Fachgebiet Software-Architekturen,
Hasso-Plattner-Institut für Softwaresystemtechnik,
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam,
Universität Potsdam

Masterarbeit

WebCards – Entwurf und Implementierung eines kollaborativen, graphischen Web-Entwicklungssystems für Endanwender

Julius Dannert
julius.dannert@student.hpi.uni-potsdam.de

13. November 2009

Betreuer:

Professor Dr. Robert Hirschfeld,

Jens Lincke

Bearbeitungszeitraum:

15.05.2009 – 15.11.2009

Überblick

Viele datenzentrierte Endanwender-Webanwendungen, wie Adressbücher oder Terminverwaltungen, lassen sich mithilfe von Wikis einfach realisieren. Allerdings haben Wikis dabei auch ihre Nachteile: Sie sind auf Freitext als Inhalt spezialisiert und bieten kaum Unterstützung für größere Mengen gleichartiger Datensätze. Des Weiteren ist es mit Wikis auf Grund ihrer Konzentration auf textuelle Inhalte nur schwer möglich, Informationen graphisch, also unter Verwendung von Position, Farbe, Größe u.ä., zu repräsentieren.

Dank Lively Kernel und dem darauf basierenden Lively Wiki können bereits jetzt Webanwendungen erstellt werden, bei denen der Benutzer vergisst, dass es sich um eine Anwendung im Browser handelt. Dies ist möglich, da eine mit Lively Kernel erstellte Webanwendung aus aktiven, graphischen Objekten, die direkt manipuliert werden können, besteht. Jedoch haben die Kollaborationsmöglichkeiten, die Lively Wiki bietet, ihre Beschränkungen. Wenn mehrere Benutzer gleichzeitig eine Seite bearbeiten, kann es bei der durch Lively Wiki ermöglichten asynchronen Kollaboration zu Konflikten kommen.

Auf Grund seiner Lebendigkeit und Direktheit stellt Lively Kernel eine gute Grundlage für Web-Entwicklung durch Endanwender dar, allerdings müssen vom Endanwender einige Hürden genommen werden, bis erste eigene Webanwendung erstellt werden können.

Um diese Barrieren zu senken, wird Lively Kernel im Rahmen dieser Arbeit um für Endanwender wichtige Funktionalitäten erweitert. Diese Erweiterungen umfassen ein Vorlagen-Konzept, die Möglichkeit zur synchronen Kollaboration, einen Undo-Mechanismus sowie automatische Persistenz.

Im Rahmen dessen wurde herausgefunden, dass all diese Erweiterungen von Lively Kernel mit Hilfe von Kommando-Objekten realisiert werden können. Die Verwendung der gleichen Kommando-Objekte für all diese Funktionalitäten bringt einige Vor- und Nachteile mit sich, die im Rahmen dieser Arbeit diskutiert werden.

Basierend auf den Erweiterungen von Lively Kernel wird ein Programm namens WebCards erstellt, das es Endanwendern ermöglicht kollaborativ datenzentrierte, graphische Endanwender-Webanwendungen einfach zu erstellen.

Abstract

Many data-centric end user web applications like address books or tools to arrange appointments can be build easily by using Wikis. But Wikis have some drawbacks: They are specialized in free text as main content and do not support a larger quantity of similar data records. Due to the concentration on textual content Wikis can hardly be utilized to display information by using graphical elements like position, colour or size.

Thanks to Lively Kernel and its extension Lively Wiki, it is already possible to build web applications which make the user forget that he is using an application in a browser. This is possible because web applications built with Lively Kernel consist of active, graphical objects, which can be manipulated directly. However, the collaboration capabilities offered by Lively Wiki have their limitations. If multiple users manipulate the same page concurrently, the asynchronous collaboration support can lead to conflicts.

Due to its liveness and directness Lively Kernel is a good foundation for end user web development, but at the moment they have to cope with several difficulties.

To tear down these barriers Lively Kernel is enhanced by important functionalities for end users in this thesis. These extensions include a template concept, the possibility to collaborate synchronously, an undo mechanism and automatic persistence.

