantic structures, and that we can use our interaction model to create semantic navigation interfaces that make a positive difference in how developers perform certain code maintenance tasks.

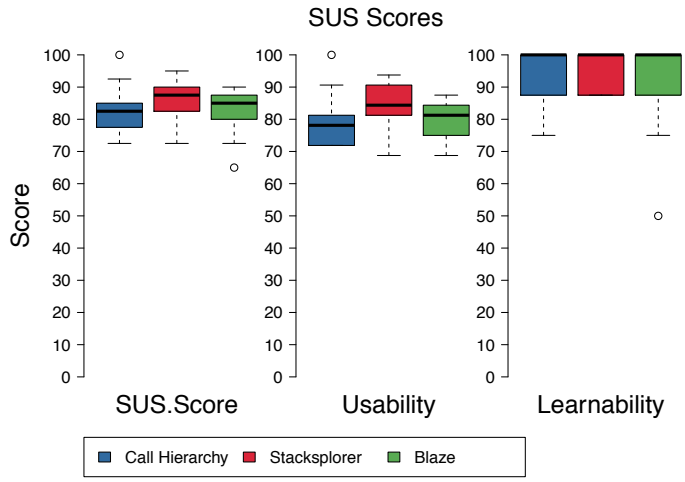

Figure 6.29: System usability scale scores, combined and separated by usability and learnability, for all three call graph navigation tools. Source: [Kurz, 2011]

Qualitative Analysis

Both UIs were also evaluated according to the SUS.

In addition to these quantitative measurements, we also collected qualitative results through a post-session questionnaire, which contained the standard system usability scale (SUS) [Brooke, 1996] and six additional questions (cf. Figure 6.30). The questions were asked with regards to the call graph navigation interface that was used by a participant but not with regards to the plain *Xcode* control condition.

All call graph navigation tools were considered 'excellent'.

The results of the SUS part of the questionnaire and the distribution of scores in the *usability* and *learnability* subscales [Lewis and Sauro, 2009] are shown in Figure 6.29. All three call-graph-based navigation interfaces were evaluated as 'excellent' [Bangor et al., 2008], with STACKSPLORER (median 87.5) scoring better than BLAZE (median 85) or the call hierarchy (median 82.5). These differences, however, are not statistically significant ($\chi^2(2) = 2.40$, $p = 0.301$).

The six additional questions covered three different aspects of how they could benefit code maintenance tasks: questions 11 and 12 were concerned with code understanding,

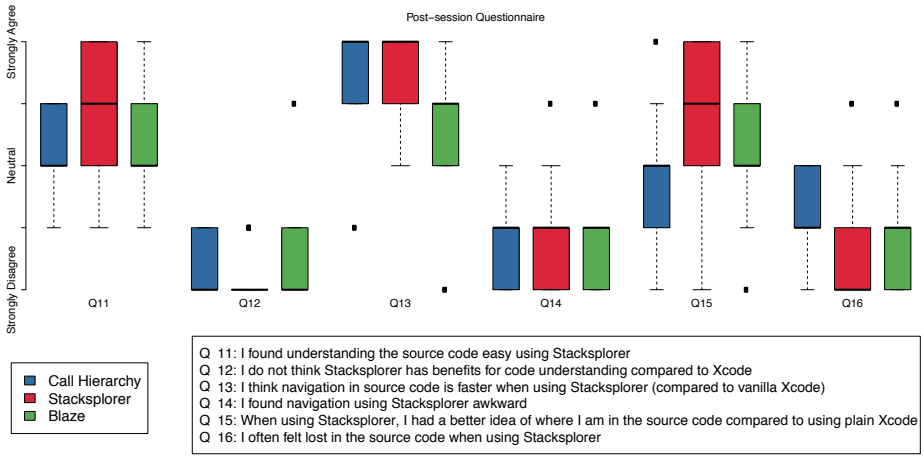

Figure 6.30: Results of the post-session questionnaire for the three call graph navigation tools. These questions were concerned with code understanding (Q11 and Q12), navigation (Q13 and Q14), and orientation in the code (Q15 and Q16). The boxes for the STACKSPLORER condition are wider to indicate the larger sample size (17 vs. 8 in the other conditions). Source: [Kurz, 2011]

questions 13 and 14 covered the users' perceived impact on the navigation itself, and questions 15 and 16 were asked to assess how the participants' sense of orientation in the source code was influenced by the tools. As can be seen in Figure 6.30, all three source code navigation interfaces that use the call graph were received relatively similarly in all three aspect areas; as a trend, STACKSPLORER generally received better scores than BLAZE, which in turn received better—with the exception of Q13—scores than the *call hierarchy*. The reason for *Blaze's* lower score regarding the perceived speed of navigation may be twofold: First, in the tested prototype, the methods in the right hand side column are not linked visually to the code editor like it is done, for example, in STACKSPLORER by means of overlaid frames (Figure 6.22). This makes it more difficult to find the implementation of a method in the code view after navigating there. Second, setting up a complete preferred call path in BLAZE requires some extra interaction steps in comparison to the other two call graph tools. While this extra time commitment is *objectively* offset by a larger gain in efficiency (cf. 6.2.4 "Quantitative Analysis"), *subjectively* it seems to feel slower overall.

6.3 Closing Remarks

We could show that our interaction model for semantic navigation can also be applied to text-based media.

In this last of our four project chapters, we have shown how our interaction model for semantic navigation in digital media can be applied to the medium of source code. In terms of its syntactic structure, this medium is probably one of the oldest types of media that have been digitally represented—until today, source code is still mainly stored and interacted with in the form of collections of plain text documents. In contrast to arbitrary text, however, it possesses a very clear notion of purpose and associated tasks, and it has stringently defined semantics. This latter point enabled us to automatically derive the semantic structure from the syntactic representation of the code for the second step of our four-step approach to creating semantic navigation interfaces.

Semantic navigation interfaces improve task performance.

Similar to the preceding three chapters, we could again show that our model for semantic navigation can not only be applied to a wide range of digital media but that we can also increase the users' effectiveness and efficiency for high-level tasks just by improving the navigation. Given the economic relevance of source code maintenance and its strong dependency on code navigation, we are confident that our proposed or similar techniques will be adopted by commercial IDEs.

A first step in this direction has already been made by the Xcode team at Apple: after we demonstrated STACKSPLORER to them at WWDC 2011, Xcode now offers caller and callee modes for its auto-updating assistant side columns (Figure 6.31). Unfortunately, the assistant columns only show one method in code and thus do not give an overview over all callers and callees. This information can be found in the method selection pop-up at the top of the column but is otherwise invisible. Also, the UI does not allow navigation, e.g. by setting the focus method to either of the methods in the assistant columns, and the position of the code editor column is restricted to the far left.

Our research on code navigation, which had began with the idea for STACKSPLORER, will be continued by Jan-Peter Krämer. As promising directions for future work we regard investigating how the conceptual model changes in the context of debugging and how navigation through runtime

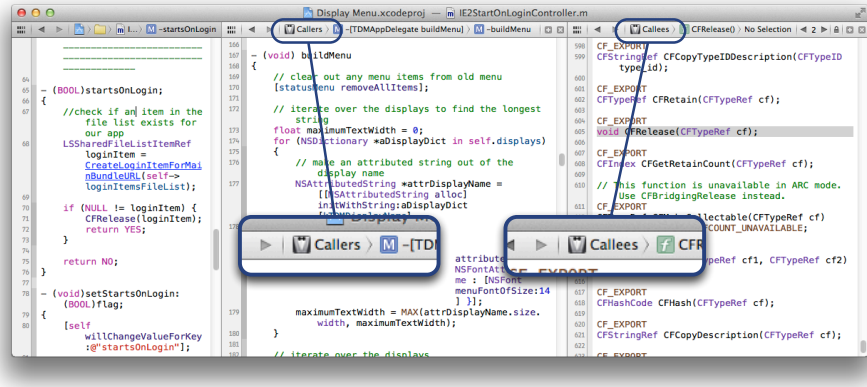

Figure 6.31: The current version of Xcode now allows its auto-updating *assistant columns* to display either a caller (middle) or a callee (right) of the focus method in the main editor column (left).

traces can help understanding the code and possible bugs. Additionally, we are in the process of analyzing individual developers' navigation paths through unknown source code to better understand the influence of the availability of different navigation tools on the developers' navigation behavior [Krämer et al., 2013].