In doing so, we discovered that all these extensions of Lively Kernel can be implemented by using Command-Objects. The usage of the same Command-Objects for all these functionalities has some advantages on the one hand and some disadvantages on the other hand, which are discussed in this thesis.

Based on the extensions of Lively Kernel an application named WebCards is built, that empowers end users to build data-centric, graphically end user web applications collaboratively in an easy way.

Inhaltsverzeichnis

Kapitel 1

Einführung

Das kollaborative Erstellen von datenzentrierten, graphischen Endanwender-Webanwendungen stellt auch heute noch eine Herausforderung dar. Wiki-Syteme haben das Erstellen und Bearbeiten von solchen Webanwendungen bereits erheblich vereinfacht, haben jedoch auch einige Schwächen. Sie ermöglichen asynchrone Kollaboration, nehmen dabei jedoch Konflikte durch gleichzeitige Bearbeitung in Kauf. Der Inhalt von Wikis besteht typischerweise aus Freitext. Größere Mengen von gleichartigen Datensätzen, wie Adressbucheinträge, lassen sich nur unzureichend handhaben. Auch die Möglichkeit, durch visuelle Eigenschaften, wie Farbe oder Position, Informationen auszudrücken, kommt in den auf Text konzentrierten Wikis zu kurz.

Diese Schwächen werden in dieser Arbeit angegangen, um das kollaborative Erstellen von datenzentrierten, graphischen Endanwender-Webanwendungen möglichst einfach zu gestalten. Mit Hilfe des im Rahmen dieser Masterarbeit entwickelten Programms, WebCards, kann ein einzelner Anwender oder auch eine Gruppe innerhalb kürzester Zeit eine Endanwender-Webanwendung erstellen, die sofort verwendet und jederzeit weiterentwickelt werden kann. Innerhalb der Webanwendung kann dabei nicht nur Text zum Darstellen von Daten verwendet werden, es stehen auch sehr viele graphische Möglichkeiten, wie Farbe oder Position, zur Verfügung.

Mit Lively Kernel lassen sich bereits jetzt interaktive Webanwendungen erstellen, die den Benutzer vergessen lassen, dass es sich um eine Anwendung im Browser handelt und nicht um eine Desktop-Anwendung (Mikkonen u. Taivalsaari, 2007). Dies wird dadurch erreicht, dass eine mit Lively Kernel erstellte Webanwendung nicht aus statischen Elementen besteht, sondern aus aktiven Objekten, die direkt manipuliert werden können (Taivalsaari u. a., 2008). Aufgrund seiner Direktheit und Lebendigkeit (Maloney u. Smith, 1995) bietet sich Lively Kernel als Autorenumgebung zur Erstellung von Webanwendungen durch Endanwender an.

In Lively Kernel können alle interaktiven graphischen Objekte, Morphs genannt, direkt bearbeitet werden, dies wird von Maloney u. Smith als *live editing* bezeichnet. Das *live editing* steht jedoch nicht bloß im Rahmen der Erstellung der UI zur Verfügung, sondern kann jederzeit verwendet werden und ist somit selbst Teil der UI. In einer auf Lively Kernel basierenden Webanwendung kann also genauso einfach die Farbe oder die Position eines Textfeldes verändert werden wie der textuelle Inhalt. Darauf basierend können Webanwendungen erstellt werden, bei denen das Verändern visueller Eigenschaften von Morphs explizit zum Darstellen von Informationen vorgesehen ist.

Einfache und trotzdem nützliche Anwendungen lassen sich mit Lively Kernel jedoch nicht so leicht erstellen, wie es wünschenswert wäre. Um dem Ziel „simple things should be simple“[1] näher zu kommen, wird deshalb Lively Kernel im Rahmen dieser Arbeit erweitert. Dabei wird das sehr erfolgreiche HyperCard als Ideengeber für WebCards verwendet. Hunderttausende Endanwender erstellten mit HyperCard Computer-Programme, die ihren persönlichen Bedürfnissen entsprachen (Goodman, 1998a). Begründen lässt sich dieser Erfolg damit, dass HyperCard einfache Konzepte verwendet und deshalb leicht zu erlernen ist. Trotz der Beschränkung auf einfache Konzepte können viele nützliche Programme erstellt werden. Insbesondere ist es aufgrund der mächtigen Hintergrund-Metapher möglich, ohne Nutzung von HyperTalk, der Programmiersprache von HyperCard, nützliche Programme zu erstellen.

Um das Erstellen von Webanwendungen möglichst einfach zu gestalten, werden die erfolgreichen und bewährten Konzepte von HyperCard, wie die Einteilung einer Anwendung in einen Stapel von Karten, in WebCards übernommen. Im Rahmen dessen wird Lively Kernel um ein Vorlagen-Konzept, das sich an der Hintergrund-Metapher von HyperCard orientiert, erweitert. Dabei stellt sich die Frage, wie die Hintergrund-Metapher aus HyperCard mit der Möglichkeit alle UI-Elemente direkt zu bearbeiten in Einklang zu bringen ist. Dazu wird auch die Frage beantwortet, welchen unterschiedlichen Zwecken die Hintergrund-Metapher in HyperCard dient. Ausgehend von diesen Fragen wird ein Vorlagen-Konzept entwickelt, das weit über die Möglichkeiten der Hintergrund-Metapher aus HyperCard hinausgeht und vielfältige Einsatzmöglichkeiten eröffnet.

Neben dem Vorlage-Konzept wird Lively Kernel weiterhin um eine einfache Möglichkeit zur synchronen Kollaboration, sowie einen Undo-Mechanismus erweitert. Endanwender profitieren erheblich, wenn Programme einen Undo-Mechanismus bieten, da die Anwender so die Möglichkeiten des Programms testen können, ohne ihre bisherigen Arbeitsergebnisse zu gefährden. Dies wird bei Anwendungen, die synchron genutzt werden, noch wichtiger, da Fehler eines einzelnen Benutzers sich auf alle Benutzer auswirken und zusätzlich kollaborative Anwendungen im Vergleich zu Programmen für Einzelnutzer mehr Raum für Entdeckungen bieten (Sun, 2002).

[1] Alan Kays: „Simple things should be simple, complex things should be possible.“ zitiert nach Leuf, B. und Cunningham, W. The Wiki way: quick collaboration on the Web, Addison-Wesley Longman Publishing Co., Inc., 2001

Für den Undo-Mechanismus und auch für die synchrone Kollaboration muss das Problem gelöst werden, dass Änderungen an UI-Elementen sowohl über die graphische Benutzerschnittstelle als auch durch Skripte möglich sind. Des Weiteren muss die Frage geklärt werden, wie damit umgegangen wird, dass visuelle Eigenschaften von graphischen Objekten teilweise UI- und teilweise Domain-Daten sind.

Bei vielen Webanwendungen, die durch WebCards ermöglicht werden, beispielsweise einer Anwendung zur Planung der Aufgabenverteilung innerhalb eines Teams, ist es gut vorstellbar, dass sie kollaborativ verwendet werden. Eine Gruppe von Menschen kann so, ohne sich zur gleichen Zeit am gleichen Ort befinden zu müssen, die Zuteilung von Aufgaben erledigen.

Die Zusammenarbeit könnte asynchron realisiert werden, beispielsweise unter Verwendung von Lively Wiki (Krahn u. a., 2009), dann müsste jedes Team-Mitglied, nachdem es Veränderungen vorgenommen hat, diese abspeichern. Dabei kann jedoch ein Konflikt auftreten, wenn bereits ein anderes Team-Mitglied Änderungen vorgenommen und abgespeichert hat. Bei der Erstellung der Anwendung müsste man sich überlegen, wie solche Konflikte vermieden werden können bzw. was bei Auftreten eines Konfliktes unternommen werden soll. Ein weiterer Nachteil bei asynchroner Zusammenarbeit ist, dass die Team-Mitglieder nicht zur gleichen Zeit die Zuteilung der Aufgaben bearbeiten können, sondern immer nur nacheinander.