Chapter 7

Conclusion

Navigation in digital media is a wide field that encompasses many facets; from the purely performance-based considerations of how people use sliders for linear navigation under different mapping functions to the psychological studies of how a change in the format of the presentation medium affects the learning performance of classes of students, all details contribute to the goal of changing the way we navigate through modern digital media in order to make this process easier, faster, and more enjoyable. Setting out from the observation that current interaction with these media is often still modeled after how we historically handled their analog counterparts—thus severely and unnecessarily limiting the design space for such interaction—we have looked for an approach to introduce concepts that have emerged in the design of graphical user interfaces into digital media navigation. The leading thought behind the research presented here were two questions: "Why is navigation in digital media still a change in location that is parameterized by the container and not the content?" and "If we can shift the navigation space from being container-defined to being content-defined, what interaction techniques would be most appropriate for such navigation?". With this thesis, we have made an attempt at proposing a concept in the light of these two questions that can be used to rethink and improve current navigation techniques, or to create completely new ones.

7.1 Summary and Contributions

While in the limited scope of such a document we can only scratch the surface of the area as a whole, we were able to make a number of contributions to media navigation and demonstrate their viability and positive effects in a number of example projects. These contributions include:

We discussed the interplay of existing interface and interaction theories, and we tried to point out where these theories are taken in the wrong spirit in today's navigation interfaces.

- *Review and discussion of established interaction and interface theories*
  We explained and analyzed the existing work regarding the modeling of user interfaces as a layer stack abstraction and different interaction techniques with an emphasis on direct manipulation. For both of these areas, we presented a literature review and a (hopefully) balanced discussion on the advantages and disadvantages of these theories. Both theories can be embedded into Norman's *seven stages of action*, which we used to illustrate the concept of semantic and syntactic distances in current navigation interfaces. We also offered an explanation as to why many widespread interface designs that are often considered direct manipulation interfaces actually violate the underlying principles of this interaction technique. Specifically, we illustrated that employing *technical metaphors* is prone to create large *syntactic distances* and that certain interfaces relying on direct representations of abstract values may suffer from large *semantic distances*. Interfaces for navigating digital media, unfortunately, often fall into either of these classes.

We proposed our idea of semantic navigation as opposed to the predominant syntactic navigation.

- *Semantic navigation interface idea*
  As a central contribution, we then proposed *semantic navigation interfaces* that allow users to move through a medium by directly manipulating its content instead of external parameters of its form. While conceptually simple, the challenge of the idea lies in the formulation of a consistent interaction theory, in which semantic navigation can be expressed and differentiated from regular, 'syntactic' navigation.

- *Interface model and design space*
  Based on the layer stack interface models and the design space for direct manipulation interfaces by

Hutchins et al. [1985], we then proposed our own design space and interface model for media navigation interfaces. While our adaptation of the layer model is a simplified version of the related work [Foley et al., 1997], for the design space, we laid a greater emphasis on the differentiation between syntactic and semantic distance in the interface. Also, we dropped the original notion of a 'directness' axis in favor of a explicit representation of the *control granularity* of an interface. The design space can be used as an evaluative means as all axis are associated with a qualitative notion of 'better' or 'worse': Along the axes representing the semantic and syntactic distances, a position close to the origin, where these distances are small, is desirable. The control granularity axis can also be formulated in a way to associate interfaces that map syntactically atomic input gestures of operations to semantically atomic navigations with values close to the origin.

We proposed an interface model and design space that can describe and evaluate different types of navigation interfaces.

- *Media model*
  If we accept the idea that users navigate digital media in order to accomplish a certain goal, the need for an explicit representation of this goal in any theory to evaluate and design such interfaces becomes clear. We proposed a dual structure of form and content, syntactics and semantics, to represent any medium for which we want to model navigation. This allowed us to differentiate between syntactic and semantic navigation tasks or goals and to match these to the layer infrastructure of an interface. The media model also contains the *semantic mapping*, a description of how the content of the medium is distributed over its containing form. This mapping is an important part both for the evaluation of the semantic distances present in existing navigation interfaces and for the implementation of semantic navigation interfaces.

  Any description of navigation in digital media must encompass not only the interface but also the medium and the navigation tasks.

- *Combined model*
  We showed how our interface model and media model can be consolidated into a single combined model that is defined by the interface, the medium, and the semantic description of the navigation task. In this combined model, we could directly identify the semantic and syntactic interfaces and derive the control granularity, thus allowing to locate—and con-

  We proposed a combined interface and media model from which most properties of the navigation process can be derived.

sequently evaluate—such an interface in our design space. The *seven stages of action* can also be directly overlaid onto the combined model, which facilitates the identification of certain gulfs in media navigation interfaces. We then showed that this combined model possesses all characteristics that are needed for an *interaction model* [Beaudouin-Lafon, 2000] and thus can be used to describe and compare existing interfaces or to guide the creation of new interfaces.

- *Design guidelines and generative aspects*
  From the combined model we derived a sequence of four steps to guide the design of semantic navigation interfaces for digital media. Each step is concerned with a different part of the model; in concert, they describe a creation process that, ideally, arrives at a navigation interface close to the origin of the design space. The four steps are:
  - *identifying the conceptual model* for the task and deriving a suitable semantic structure from it,
  - *finding the semantic mapping* between that semantic structure and the syntactic structure of the medium,
  - *designing the interface* for accessing the semantic structure according to the key principles of direct manipulation, and
  - *evaluating the interface* with users to make sure that it benefits the user.

We proposed a sequence of four steps as a design guideline for the creation of semantic navigation interfaces.

- *Application examples*
  We demonstrated that the design approach of creating syntactic navigation interfaces for digital media by following the aforementioned four steps can be successfully applied to a wide range of different types of digital media. Our four example projects cover time-based as well as graphic- and text-based media, and we described the process of developing each project with a different emphasis on one of the four steps.

The proposed theoretical foundation was evaluated in four large example projects.

Apart from these contributions to interface design theory, the example applications each represent independent contributions to their respective fields and consequently have been published before as peer-reviewed notes, papers, and articles. These contributions are mostly summarized or

only referenced in the context of this thesis to allow more emphasis to be put on their purpose as illustrations of the individual apects of the theoretical framework and the proposed interaction model.

PERSONAL ORCHESTRA is an example for the technical complexity of creating an algorithmic implementation of the semantic mapping.

PERSONAL ORCHESTRA served as an introductory example, showing that even if the semantic structure is just a non-linear re-parametrization of the syntactic structure, the resulting navigation possibilities can give rise to novel ways of interacting with a medium. The focus of the discussion lay on the creation of the semantic mapping as the second step in our four-step-approach to the design of navigation interfaces: While the initial mapping, which annotates the syntactic structure with locations in the semantic space, is created manually, the inversion of the mapping and its application to the actual medium were a challenge in signal processing theory and implementation. The result is a system that allows an—at the time—novel kind of direct manipulation playback of orchestral music where each beat can be placed individually in time without corrupting the listening experience.

DRAGON shows how different designs for the semantic and syntactic layers of the interfaces can change users' interaction strategies and circumvent mathematical limitations of the semantic mapping.

DRAGON advanced the concept of semantic media navigation by presenting the possibility of having a dynamic set of semantic structures that allows different simultaneous semantic mappings. Navigation tasks that are concerned with multiple objects of interest are thus possible. In this chapter, we concentrated on the different design possibilities for video navigation interfaces once the conceptual model, the semantic structure, and the semantic mappings have been established. Minor changes to the interface mechanics, like, for example, the different distance measures that govern the direct manipulation of objects in the scene, were shown to cause very different effects in certain cases. We presented a number of refinements and extensions to the original interface and demonstrated how these can fix some of the shortcomings that can arise from the potentially singular nature of the semantic mappings.

FLY changed the syntactic layer of the medium itself; we demonstrated how such changes can be evaluated.

FLY is a software to create and navigate through presentation visuals and an example where the normal strategy of accessing the syntactic structure of the medium through the semantic mapping does not work as straightforward as in the previous applications. Instead of inverting an initial estimate of the semantic mapping's inverse, we re-defined the

format of the medium to contain an explicit syntactic representation of the semantic structure of the content. This, of course, led to a semantic mapping that is close to the identity, thus allowing for a much simpler design and implementation of the interface. Using FLY as an example, we illustrated in detail the fourth step of our design sequence—evaluating the interface—, which encompassed three different aspects of media navigation as seen by the three different user groups of the system.

The biggest challenge regarding STACKSPLORER and BLAZE was to find an appropriate conceptual model and semantic structure for the efficient navigation of source code.

STACKSPLORER and BLAZE served as examples for navigation in a medium that is highly structured and has a very tight coupling between the semantics of the content and its syntactic form. This allows an easy retrieval of the semantic mapping, and, consequently, the challenges rather lay with identifying suitable semantic substructures that represent the relevant parts of the users' conceptual model. Therefore, both tools facilitate navigation in and access to topologically different parts of the call graph. Our analysis showed how these particular choices for the semantic structure together with well-designed interfaces can help developers to increase both success rates and task performance times over existing call-graph-based interfaces for certain code maintenance tasks.