Dadurch, dass WebCards synchrone Zusammenarbeit ermöglicht, können die Team-Mitglieder gleichzeitig die Aufgabenzuteilung bearbeiten. Des Weiteren wird so vermieden, dass es beim Speichern zu einem Konflikt kommt, da bereits ein anderer Benutzer, ausgehend von der gleichen Version, seine Änderungen abgespeichert hat. Auf technischer Ebene treten also keine Konflikte mehr auf. Natürlich kann es aber auf menschlicher Ebene dazu kommen, dass zwei Team-Mitglieder über die Zuteilung einer Aufgabe uneinig sind und diese deshalb fortwährend ändern. Solche Unstimmigkeiten werden bei synchroner Kollaboration schneller sichtbar als bei asynchroner.

Die im Rahmen dieser Arbeit vorgenommenen Erweiterungen von Lively Kernel um eine Undo-Funktionalität, die Möglichkeit synchron zu kollaborieren und auch die Vorlagen-Funktionalität basieren auf einem einheitlichen Grundkonzept: die durchgehende Verwendung von Kommando-Objekten. Die Implementierung einer Undo-Funktionalität auf Basis von Kommando-Objekten wird von Gamma u. a. (1995) nahe gelegt. Doch auch Persistenz, asynchrone Kollaboration, synchrone Kollaboration sowie eine Vorlagen-Funktionalität können unter Verwendung derselben Kommando-Objekte realisiert werden. Die sich daraus ergebenden Vorteile, aber auch die Beschränkungen werden im Rahmen dieser Arbeit beleuchtet. Zusätzlich wird die Frage beantwortet, wie eine bereits bestehende Autorenumgebung um die Verwendung von Kommando-Objekten erweitert werden kann. Dabei ist ebenfalls zu beachten, dass Änderungen, die durch Kommando-Objekte repräsentiert werden, sowohl über die UI als auch durch Skripte vorgenommen werden.

Der verbleibende Teil dieser Arbeit gliedert sich wie folgt: Kapitel 2 erläutert die Grundlagen von WebCards. Dazu wird als erstes Lively Kernel als Ausgangspunkt für WebCards näher untersucht. Anschließend wird ein Überblick über HyperCard gegeben, das als Ideengeber für WebCards fungiert. Dabei werden die in HyperCard verwendeten Konzepte und Methaphern vorgestellt und die Vor- und Nachteile von HyperCard analysiert sowie die Gründe für den Erfolg von HyperCard aufgezeigt.

Kapitel 3 beschreibt darauf aufbauend das Konzept von WebCards und geht dabei auf die Hintergrund-Metapher, die Möglichkeit synchron zu kollaborieren sowie die Funktionalität zum Rückgängig machen von Veränderungen an Morphs ein.

Im darauf folgenden Kapitel 4 wird die Implementierung dieser Funktionalitäten unter Verwendung von Kommando-Objekten erläutert. Dabei wird zunächst die Erweiterung von Lively Kernel um einen Mechanismus zur Erzeugung und Verwendung von Kommando-Objekten erklärt. Danach werden die Algorithmen, die die einzelnen Funktionalitäten realisieren, erläutert und bewertet. Im Anschluss wird die Verwendung von Kommando-Objekten für viele Funktionalitäten gleichzeitig diskutiert.

In Kapitel 5 wird anhand zweier mit WebCards erstellter Anwendungen das Konzept und die Implementierung von WebCards evaluiert.

Kapitel 6 gibt einen Überblick über verwandte Arbeiten, dabei wird jede Arbeit von WebCards abgegrenzt.

Abschließend gibt Kapitel 7 eine Zusammenfassung über diese Arbeit und macht Vorschläge für zukünftige Untersuchungen und Weiterentwicklungen.

Kapitel 2

Grundlagen

2.1 Lively Kernel

Lively Kernel[1] ist ein Framework zur Entwicklung von Webanwendungen im Stil von Desktopanwendungen. Dabei hebt sich Lively Kernel von Adobe Flash, Microsoft Silverlight, JavaFX u.a. ab, da zur Ausführung kein Plugin, sondern lediglich ein aktueller Webbrowser benötigt wird.