7.2 Future Work

With this thesis, we have developed a theoretical framework for interfaces that allow semantic navigation in digital media, and we have validated the concept by successfully applying it to four different types of media. Still, there is room for future extensions and improvements in many areas of this work—in the theoretical foundation as well as in the individual application areas.

Semantic scoping remains an issue to be investigated further.

Regarding the idea and concept of semantic media navigation, it would be necessary to investigate the notion of semantic scoping with respect to our interaction model. While, for example, we have applied the model to enable semantic video navigation inside a single scene, and although it would be relatively easy to come up with other semantic structures for longer movies (cf. Figure 1.2), the problem of

letting the user specify the semantic scope remains open. In the future, we thus have to develop a theory of how semantic scopes can be structured, parametrized, and represented in the interface.

We have not investigated all implications when shifting the active domain during direct manipulation interactions.

Another area that could be explored is the idea of dynamic domain switching in the context of direct manipulation interaction. We have touched this briefly when designing ways to avoid the navigation singularities that arise in the presence of temporal ambiguities in our semantic video navigation systems (cf. 4.2.3 "Common Interaction Problems" (p. 136)). While our proposed solution to have the manipulated domain dynamically shift from spatial to temporal and back solved a specific problem in a specific context, the implications of this concept on the theoretical basis of direct manipulation should be investigated.

The theoretical foundation could potentially be applied to other types of interfaces besides navigation.

Generally, it remains to be seen if our interaction model could be extended to be applicable past the field of media navigation alone. We have already partly adapted the concept to the creation process of medial content when designing FLY (cf. 5 "Hybrid Media: Presentation Visuals" (p. 185)), and, very recently, the trajectory-based direct manipulation interaction of *DRAGON* (cf. 4 "Time-based Media: Video Scenes" (p. 89)) has been extended to create dynamic video overpaintings [Santosa et al., 2013]. Similarly, many of the prototypes of interaction visionary Brett Victor[1] utilize the same ideas and make the semantic structure of media or source code directly accessible and modifiable. And lastly, our DRAGIMATION project enabled animators to interactively define the timing for an animation, which is also closer to the process of authoring media than to navigating existing content. These examples are hopefully just the first steps in developing a new class of semantic content creation and manipulation interfaces.

All four projects open up additional research questions in their respective fields.

Each of our four example applications for semantic navigation in digital media, PERSONAL ORCHESTRA, DRAGON, FLY, and STACKSPLORER, of course, also opens up its own field of research questions to be explored: For audio navigation, other more accessible interaction metaphors than conducting remain to be developed, and the quality of the gesture detection and the time-stretching could be improved.

[1] www.worrydream.com

The computer vision approaches in DRAGON to generate object trajectories in videos need to be improved in terms of speed and reliability, and they need to be more robust when tracking objects across different shots and scenes. Video navigation also seems to be a suitable candidate to investigate how interfaces for a wider range of semantic scopes can be designed. With FLY, an analysis of the user experience for the presenter—especially with the recently developed mobile version of the system—remains do be done. At the same time, the current implementation is still missing some important features that would allow a fairer comparison with commercial systems, such as animated builds or the possibility to have branching presentation paths. Finally, our work on STACKSPLORER just has opened up the opportunity to gain deeper insight into developers' source code navigation patterns and understanding strategies (cf. [Krämer et al., 2013]). Long-term user tests, and therefore the implementation of a stable version that can be distributed in the field, would be the next logical steps.

7.3 Closing Remarks

The consumption but also the creation of digital media is one of the central activities users of computers or mobile devices engage in, and both of these activities heavily depend on media navigation. The majority of navigation interfaces, however, has inherited the technical limitations of their analog ancestors, representing the position inside a medium only by means of the syntactic structure. While these interfaces are functionally sufficient to navigate through a medium, we argue that semantic navigation interfaces can be created that leverage the capabilities of today's devices to provide faster, more direct, and more enjoyable means of navigation. We hope that the theoretical framework for such content-based navigation that is provided in this thesis together with the detailed application examples enables researchers and software developers to help design and create a new class of navigation and editing interfaces: one where content comes before format and where we can directly interact with our objects of interest regardless of their medial container.

Appendix A

Questionnaires

This appendix contains excerpts of some the questionnaires that are referred to in the body of the thesis. These are meant as a quick reference for better readability, the full questionnaires can be found in the original papers or thesis documents.

regarding the slides	**absolutely not**		**unsure**		**very much**
Did you feel that the size of the slides negatively limited the way you wanted to do your presentation?	☐	☐	☐	☐	☐
Did you feel positively guided by the slide size?	☐	☐	☐	☐	☐
regarding the plane					
Before putting information on the plane, did you feel lost in the big free space?	☐	☐	☐	☐	☐
Do you feel that your final result looks messy?	☐	☐	☐	☐	☐
comparison	**slides**		**no preference**		**plane**
Was it easier for you to express your ideas on the unlimited plane or the slides?	☐		☐		☐
As a presenter, what would you prefer for your real presentations?	☐		☐		☐

Figure A.1: Questionnaire for the qualitative analysis of the authoring experience with the *Fly* paper prototype. Source: [Lichtschlag, 2008]

old set	new set	
Content ordered by time when possible without conflicts		1
Perfectly time ordered content		1
Alto/Lisa relationship	NEXTSTEP, Macintosh and Mac Os X	1
Lisa/Classic Macintosh relationship	UNIX and NEXTSTEP relationship	0.5
Alto/Star relationship	Windows 1.0 and Mac relationship	0.5
Systems from Apple together	Systems from Apple together	1
Systems from PARC together	Open Source systems together	1
Systems ordered by success	Systems ordered by success	1

Table A.1: Scoring points for the paper prototypes. Points for relationships were awarded when a visual connection between the items was clear, for example, by proximity, by a drawn line, or by grouping. Source: [Lichtschlag, 2008]

.

Attitude Questions	
A1 *(5-point, 1=fully agree)*	The presentation was interesting.
A2 *(5-point, 1=fully agree)*	I liked the presentation's visuals.
A3 *(5-point, 1=fully agree)*	I liked the presentation's commentary.
A4 *(5-point, 1=fully agree)*	I liked the presentation overall.

Satisfaction Questions	
S1 *(5-point, 1=fully agree)*	The presentation was comprehensible.
S2 *(5-point, 1=fully agree)*	The presentation did not loose me. I was never disoriented.
S3 *(3-point, 1=too slow)*	How was the speed of the presentation?
S4 *(3-point, 1=too little)*	How was the amount of content shown at once?
S5 *(5-point, 1=very often)*	The visuals distracted me from the spoken narration.
S6 *(5-point, 1=fully agree)*	I had sufficient time to look at all the content on the screen.
S7 *(5-point, 1=fully agree)*	The presentation's structure was easy to understand.
S8 *(5-point, 1=fully agree)*	I always knew which part of the presentation was currently shown.
S9 *(5-point, 1=fully agree)*	I always knew approximately how far advanced the presentation was.

Learning Questions	
L1 *(1=visuals, 2=narration, 3=both)*	Where did you get most of the information from?
L2 *(5-point, 1=fully agree)*	I remembered information based on its spatial location. *(Only for the Fly presentation.)*

Open Questions	
O1	What did you like/dislike about the presentation?

Table A.2: English translation of the questionnaire for the qualitative analysis of the learning experience with *Fly* as compared to *PowerPoint*. Source: [Hess, 2011]

Statement	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
I am satisfied with the resulting PowerPoint document.					
I am satisfied with the resulting Fly document.					

Statement	PowerPoint	Neither	Fly
It was easier for me to express myself with...			
Overall for my real presentations I would prefer...			

Figure A.2: Excerpt (questions **Q7–10**) from the questionnaire for the qualitative analysis of the authoring experience with *Fly* as compared to *PowerPoint*. Original source: [Lichtschlag, 2008]

Bibliography

A history of windows—highlights from the first 25 years. http://windows.microsoft.com/en-US/windows/history.

J. Accot and S. Zhai. Beyond fitts' law: models for trajectory-based hci tasks. In CHI '97: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 295–302, New York, NY, USA, 1997. ACM.

J. G. Adair. The hawthorne effect: A reconsideration of the methodological artifact. *Journal of Applied Psychology*, 69 (2):334–345, May 1984.

C. Ahlberg and B. Shneiderman. The alphaslider: a compact and rapid selector. In CHI '94: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 365–371, New York, NY, USA, 1994. ACM.

A. Amidi. *The Art of Pixar—The Complete Color Scripts and Select Art from 25 Years of Animation*. Chronicle Books, 2011.

R. Anderson, P. Davis, N. Linnell, C. Prince, V. Razmo, and F. Videon. Classroom presenter: Enhancing interactive education with digital ink. *Computer*, 40(9):56 –61, sept. 2007.

R. Anderson, R. Anderson, B. Simon, S. A. Wolfman, T. VanDeGrift, and K. Yasuhara. Experiences with a tablet pc based lecture presentation system in computer science courses. In SIGCSE '04: *Proceedings of the 35th SIGCSE technical symposium on Computer Science Education*, pages 56–60, New York, NY, USA, 2004. ACM.

C. R. Argueta, C.-J. Ko, and Y.-S. Chen. Interacting with a music conducting system. In *Proceedings of the 13th International Conference on Human-Computer Interaction. Part II:*

Novel Interaction Methods and Techniques, pages 654–663, Berlin, Heidelberg, 2009. Springer-Verlag.

M. Arment. Overdoing the interface metaphor. http://www.marco.org/2010/03/11/overdoing-the-interface-metaphor, March 2010.

Y. Ayatsuka, J. Rekimoto, and S. Matsuoka. Popup vernier: a tool for sub-pixel-pitch dragging with smooth mode transition. In UIST '98: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 39–48, New York, NY, USA, 1998. ACM.

E. Babic. Ethnografische Videodokumentationen in der Pharma- und Gesundheitsbranche. *media spectrum*, Special: Planungshandbuch Healthcare, Pharma & OTC, November 2010.

R. A. Ballagas. *Bringing Iterative Design to Ubiquitous Computing*. PhD thesis, RWTH Aachen University, 2007.

D. H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, January 1981.

A. Bangor, P. Kortum, and J. Miller. An Empirical Evaluation of the System Usability Scale. *Intl. Journal of Human-Computer Interaction*, 24(6), August 2008.

C. Barnes, D. B. Goldman, E. Shechtman, and A. Finkelstein. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics—Proceedings of SIGGRAPH*, 29(3), August 2010.

P. Baudisch and R. Rosenholtz. Halo: a technique for visualizing off-screen objects. In CHI '03: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 481–488, New York, NY, USA, 2003. ACM.

M. Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In CHI '00: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 446–453, New York, NY, USA, 2000. ACM.

B. B. Bederson. The promise of zoomable user interfaces. In VINCE '10: *Proceedings of the 3rd International Symposium on Visual Information Communication*, pages 2:1–2:1, New York, NY, USA, 2010. ACM.

B. B. Bederson and J. D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In UIST '94: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 17–26, New York, NY, USA, 1994. ACM.

C. Bemtgen. Reciting canvas presentations with an ipad. Bachelor's Thesis, RWTH Aachen University, 2012.

E. W. Birss. The integrated software and user interface of apple's lisa. In AFIPS '84: *Proceedings of the national computer conference and exposition*, pages 319–328, New York, NY, USA, 1984. ACM.

B. W. Boehm. Software engineering. *IEEE Transactions on Computers*, 25(12), November 1976.

J. Bonada. Automatic technique in frequency domain for near-lossless time-scale modification of audio. In ICMC'00: *Proceedings of the International Computer Music Conference*, 2000.

J. Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, Ltd, 2001.

J. Borchers, W. Samminger, and M. Mühlhäuser. Personal Orchestra: conducting audio/video music recordings. In WEDELMUSIC'02: *Proceedings of the Second international conference on Web delivering of music*, pages 93–100, Washington, DC, USA, 2002. IEEE Computer Society.

J. Borchers, E. Lee, W. Samminger, and M. Mühlhäuser. Personal Orchestra: A real-time audio/video system for interactive conducting. *ACM Multimedia Systems Journal Special Issue on Multimedia Software Engineering*, 9(5):458–465, March 2004.

J. Borchers, A. Hadjakos, and M. Mühlhäuser. Micon: A music stand for interactive conducting. In NIME 2006: *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 254–259, Paris, France, June 2006.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

C. Brockly. Evaluation of direct manipulation techniques for in-scene video navigation. Master's Thesis, RWTH Aachen University, 2009.

J. Brooke. SUS: a quick and dirty usability scale. *Usability evaluation in industry*, 1996.

R. Brooks. Towards a Theory of the Comprehension of Computer Programs. *International Journal of Man-Machine Studies*, 18(6), 1983.

P. M. Brossier, J. P. Bello, and M. D. Plumbley. Fast labelling of notes in music signals. In ISMIR 2004: *Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain, October 2004.

A. L. Brown. Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *The Journal of the Learning Sciences*, 2 (2):141–178, 1992.

V. Brown. The Power of Powerpoint: Is It in the User or the Program?. *Childhood Education*, 83(4):3, 2007.

T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In T. Pajdla and J. Matas, editors, *Computer Vision - ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin / Heidelberg, 2004.

D. Buchla. Buchla electronic musical instruments: Lightning III. http://www.buchla.com/lightning3.html.

D. Buckingham. Defining digital literacy. *Digital Kompetanse*, 1:263–276, April 2006.

P. L. Buttigieg. Perspectives on presentation and pedagogy in aid of bioinformatics education. *Briefings in Bioinformatics*, 11(6):587–597, 2010.

W. Buxton. Lexical and pragmatic considerations of input structures. *SIGGRAPH Computer Graphics*, 17(1):31–37, January 1983.

W. Buxton. Chunking and phrasing and the design of human-computer dialogues. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, editors, *Human-computer interaction*, pages 494–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

W. Buxton, M. Billinghurst, Y. Guiard, A. Sellen, and S. Zhai. *Human Input to Computer Systems: Theories, Techniques and Technology*. (in draft at http://www.billbuxton.com/inputManuscript.html).

T. Buzan. *The Mind Map Book*. Penguin Books, New York, NY, USA, 1991.

S. Card, T. P. Moran, and A. Newell. The model human processor: An engineering model of human performance. K. R. Boff, L. Kaufman, and J. P. Thomas, editors, *Handbook of Perception and Human Performance*, volume 2: Cognitive Processes and Performance, pages 1–35, New York, 1986. Wiley and Sons.

S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In CHI '91: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 181–186, New York, NY, USA, 1991. ACM.

M. Carnot, B. Dunn, A. Cañas, P. Gram, and J. Muldoon. Concept Maps vs. Web Pages for Information Searching and Browsing. *Institute for Human and Machine Cognition, University of West Florida.*, 2001.

G. Casiez and N. Roussel. No more bricolage!: methods and tools to characterize, replicate and compare pointing transfer functions. In UIST '11: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 603–614, New York, NY, USA, 2011. ACM.

M. Chau. Computer Supported Concept Maps: Excellent Tools for Enhancing Library Workshop Presentations. *LIBRES: Library and Information Science Research Electronic Journal*, 8(2), September 1998.

K. Chen and V. Rajlich. Case study of feature location using dependence graph. In IWPC 2000: *Proceedings of the 8th International Workshop on Program Comprehension*, Limerick, Ireland, June 2000. IEEE Computer Society.

L. Chen, M. T. Özsu, and V. Oria. Symbolic representation and retrieval of moving object trajectories. In MIR '04: *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 227–234, New York, NY, USA, 2004. ACM.

R. E. Clark. Reconsidering Research on Learning from Media. *Review of Educational Research*, 53(4):445, 1983.

R. E. Clark. Media will never influence learning. *Educational Technology Research and Development*, 42(2):21–29, 1994.

R. E. Clark. *Learning from Media*. Information Age Publishing, 2001.

D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, May 2003.

C. L. Corritore and S. Wiedenbeck. An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Humand-Computer Studies*, 54(1):1–23, January 2001.

C. Corsten. Dragonfly - reviewing lecture recordings with spatial navigation. Bachelor's Thesis, RWTH Aachen University, 2009.

R. Craig and J. Amernic. PowerPoint Presentation Technology and the Dynamics of Teaching. *Innovative Higher Education*, 31(3):147–160, 2006.

R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In VLHCC'05: *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2005a.

R. DeLine, A. Khella, M. Czerwinski, and G. Robertson. Towards understanding programs through wear-based filtering. In SoftVis '05: *Proceedings of the 2005 ACM Symposium on Software visualization*, pages 183–192, New York, NY, USA, 2005b. ACM.

R. Deline, M. Czerwinski, B. Myers, G. Venolia, S. Drucker, and G. Robertson. Code thumbnails: Using spatial memory to navigate source code. In VLHCC'06: *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2006.

A. Dillon, R. J. Spiro, J. T. Levonen, and J.-F. Rouet, editors. *HyperText and Cognition*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1st edition, 1996.

P. Dragicevic, G. Ramos, J. Bibliowitcz, D. Nowrouzezahrai, R. Balakrishnan, and K. Singh. Video browsing by direct manipulation. In CHI '08: *Proceedings of the SIGCHI*

conference on Human Factors in Computing Systems, pages 237–246, New York, NY, USA, 2008. ACM.