2.1.1 JavaScript Basierung

Ein erklärtes Ziel von Lively Kernel ist es, im Gegensatz zu herkömmlichen Webentwicklungs-Frameworks, nur wenige Technologien als Grundlage zu haben (Taivalsaari u. a., 2008, S. 7). Da die Entwickler von Lively Kernel des Weiteren JavaScript als die de facto Programmiersprache des Webs ansehen, ist Lively Kernel komplett in JavaScript geschrieben. JavaScript bildet somit eine der vier Hauptkomponenten[2] von Lively Kernel (Taivalsaari, 2008, S. 3).

Auch der Benutzer des Frameworks schreibt seine Anwendung in JavaScript. Dabei sind für ihn drei APIs von Bedeutung (Taivalsaari, 2008, S. 5):

1. **Core JavaScript API**[3]. Sie wird vom Browser zur Verfügung gestellt. Um mit möglichst allen Browsern kompatibel zu seien, sollten bevorzugt Merkmale verwendet werden, die in ECMAScript (ECMA International, 1999) definiert sind.

[1] `http://www.lively-kernel.org/` (Abgerufen am 24.08.2009)
[2] Die verbleibenden drei Hauptkomponenten sind: XMLHttpRequest, Morphic und Built-in IDE.
[3] Als Referenz kann `https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference` verwendet werden (Abgerufen am 7.10.2009).

2. **Prototype Library API**[4]. Sie wird vom Server, von dem Lively Kernel geladen wird, ausgeliefert. Prototype ergänzt die Core JavaScript API um einige nützliche Funktionen und liefert die Mechanismen zur Klassen-Definition und Erweiterung.
3. **Lively Kernel API**. Auf sie wird teilweise im Folgenden eingegangen.

2.1.2 UI-Framework Morphic

Ein Großteil der in Lively Kernel definierten Klassen gehört zum UI-Framework Morphic (Taivalsaari, 2008, S. 6). Morphic wurde ursprünglich für Self entwickelt (Maloney, 1995; Maloney u. Smith, 1995) und später in Squeak adaptiert (Ingalls u. a., 1997).

Die Hauptkonzepte in Morphic sind *Morphs*, *Welten* und *Hände*[5] (Taivalsaari, 2008, S. 6), wobei Morphs die Grundlage sind, da Welt und Hand Unterklassen von Morph sind. Der Begriff Morph kommt aus dem Griechischen und steht für Gestalt bzw. Form (Maloney, 1995, S. 2). Ein Morph ist ein aktives Objekt (Ingalls u. a., 2008, S. 35) mit einer visuellen Repräsentation. Die wichtigsten Merkmale eines Morphs werden im Folgenden erläutert:

- Ein Morph kann aus mehreren anderen Morphs zusammengesetzt werden. Die Morphs bilden dabei eine Baumstruktur. Dafür besitzt jeder Morph einen Array namens `submorphs`, der die Kind-Morphs enthält.
- Ein weiterer Bestandteil jedes Morphs ist eine Koordinatentransformation, die seine Form, sowie die Form all seiner Submorphs betrifft. Eine solche affine Transformation spiegelt beispielsweise die Rotation, die Skalierung und die Position eines Morphs wider.
- Des Weiteren definiert jeder Morph für sich, wie er mit Benutzer-Eingaben umgeht. Dies betrifft sowohl grundlegende Benutzer-Aktionen wie Mausklicks oder Tastatureingaben als auch spezielle Aktionen wie Änderung der Größe, der Rotation oder der Position.

Eine Welt ist das Lively Kernel Pendant zu einer klassischen Webseite. Eine Welt nimmt den ganzen Platz ein, den der Browser zur Verfügung stellt. Ein Morph ist nur dann sichtbar, wenn er der Submorph einer Welt ist. In einem laufenden Lively Kernel kann es mehrere Welten geben. Diese können dann durch `LinkMorphs` aufeinander verweisen.

Eine Hand ist die Repräsentation des Mauszeigers. Lively Kernel ist so ausgelegt, dass später Unterstützung für mehrere Mauszeiger nachgerüstet werden kann, um so Kollaboration zu ermöglichen.

[4] Als Referenz dient `http://www.prototypejs.org/api` (Abgerufen am 7.10.2009).
[5] Im englischen Original ist von *Morphs*, *Worlds* und *Hands* die Rede.