S. M. Drucker, G. Petschnigg, and M. Agrawala. Comparing and managing multiple versions of slide presentations. In UIST '06: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 47–56, New York, NY, USA, 2006. ACM.

C. Duxbury, M. Davies, and M. Sandler. Separation of transient information in musical audio using multiresolution analysis techniques. In DAFX'01: *Proceedings of the COST G-6 Conference on Digital Audio Effects*, Limerick, Ireland, December 2001.

W. J. Earnest. *Developing Strategies to Evaluate the Effective Use of Electronic Presentation Software in Communication Education*. PhD thesis, The University of Texas at Austin, 2003.

S. G. Eick, J. L. Steffen, and E. E. Sumner Jr. Seesoft - A Tool for Visualizing Line Oriented Software Statistics. *IEEE Transactions on Software Engineering*, 18(11), 1992.

A. Ekin, A. M. Tekalp, and R. Mehrotra. Automatic soccer video analysis and summarization. *IEEE Transactions on Image Processing*, 12(7):796–807, July 2003.

R. B. Ekstrom, J. W. French, and H. H. Harman. Cognitive factors: Their identification and replication. *Multivariate Behavioral Research Monographs*, 79(2), 1979.

L. Ellis and D. Mathis. College student learning from televised versus conventional classroom lectures: A controlled experiment. *Higher Education*, 14(2):165–173, 1985.

D. C. Engelbart and W. K. English. A research center for augmenting human intellect. *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, volume 33, pages 395–410, 1968.

L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17 –23, may/jun 2000.

D. Farkas. Understanding and using PowerPoint. *Proceedings of the STC Annual Conference*, volume 3, pages 312–320, 2005.

L. Favalli, A. Mecocci, and F. Moschetti. Object tracking for retrieval applications in mpeg-2. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(3):427 –432, apr 2000.

T. Feldman. *an Introduction to Digital Media*. Routledge, 1997.

C. J. Fillmore. Form and meaning in language: Volume I, Papers on semantic roles. *Volume 121 of CSLI lecture notes*, 2003.

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

J. L. Flanagan and R. M. Golden. Phase vocoder. *The Bell System Technical Journal*, 45:1493–1509, November 1966.

J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1997.

A. Fouse, N. Weibel, E. Hutchins, and J. D. Hollan. Chronoviz: a system for supporting navigation of time-coded data. In CHI EA '11: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 299–304, New York, NY, USA, 2011. ACM.

P. Freiberger and M. Swaine. *Fire in the valley: the making of the personal computer*. Osborne/McGraw-Hill, Berkeley, CA, USA, 1984.

D. M. Frohlich. The history and future of direct manipulation. *Behaviour & information technology*, 12(6):315–329, November 1993.

G. W. Furnas. Generalized fisheye views. In CHI '86: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 16–23, New York, NY, USA, 1986. ACM.

G. W. Furnas and X. Zhang. Muse: a multiscale editor. In UIST '98: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 107–116, New York, NY, USA, 1998. ACM.

S. Galic and S. Loncaric. Spatio-temporal image segmentation using optical flow and clustering algorithm. In IWISPA'00: *Proceedings of the First International Workshop on Image and Signal Processing and Analysis*, pages 63 –68, 2000.

J. J. Gibson. *The perception of the visual world*. Houghton-Mifflin, 1950.

A. Girgensohn, F. Shipman, A. Dunnigan, T. Turner, and L. Wilcox. Support for effective use of multiple video streams in security. In VSSN '06: *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 19–26, New York, NY, USA, 2006. ACM.

A. Girgensohn, D. Kimber, J. Vaughan, T. Yang, F. Shipman, T. Turner, E. Rieffel, L. Wilcox, F. Chen, and T. Dunnigan. Dots: support for effective video surveillance. In MULTIMEDIA '07: *Proceedings of the 15th international conference on Multimedia*, pages 423–432, New York, NY, USA, 2007. ACM.

D. B. Goldman, C. Gonterman, B. Curless, D. Salesin, and S. M. Seitz. Video object annotation, navigation, and composition. In UIST '08: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 3–12, New York, NY, USA, 2008. ACM.

D. R. Goldman. *A Framework for Video Annotation, Visualization, and Interaction*. PhD thesis, University of Washington, 2007.

L. Good. *Zoomable User Interfaces for the Authoring and Delivery of Slide Presentations*. PhD thesis, University of Maryland, 2003.

L. Good and B. Bederson. CounterPoint: Creating Jazzy Interactive Presentations. Technical report, 2001.

L. Good and B. B. Bederson. Zoomable user interfaces as a medium for slide show presentations. *Information Visualization*, 1(1):35–49, March 2002.

K. Gopal and K. Morapakkam. Incorporating concept maps in a slide presentation tool for the classroom environment. In ED-MEDIA'02: *Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications.*, Denver, Colorado, June 2002. AACE.

J. Gosling, D. S. H. Rosenthal, and M. J. Arden. *The NeWS book: an introduction to the network/extensible window system*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

M. Goto. An audio-based real-time beat tracking system for music with or without drum-sounds. *Jornal of New Music Research*, 30(2):159–171, 2001.

J. Gould. Some psychological evidence on how people debug computer programs. *International Journal of Man-Machine Studies*, 7(2):151–182, 1975.

G. Goyal, V. Prakash, and S. S. Manvi. Usage of concept maps in dynamic content presentation for online learning system. In A. J. Cañas and J. D. Novak, editors, *Proceedings of the Second International Conference on Concept Mapping*, San José, Costa Rica, 2006.

B. Grad. The creation and the demise of visicalc. *IEEE Annals of the History of Computing*, 29(3):20–31, July 2007.

I. Gruell. conga: A conducting gesture analysis framework. Master's thesis, University of Ulm, 2005.

F. Guimbretiére and T. Winograd. Flowmenu: combining command, text, and data entry. In UIST '00: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 213–216, New York, NY, USA, 2000. ACM.

S. Gustafson, P. Baudisch, C. Gutwin, and P. Irani. Wedge: clutter-free visualization of off-screen locations. In CHI '08: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 787–796, New York, NY, USA, 2008. ACM.

F. Halasz and T. P. Moran. Analogy considered harmful. In CHI '82: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 383–386, New York, NY, USA, 1982. ACM.

F. Hammer. Time-scale modification using the phase vocoder. Master's thesis, Institute for Electronic Music and Acoustics (IEM), Graz University of Music and Dramatic Arts, 2001.

R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

I. Herman, G. Melancon, and M. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), 2000.

T. Hess. Fly - expressive and conveying planar presentations. Master's thesis, RWTH Aachen University, 2011.

E. Hill, L. Pollock, and K. Vijay-Shanker. Exploring the neighborhood with dora to expedite software maintenance. In ASE '07: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 14–23, New York, NY, USA, 2007. ACM.

D. Holman, P. Stojadinović, T. Karrer, and J. Borchers. Fly: an organic presentation tool. In CHI EA '06: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 863–868, New York, NY, USA, 2006. ACM.

N. Holmes. In defense of PowerPoint. *Computer*, 37(7):100, 2004.

B. K. Horn and B. G. Schunck. Determining optical flow. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1980.

K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Transactions on Computer-Human Interaction*, 9(4):362–389, December 2002.

L. J. Hornbeck. From cathode rays to digital micromirrors: A history of electronic projection display technology. *TI Technical Journal*, pages 7–46, July–September 1998.

R. House, A. Watt, and J. Williams. Work in Progress - What is PowerPoint? Educating Engineering Students in

its Use and Abuse. In *Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference*. IEEE, 2005.

J. Hoyt. Does the delivery method matter?: Comparing technologically delivered distance education with on-campus instruction. Technical report, Utah Valley State College, Department of Institutional Research, 1999.

W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3):334 –352, aug. 2004.

M. Huang, P. Eades, J. Wang, and P. R. China. Online Animated Graph Drawing Using a Modified Spring Algorithm. *Journal of Visual Languages and Computing*, 9(6), 1998.

W. Hürst and G. Götz. Interface designs for pen-based mobile video browsing. In DIS '08: *Proceedings of the 7th ACM conference on Designing interactive systems*, pages 395–404, New York, NY, USA, 2008. ACM.

W. Hürst and P. Jarvers. Interactive, dynamic video browsing with the zoomslider interface. In ICME 2005: *Proceedings of the IEEE International Conference on Multimedia and Expo*, July 2005.

W. Hürst and K. Meier. Interfaces for timeline-based mobile video browsing. In MULTIMEDIA '08: *Proceedings of the 16th ACM international conference on Multimedia*, pages 469–478, New York, NY, USA, 2008. ACM.

W. Hürst, G. Götz, and P. Jarvers. Advanced user interfaces for dynamic video browsing. In MULTIMEDIA '04: *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 742–743, New York, NY, USA, 2004a. ACM.

W. Hürst, T. Lauer, and G. Götz. An elastic audio slider for interactive speech skimming. In NordiCHI '04: *Proceedings of the third Nordic conference on Human-Computer Interaction*, pages 277–280, New York, NY, USA, 2004b. ACM.

W. Hürst, T. Lauer, and R. Kaschuba. Interfaces for interactive audio-visual media browsing. In MULTIMEDIA '06: *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 807–808, New York, NY, USA, 2006. ACM.

E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Human-Computer Interaction*, 1:311–338, 1985.

S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In UIST '11: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 559–568, New York, NY, USA, 2011. ACM.

G. James. *The Tao of Programming*. Info Books, 1986.

D. Janzen and K. De Volder. Navigating and querying code without getting lost. In AOSD '03: *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 178–187, New York, NY, USA, 2003. ACM.

W. L. Jenkins and M. B. Connor. Some design factors in making settings on a linear scale. *Journal of Applied Psychology*, 33(4):395–409, 1949.

K. Jensen and T. H. Andersen. Real-time beat estimation using feature extraction. In *Proceedings of the Computer Music Modeling and Retrieval Symposium, Lecture Notes in Computer Science*. Springer Verlag, 2003.

J. Johnson, T. L. Roberts, W. Verplank, D. C. Smith, C. H. Irby, M. Beard, and K. Mackey. The Xerox Star: A retrospective. *Computer*, 22(9):11–26, 28–29, September 1989.

J. A. Johnson and B. A. Nardi. Creating presentation slides: a study of user preferences for task-specific versus generic application software. *ACM Transations on Computer-Human Interaction*, 3(1):38–65, 1996.

K. Johnson and V. Sharp. Is PowerPoint Crippling Our Students? *Learning and Leading with Technology*, volume 33, pages 6–7, Eugene, OR, November 2005. International Society for Technology in Education (ISTE).

E. H. Joy II and F. E. Garcia. Measuring Learning Effectiveness: A New Look at No-Significant-Difference Findings. *Journal of Asynchronous Learning Networks*, 4(1):33–39, 2000.

T. Karrer. PhaVoRIT: A phase vocoder for real-time interactive time-stretching. Diploma Thesis, RWTH Aachen University, 2005.

T. Karrer, E. Lee, and J. Borchers. Phavorit: A phase vocoder for real-time interactive time-stretching. In ICMC '06: *Proceedings of the International Computer Music Conference*, pages 708–715, New Orleans, USA, November 2006.

T. Karrer, M. Weiss, E. Lee, and J. Borchers. Dragon: a direct manipulation interface for frame-accurate in-scene video navigation. In CHI '08: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 247–250, New York, NY, USA, 2008. ACM.

T. Karrer, J.-P. Krämer, J. Diehl, B. Hartmann, and J. Borchers. Stacksplorer: call graph navigation helps increasing code maintenance efficiency. In UIST '11: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 217–224, New York, NY, USA, 2011. ACM.

T. Karrer, M. Wittenhagen, and J. Borchers. Draglocks: handling temporal ambiguities in direct manipulation video navigation. In CHI '12: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 623–626, New York, NY, USA, 2012. ACM.

A. Kathrein. Event detection in videos based on object trajectories. Bachelor's Thesis, RWTH Aachen University, 2011.

B. W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In AOSD '05: *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM.

D. Kimber, T. Dunnigan, A. Girgensohn, F. Shipman, T. Turner, and T. Yang. Trailblazing: Video playback control by direct object manipulation. In ICME'07: *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 1015 –1018, July 2007.

J. Kjeldsen. The Rhetoric of PowerPoint. *Seminar.net, International journal of media, technology and lifelong learning*, 2: 1–17, 2006.

A. Ko, H. H. Aung, and B. Myers. Eliciting design requirements for maintenance-oriented ides: a detailed study of corrective and perfective maintenance tasks. In ICSE '05: *Proceedings of the 27th International Conference on Software Engineering*, pages 126 – 135, May 2005.

A. Ko, B. Myers, M. Coblenz, and H. Aung. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering*, 32(12), 2006.

J. Koenemann and S. P. Robertson. Expert problem solving strategies for program comprehension. In CHI '91: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 125–130, New York, NY, USA, 1991. ACM.

R. B. Kozma. Learning with Media. *Review of Educational Research*, 61(2):179, 1991.

R. B. Kozma. Will media influence learning? Reframing the debate. *Educational Technology Research and Development*, 42(2):7–19, 1994.

J.-P. Krämer. Stacksplorer: Understanding dynamic program behavior. Diploma Thesis, RWTH Aachen University, 2011.

J.-P. Krämer, T. Karrer, J. Diehl, and J. Borchers. Stacksplorer: understanding dynamic program behavior. In UIST '10: *Adjunct proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 433–434, New York, NY, USA, 2010. ACM.

J.-P. Krämer, J. Kurz, T. Karrer, and J. Borchers. Blaze: supporting two-phased call graph navigation in source code. In CHI EA '12: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 2195–2200, New York, NY, USA, 2012. ACM.

J.-P. Krämer, T. Karrer, and J. Kurz. How tools in IDEs shape developers' navigation behavior. To appear in CHI '13: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 2013.

G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object Oriented Programming*, 1(3): 26–49, August 1988.

J. Kurz. Blaze Navigating Source Code via Call Stack Contexts. Bachelor's Thesis, RWTH Aachen University, 2011.

B. c. Kwon, W. Javed, N. Elmqvist, and J. S. Yi. Direct manipulation through surrogate objects. In CHI '11: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 627–636, New York, NY, USA, 2011. ACM.

J. Laroche and M. Dolson. Phase vocoder: About this phasiness business. In ASSP '97: *Proceedings of the IEEE Workshop on application of signal processing to audio and acoustics*, New Paltz, NY, 1997.

J. Laroche and M. Dolson. Improved phase vocoder timescale modification of audio. *IEEE Transactions on Speech and Audio Processing*, volume 7, pages 323–332, May 1999.

T. D. LaToza and B. A. Myers. Searching across paths. In SUITE '10: *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, pages 29–32, New York, NY, USA, 2010a. ACM.

T. D. LaToza and B. A. Myers. Developers ask reachability questions. In ICSE '10: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 185–194, New York, NY, USA, 2010b. ACM.

J. Lawrance, R. Bellamy, and M. Burnett. Scents in programs:does information foraging theory apply to program maintenance? In VLHCC '07: *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 15 –22, sept. 2007.

J. Lawrance, R. Bellamy, M. Burnett, and K. Rector. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In CHI '08: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1323–1332, New York, NY, USA, 2008. ACM.

J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. D. Fleming. How Programmers Debug, Revisited: An Information Foraging Theory Perspective. *IEEE Transactions on Software Engineering*, PP(99):1–1, February 2010.

E. Lee. *A Semantic Time Framework for Interactive Media Systems*. PhD thesis, RWTH Aachen University, 2007.

E. Lee and J. Borchers. The role of time in engineering computer music systems. In NIME '05: *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 204–207, Vancouver, Canada, May 2005.

E. Lee and J. Borchers. DiMaß: A technique for audio scrubbing and skimming using direct manipulation a technique for audio scrubbing and skimming using direct manipulation. In AMCMM '06: *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 107–114, New York, NY, USA, 2006. ACM.

E. Lee, T. M. Nakra, and J. Borchers. You're the conductor: A realistic interactive conducting system for children. In NIME '04: *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 68–73, Hamamatsu, Japan, June 2004.

E. Lee, T. Karrer, and J. Borchers. Toward a framework for interactive systems to conduct digital audio and video streams. *Computer Music Journal*, 30(1):21–36, Spring 2006a.

E. Lee, H. Kiel, S. Dedenbach, I. Grüll, T. Karrer, M. Wolf, and J. Borchers. isymphony: an adaptive interactive orchestral conducting system for digital audio and video streams. In CHI EA '06: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 259–262, New York, NY, USA, 2006b. ACM.

E. Lee, T. Karrer, and J. Borchers. An analysis of startup and dynamic latency in phase vocoder-based time-stretching algorithms. *Proceedings of the International Computer Music Conference*, volume 2 of *ICMC '07*, pages 73–80, Copenhagen, Denmark, August 2007.

S. Letovsky. Cognitive processes in program comprehension. *Journal of Systems and Software*, 7(4), 1987.

S. N. Levine and J. O. S. III. A sines+transients+noise audio representation for data compression and time/pitch scale modications. In *105th Audio Engineering Society Convention*, San Francisco, 1998.