Direktheit und Lebendigkeit

Laut Maloney u. Smith (1995) bedeutet Direktheit, dass der Prozess, eine Komponente der Benutzerschnittstelle zu verändern, dadurch begonnen wird, dass auf die betreffende Komponente gezeigt wird. Lebendigkeit meint, dass die Benutzerschnittstelle immer aktiv und reaktiv ist.

Dadurch, dass Morphic in hohem Maße direkt und lebendig ist, macht es Spaß Morphic zu benutzen (Maloney u. Smith, 1995). Des Weiteren ist das Erzeugen und Editieren von interaktiven graphischen Objekten einfach möglich (Taivalsaari u. a., 2008, S. 9). Dies kann dabei nicht nur programmatisch, sondern auch direkt vom Benutzer des laufenden Programms vorgenommen werden. Der Benutzer muss dazu nicht in einen speziellen Bearbeitungsmodus wechseln. Er kann aus dem laufenden Programm heraus per Kontextmenü für jeden beliebigen Morph einen *Style Panel* öffnen, der das Ändern der Farbe, der Kantendicke, der Kantenfarbe, der Deckkraft usw. erlaubt. Auch kann der Benutzer per Drag-and-Drop die Position des Morphs ändern und sogar den Morph von seinem Eltern-Morph trennen und mit einem anderen Morph kombinieren. Genauso einfach kann die Rotation sowie die Skalierung eines Morphs vom Benutzer geändert werden. Auch das Duplizieren eines Morphs mit all seinen Eigenschaften ist leicht möglich. Der Entwickler eines speziellen Morphs hat zwar die Möglichkeit all diese Freiheiten des Benutzers einzuschränken, oft würde dies jedoch die Direktheit und Lebendigkeit von Morphic unnötigerweise einschränken.

In Morphic wird bewusst auf eine Unterscheidung zwischen Ausführen und Bearbeiten der UI verzichtet. Dadurch wird das Erstellen der UI beschleunigt. Des Weiteren muss sich der Entwickler nicht merken, in welchem Modus er sich gerade befindet. In einem System, das gleichzeitig von mehreren Anwendern genutzt wird, hat der Verzicht auf getrennte Ausführungen- und Bearbeiten-Modi noch einen weiteren Vorteil. Durch den Verzicht wird vermieden, dass Benutzer sich in unterschiedlichen Modi befinden, was einerseits zu Verwirrungen auf Seite der Nutzer führen könnte und andererseits beim Entwickeln des synchron genutzten Systems beachtet werden müsste.

2.1.3 Selbsttragend

Die Möglichkeiten für Veränderung am laufenden Programm, die ein Benutzer von Lively Kernel hat, gehen weit über die eben aufgezählten simplen Manipulationen an Morphs hinaus. So kann beispielsweise für jedes Objekt ein *Inspector* geöffnet werden, der alle JavaScript-Eigenschaften[6] des Objektes anzeigt und ihre Veränderung erlaubt. Aber nicht nur das Verändern von Eigenschaften einzelner Objekte ist möglich. Der *Code Browser* erlaubt auch das Betrachten und Editieren von Klassen und

[6] Jedes Objekt in JavaScript hat Eigenschaften. Auf sie kann per Punktnotation oder Eckige-Klammer-Notation zugegriffen werden. Auch die Methoden eines Objektes sind als normale Eigenschaften im Objekt abgelegt. Zu jeder Zeit können einem Objekt neue Eigenschaften gegeben und bestehende Eigenschaften gelöscht werden.

deren Methoden. Mit Hilfe des Code Browsers, des System Browsers und anderen direkt in Lively Kernel implementierten Tools ist es möglich, Lively Kernel aus sich heraus weiterzuentwickeln; Lively Kernel ist somit selbsttragend (Ingalls u. a., 2008).

2.1.4 SVG als eine Implementierungsgrundlage

Die meisten Webanwendungen basieren auf HTML. Lively Kernel hingegen verwendet zur Darstellung der Morphs auf Implementierungssebene Scalable Vector Graphics[7] (SVG). Dies bietet sich an, da viele der in Morphic geforderten Eigenschaften wie beispielsweise affine Transformationen, die für das Drehen und Skalieren benötigt werden, in SVG direkt zur Verfügung stehen.