D. Lewandowski. Camshift-Tracking mit Tiefeninformationen. Bachelor's Thesis, Fachhochschule Aachen, 2011.

J. Lewis and J. Sauro. The Factor Structure of the System Usability Scale. *LNCS: Human Centered Design*, 5619, 2009.

J. P. Lewis. Fast normalized cross-correlation. *Vision Interface*, 1995.

Y. Li, J. A. Landay, Z. Guan, X. Ren, and G. Dai. Sketching informal presentations. In ICMI '03: *Proceedings of the 5th international conference on Multimodal interfaces*, pages 234–241, New York, NY, USA, 2003. ACM.

L. Lichtschlag. Fly: An organic authoring tool for presentations. Diploma Thesis, RWTH Aachen University, 2008.

L. Lichtschlag, T. Karrer, and J. Borchers. Fly: a tool to author planar presentations. In CHI '09: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 547–556, New York, NY, USA, 2009. ACM.

L. Lichtschlag, T. Hess, T. Karrer, and J. Borchers. Fly: studying recall, macrostructure understanding, and user experience of canvas presentations. In CHI '12: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1307–1310, New York, NY, USA, 2012a. ACM.

L. Lichtschlag, T. Hess, T. Karrer, and J. Borchers. Canvas presentations in the wild. In CHI EA '12: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 537–540, New York, NY, USA, 2012b. ACM.

D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental models and software maintenance. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 80–98, Norwood, NJ, USA, 1986. Ablex Publishing Corp.

J. Lovgren. How to choose good metaphors. *IEEE Software*, 11(3):86 –88, May 1994.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2): 91–110, 2004.

B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In IJCAI'81: *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

F. Lv, X. Song, B. Wu, V. Kumar, and S. R. Nevatia. Left luggage detection using bayesian inference. In *Proceedings of the IEEE International Workshop on Performance and Evaluation of Tracking and Surveillance*, pages 83–90, 2006.

T. Marrin and R. Picard. The 'Conductor's Jacket': A Device for Recording Expressive Musical Gestures. In ICMC '98: *Proceedings of the International Computer Music Conference*, pages 215–219, 1998.

P. Masri and A. Bateman. Improved modelling of attack transients in music analysis-resynthesis. In ICMC '96: *Proceedings of the International Computer Music Conference*, 1996.

T. Masui, K. Kashiwagi, and G. R. Borden, IV. Elastic graphical interfaces to precise data manipulation. In CHI '95: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 143–144, New York, NY, USA, 1995. ACM.

L. Mathis. Realism in ui design. http://ignco.de/240, January 2010.

R. E. Mayer and J. K. Gallini. When is an illustration worth ten thousand words? *Journal of Educational Psychology*, 82 (4):715–726, December 1990.

C. McCarthy and N. Barnes. Performance of optical flow techniques for indoor navigation with a mobile robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 5 of *ICRA '04*, pages 5093–5098, April 2004.

M. McLuhan. *Understanding Media*. Routledge, 1964.

G. Medioni, I. Cohen, F. Bremond, S. Hongeng, and R. Nevatia. Event detection and analysis from video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):873–889, August 2001.

A. Michotte. *The Perception of Causality*. Basic Books, 1963.

R. B. Miller. Response time in man-computer conversational transactions. In AFIPS '68 (Fall, part I): *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, New York, NY, USA, 1968. ACM.

T. P. Moran. The Command Language Grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies*, 15 (1):3–50, July 1981.

T. Moscovich, K. Scholz, J. Hughes, and D. Salesin. Customizable Presentations. Technical report, Computer Science Dept., Brown University, 2004.

T. Munzner. H3: laying out large directed graphs in 3d hyperbolic space. In INFOVIS '97: *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–10, Washington, DC, USA, 1997. IEEE Computer Society.

B. A. Myers. A brief history of human-computer interaction technology. *interactions*, 5(2):44–54, March 1998.

L. Nelson, S. Ichimura, E. R. Pedersen, and L. Adams. Palette: a paper interface for giving presentations. In CHI '99: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 354–361, New York, NY, USA, 1999. ACM.

A. Nett. Finding cognitive strategies for navigating with 1D-controls. Diploma Thesis, RWTH Aachen University, 2012.

R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In ISMAR '11: *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127 –136, October 2011.

K. Nieuwenhuizen, D. Aliakseyeu, and J.-B. Martens. Insight into goal-directed movement strategies. In CHI '10: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 883–886, New York, NY, USA, 2010. ACM.

D. A. Norman. *The Psychology of Everyday Things*. Basic Books, New York, 1988.

D. A. Norman. In defense of PowerPoint. http://www.jnd.org/dn.mss/in_defense_of_p.html, 2005.

J. Novak. Concept Mapping: A Useful Tool for Science Education. *Journal of Research in Science Teaching*, 27(10):937–49, 1990.

J. D. Novak and D. B. Gowin. *Learning how to learn*. Cambridge University Press, 1984.

A. Novosad. Object selection and adaptive trajectories in dragon. Master's Thesis, RWTH Aachen University, 2012.

P. Oman. Maintenance tools. *IEEE Software*, 7(3):59–65, May 1990.

I. Parker. *Absolute PowerPoint: Can a software package edit our thoughts?* The New Yorker, 2001.

E. R. Pedersen, T. Sokoler, and L. Nelson. Paperbuttons: expanding a tangible user interface. In *DIS '00: Proceedings of the 3rd conference on Designing interactive systems*, pages 216–223, New York, NY, USA, 2000. ACM.

N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19, 1987.

K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64, New York, NY, USA, 1993. ACM.

D. Piorkowski, S. D. Fleming, C. Scaffidi, L. John, C. Bogart, B. E. John, M. Burnett, and R. Bellamy. Modeling programmer navigation: A head-to-head empirical evaluation of predictive models. In VLHCC '11: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 109–116, 2011.

D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy, and C. Swart. Reactive information foraging: an empirical investigation of theory-based recommender systems for programmers. In CHI '12: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1471–1480, New York, NY, USA, 2012. ACM.

P. Pirolli and S. Card. Information foraging in information access environments. In CHI '95: *Proceedings of the SIGCHI Conference on Human Factors in Computing*

Systems, pages 51–58, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

Plato. *Phaedrus*. 370 BC.

Plato. *Gorgias*. 380 BC.

G. Pólya. *How to Solve It*. Doubleday, Garden City, NY, 1957.

S. Pook. *Interaction and Context in Zoomable User Interfaces*. PhD thesis, Paris, France, 2001.

R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 7th edition, 2010.

M. Puckette. Phase-locked vocoder. In ASSP '95: *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, New York, 1995.

T. Ramage. The "No Significant Difference" Phenomenon: A Literature Review. *e-Journal of Instructional Science and Technology*, 5, 2002.

G. Ramos and R. Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. In UIST '03: *Proceedings of the annual ACM symposium on User Interface Software and Technology*, pages 105–114, New York, NY, USA, 2003. ACM.

J. Raskin. *The humane interface: new directions for designing interactive systems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

R. A. Reiser. Clark's invitation to the dance: An instructional Designer's response. *Educational Technology Research and Development*, 42(2):45–48, 1994.

M. Revelle, B. Dit, and D. Poshyvanyk. Using data fusion and web mining to support feature location in software. In ICPC '10: *Proceedings of the 18th IEEE International Conference on Program Comprehension*, pages 14 –23, 30 2010-july 2 2010.

H. Richter, J. Brotherton, and G. Abowd. A multi-scale timeline slider for stream visualization and control. Technical Report GIT-GVU-99-30, GVU Center, Georgia Institute of Technology, June 1999.

A. Röbel. Transient detection and preservation in the phase vocoder. In ICMC '03: *Proceedings of the International Computer Music Conference*, pages 247–250, Singapore, 2003a.

A. Röbel. A new approach to transient processing in the phase vocoder. In DAFx '03: *Proceedings of the 6th Int. Conference on Digital Audio Effects*, pages 344–349, London, UK, September 2003b.

M. P. Robillard, W. Coelho, and G. C. Murphy. How Effective Developers Investigate Source Code: An Exploratory Study. *IEEE Transactions on Software Engineering*, 30(12), 2004.

M. Rudolf. *The grammar of conducting: a practical guide to baton technique and orchestral interpretation*. Schirmer Books, 1980.

T. L. Russell. *The No Significant Difference Phenomenon: As Reported in 355 Research Reports, Summaries and Papers*. IDECC, 1999.

G. Salvendy. *Handbook of Human Factors and Ergonomics*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In CVPR '06: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, pages 2195–2202, Washington, DC, USA, 2006. IEEE Computer Society.

S. Santosa, F. Chevalier, R. Balakrishnan, and K. Singh. Direct space-time trajectory control for visual media editing. To appear in CHI '13: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 2013.

F. d. Saussure. Nature of the linguistics sign. In C. Bally and A. Sechehaye, editors, *Cours de linguistique générale*. McGraw Hill Education, 1916.

S. Saxena, F. Brémond, M. Thonnat, and R. Ma. Crowd behavior recognition for video surveillance. In ACIVS '08: *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 970–981, Berlin, Heidelberg, 2008. Springer-Verlag.

D. L. Schacter and L. Nadel. Varieties of spatial memory: A problem for cognitive neuroscience. *Perspectives on cognitive neuroscience*, pages 165–185, 1991.

R. W. Scheifler and J. Gettys. The X window system. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

B. J. Scholl and P. D. Tremoulet. Perceptual causality and animacy. *Trends in Cognitive Sciences*, 4(8):299 – 309, 2000.

R. Shah and P. Narayanan. Trajectory based video object manipulation. In ICME '11: *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 1 –4, july 2011.

F. M. Shipman, C. C. Marshall, R. Furuta, D. A. Brenner, H. wei Hsieh, and V. Kumar. Creating educational guided paths over the world-wide web. P. Carlson and F. Makedon, editors, *Proceedings of Educational Telecommunications*, volume 96 of *ED-TELECOM '96*, pages 326–331. Association for the Advancement of Computing in Education, June 1996.

F. M. Shipman, R. Furuta, D. Brenner, C.-C. Chung, and H. wei Hsieh. Using paths in the classroom: experiences and adaptations. In *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia*, pages 267–270, New York, NY, USA, 1998. ACM.

B. Shneiderman. Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences*, 5(2):123–143, June 1976.

B. Shneiderman. Measuring Computer Program Quality and Comprehension, Technical Repon No. 16, Department of Information Systems Management. *College Park, University of Maryland*, 1977.

B. Shneiderman. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Parallel Programming*, 1979.

B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & information technology*, 1(3):237–256, July 1982.

B. Shneiderman and P. Maes. Direct manipulation vs. interface agents. *interactions*, 4(6):42–61, November 1997.

B. Shneiderman and C. Plaisant. *Designing the User Interface*. Pearson Higher Education, international fifth edition, 2010.

N. T. Siebel and S. J. Maybank. The advisor vsual surveillance system. In ACV '04: M. Clabian, V. Smutny, and G. Stanke, editors, *Proceedings of the ECCV 2004 workshop "Applications of Computer Vision"*, pages 103–111, Prague, Czech Republic, May 2004.

B. Signer and M. C. Norrie. Paperpoint: a paper-based presentation and interactive paper prototyping tool. In TEI '07: *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 57–64, New York, NY, USA, 2007. ACM.

J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.

S. E. Sim, C. L. Clarke, and R. C. Holt. Archetypal source code searches: a survey of software developers and maintainers. In *Proceedings of the 6th International Workshop on Program Comprehension*. IEEE Computer Society, 1998.

J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An Examination of Software Engineering Work Practices. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, 1997. IBM Press.

J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software maintenance. In ICSM '05: *Proceedings of the 21st IEEE Conference on Software Maintenance*, pages 325–334. IEEE, September 2005.

A. Singh. *Mac OS X Internals: A Systems Approach*. Addison-Wesley Professional, 2006.

D. K. Smith and R. D. Alexander. *Fumbling the future: how Xerox invented, then ignored, the first personal computer*. William Morrow & Co., Inc., New York, NY, USA, 1988.

E. Soloway and K. Ehrlich. Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, SE-10(5), 1984.

E. Soloway, R. Lampert, S. Letovsky, D. Littman, and J. Pinto. Designing documentation to compensate for delocalized plans. *Communications of the ACM*, 31(11):1259–1267, November 1988.

C. Stauffer and W. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, August 2000.

R. J. Sternberg. *Cognitive Psychology*. Wadsworth Publishing, 2002.

M.-A. Storey, F. Fracchia, and H. Muller. Cognitive design elements to support the construction of a mental model during software visualization. In IWPC '97: *Proceedings of the Fifth International Workshop on Program Comprehension*, Dearborn, MI, 1997. IEEE Comput. Soc. Press.

I. E. Sutherland. Sketchpad: a man-machine graphical communication system. In AFIPS '63 (Spring): *Proceedings of the May 21-23, 1963, spring joint computer conference*, pages 329–346, New York, NY, USA, 1963. ACM.

S. C. L. Terra and R. A. Metoyer. A performance-based technique for timing keyframe animations. *Graph. Models*, 69 (2):89–105, March 2007.

S. R. Tilley, D. B. Smith, and S. Paul. Towards a framework for program understanding. In WPC '96: *Proceedings of the 4th International Workshop on Program Comprehension*, pages 19–28, Washington, DC, USA, 1996. IEEE Computer Society.

V. Tovinkere and R. Qian. Detecting semantic events in soccer games: towards a complete solution. In ICME '01: *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 833 –836, August 2001.

E. Tufte. *The Cognitive Style of PowerPoint*. Graphics Press, Cheshire, Connecticut, USA, 2003.

Unknown Author. *Rhetorica ad Herennium*. approx. 90 BC.

J. M. Utterback and W. J. Abernathy. A dynamic model of process and product innovation. *Omega*, 3(6):639 – 656, 1975.

J. Van Pelt. Lantern slides and such. *Quarterly Journal of Speech*, 36:44–50, 1950.

D. Čubranić and G. C. Murphy. Hipikat: Recommending Pertinent Software Development Artifacts. In ICSE '03: *Proceedings of the 25th International Conference of Software Engineering*, pages 408–418. IEEE, 2003.

V. Walsh and J. Kulikowski, editors. *Perceptual Constancy—Why Things Look as They do*. The press syndicate of the university of cambridge, 1998.

B. Walther-Franks, M. Herrlich, T. Karrer, M. Wittenhagen, R. Schröder-Kroll, R. Malaka, and J. Borchers. Dragimation: direct manipulation keyframe timing for performance-based animation. In GI '12: *Proceedings of the 2012 Graphics Interace Conference*, pages 101–108, Toronto, Ont., Canada, Canada, 2012. Canadian Information Processing Society.

M. Weiß. Depth-discontinuity preserving optical flow using time-multiplexed illumination. Diploma Thesis, RWTH Aachen University, University of Southern California, 2007.

D. Wiegmann, D. Dansereau, E. McCagg, K. Rewey, and U. Pitre. Effects of knowledge map characteristics on information processing. *Contemporary educational psychology*, 17(2):136–155, 1992.

M. Wittenhagen. Dragoneye - fast object tracking and camera motion estimation. Diploma Thesis, RWTH Aachen University, 2008.

D. Wixon, K. Holtzblatt, and S. Knox. Contextual design: an emergent view of system design. In CHI '90: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 329–336, New York, NY, USA, 1990. ACM.

R. S. Woodworth. Accuracy of voluntary movement. *The Psychological Review: Monograph Supplements*, 3(3):1–114, July 1899.

J. Wright. Notes from Left Field: Corporate Slide Presentations. *IEEE Computer Graphics and Applications*, 3(4):39–44, 1983.

C. Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/∼ccwu/siftgpu, 2006.

Y. Yanagisawa. Social behavior and mating system of the gobiid fish amblyeleotris japonica. *Japanese Journal of Ichthyology*, 28(4), 1982.

P. T. Zellweger. Scripted documents: a hypermedia path mechanism. In HYPERTEXT '89: *Proceedings of the second annual ACM conference on Hypertext*, pages 1–14, New York, NY, USA, 1989. ACM.

P. T. Zellweger, J. D. Mackinlay, L. Good, M. Stefik, and P. Baudisch. City lights: contextual views in minimal space. In CHI EA '03: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Extended Abstracts*, pages 838–839, New York, NY, USA, 2003. ACM.

H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425 – 432, April 1980.

B. Zipser. personal correspondence with Anne Kathrein.

D. E. Zongker and D. H. Salesin. On creating animated presentations. In SCA '03: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 298–308, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

Index

Curriculum Vitae

Personal Data	Thorsten Karrer Media Computing Group RWTH Aachen University
Telefone	+49 214 8021064
Email	karrer@cs.rwth-aachen.de
27-Jan-1980	Born in Hamburg, Germany
Jul 2000–Sep 2000	Internship at Siemens SIS, Munich, Germany
Oct 2000–Nov 2005	Diploma in Computer Science at RWTH Aachen University, Germany
Feb 2006–Aug 2006	Research Assistant at CSIRO Textile and Fibre Division, Geelong, Australia
Jun 2008–Aug 2008	Internship at Apple Inc., Cupertino, USA
Sep 2006–May 2013	Doctoral Candidate at the Media Computing Group, Department of Computer Science, RWTH Aachen University, Germany Advisor: Prof. Dr. Jan Borchers

Typeset July 9, 2013