Doch die Verwendung von SVG bringt auch Nachteile mit sich, da die Verbreitung von SVG noch immer nicht sehr groß ist. Dies führt dazu, dass die verfügbaren SVG-Engines erstaunlich langsam sind (Taivalsaari u. a., 2008, S. 16). Des Weiteren unterstützt kein Browser SVG komplett (Molina u. a., 2007), insbesondere der Windows Internet Explorer unterstützt SVG nativ gar nicht.

Jeder Morph in Lively Kernel hat als Bestandteil ein Objekt namens `shape`. Dieses Shape-Objekt kapselt die Implementierung der graphischen Repräsentation des Morphs nach dem Bridge-Pattern (Maier u. Dwornitzak, 2007, S. 9). Dies hat den Vorteil, dass die Implementierung später durch eine andere ausgetauscht werden kann. So existiert bereits eine experimentelle Canvas-Version[8] von Lively Kernel, in der anstelle von SVG das Canvas-Element aus dem Entwurf zur HTML 5 Spezifikation[9] verwendet wird.

2.1.5 Schlussfolgerung

Lively Kernel ermöglicht es, aktive graphische Objekte einfach zu erzeugen und zu verändern. Aufgrund der Direktheit und Lebendigkeit von Morphic macht es Spaß, Lively Kernel zu benutzen. Benutzer haben die volle Kontrolle über das System und können alle Morphs direkt inspizieren sowie verändern und werden dabei nicht durch Modi eingeengt. Lively Kernel lädt Benutzer so zum Experimentieren ein.

Beim Entdecken der Möglichkeiten von Lively Kernel ist der Benutzer jedoch nicht durch einen Undo-Mechanismus abgesichert. Einzig der Textmorph bietet in Bezug auf seinen textuellen Inhalt die Möglichkeit, Änderungen zurückzunehmen.

Bis jetzt bietet Lively Kernel keine Unterstützung für synchrone Kollaboration. Lively Wiki (Krahn u. a., 2009), ein auf Lively Kernel basierendes Wiki, ermöglicht asynchrone

[7] `http://www.w3.org/TR/2003/REC-SVG11-20030114/` (Abgerufen am 24.08.2009)
[8] `http://livelykernel.sunlabs.com/repository/lively-kernel/trunk/source/kernel/expt.xhtml` (Abgerufen am 12.10.2009)
[9] `http://www.w3.org/TR/html5/the-canvas-element.html` (Abgerufen am 12.10.2009)

Kollaboration. Dabei kann es jedoch zu Konflikten kommen, wenn zwei Benutzer die gleiche Version eines Wiki-Eintrags verändern und anschließend speichern wollen.

Mit Lively Kernel kann das Aussehen von Morphs so einfach und direkt verändert werden wie Grafiken in einem Zeichenprogramm. Das Erstellen kompletter Anwendungen ist hingegen erheblich komplexer und erfordert Einarbeitung in JavaScript, Morphic und Lively Kernel.

2.2 HyperCard

Aufgrund seiner Vielschichtigkeit fällt es schwer, HyperCard mit nur wenigen Worten zu erklären. Hier ein Versuch aus der c't 4/90:

> „HyperCard ist ein erstaunlich mächtiges Produkt, das seit langem jedem verkauften Macintosh beiliegt – sozusagen als BASIC-Ersatz für den Renommier-Rechner der Neunziger: eine gut gerührte Mischung aus Malprogramm, Datenbank, objektorientierter Hochsprache [...]. So gibt es wirklich kaum eine Anwendung, die man mit diesem Werkzeug nicht angehen könnte."

Im Folgenden wird HyperCard, das als Ideengeber für WebCards dient, untersucht. Dabei wird beleuchtet, was die Vorteile von HyperCard sind. Dazu werden die Metaphern von HyperCard betrachtet und es wird aufgezeigt welche Arten von Anwendungen mit welchem Aufwand unter Anwendung welcher Kenntnissen und Fähigkeiten mit Hilfe von HyperCard erstellt werden können. Zusätzlich zu den Vorteilen werden auch die, zumindest aus heutiger Sicht, vorhandenen Schwächen von HyperCard aufgezeigt.

2.2.1 Zeitliche Einordnung

Bill Atkinson entwickelte HyperCard für Apple Computer Incorporated[10]. Die erste Version erschien im August 1987 (Goodman, 1998a, S. xix). HyperCard wurde kostenlos mit jedem Macintosh-Rechner ausgeliefert und stand somit einer großen Benutzergruppe zur Verfügung (Coulouris u. Thimbleby, 1993, S. x).

Im Oktober 1990 erschien die stark erweiterte Version 2.0 (Günther, 1990). Etwas später wurden die Rechte sowie die Entwicklung an das Tochterunternehmen Claris ausgegliedert. Um mit HyperCard Geld zu verdienen, wurde es nicht mehr jedem Macintosh-Rechner beigelegt, sondern musste zusätzlich gekauft werden. Den Rechnern lag lediglich ein sogenannter HyperCard-Player bei, der das Abspielen, nicht jedoch das Bearbeiten oder Erzeugen von HyperCard-Programmen erlaubte.
Ende 1993 wurde HyperCard wieder zu Apple zurück transferiert und Version 2.2 erschien (Goodman, 1998a, S. xix).

[10] heute: Apple Inc.

Becker u. Dörfler (1991, S. 267) schrieben zur Zukunft von HyperCard:

> „Es ist klar, daß es weitergehen wird. Wir denken, daß HyperCard eines der strategischen Produkte von Apple sein wird — vergleichbar in seiner Bedeutung mit der Systemsoftware der Rechner der Macintosh-Familie."

Entgegen dieser Einschätzung erschien 1998 die letzte Version[11] von HyperCard.

2.2.2 Grundlegende Konzepte

HyperCard stellt fünf verschiedene Objekt-Klassen zur Verfügung, die das Bauen kompletter Anwendungen erlauben: Stapel, Karten, Hintergründe, Textfelder und Knöpfe[12] (Coulouris u. Thimbleby, 1993, S. 49).

Ein HyperCard-Programm wird als Stapel bezeichnet[13]. Ein Stapel stellt eine geordnete Sammlung von Karten dar. Auf einer Karte können sich Textfelder und Knöpfe befinden. Zu jedem Zeitpunkt ist immer nur eine Karte eines Stapels sichtbar.

Bei vielen HyperCard-Anwendungen kommt es vor, dass es viele ähnliche Karten geben soll, die sich nur in wenigen Punkten, wie dem im Inhalt der Textfelder, unterscheiden. Für diesen Zweck gibt es Hintergründe. Jede Karte hat genau einen Hintergrund, wobei mehrere Karten denselben Hintergrund haben können. Elemente, die sich auf einem Hintergrund befinden, stehen dann auf allen Karten mit diesem Hintergrund zur Verfügung.

Knöpfe

Vor HyperCard wurden Mac-Programme strikt nur über die Menüleiste und mit Dialogboxen gesteuert (Becker u. Dörfler, 1991, S. 66). In typischen HyperCard-Stapeln ist dies jedoch anders. Hier steckt die meiste Funktionalität in Knöpfen (Becker u. Dörfler, 1991, S. 160). Für jeden Knopf kann ein Skript festgelegt werden, das beim Betätigen des Knopfes ausgeführt wird. Als Programmiersprache dient dafür HyperTalk (siehe 2.2.5).

Zusätzlich zum HyperTalk-Skript kann detailliert das Aussehen des Knopfes festlegt und ein Name angeben werden. Der Name wird auf dem Knopf angezeigt und kann verwendet werden um in HyperTalk-Skripten auf den Knopf Bezug zu nehmen.

Knöpfe können innerhalb und auch zwischen verschiedenen Stapel hin und her kopiert werden. Dabei wird sowohl das Aussehen als auch die Funktionalität mit kopiert.

[11] `http://docs.info.apple.com/article.html?artnum=24522` (Abgerufen am 2.11.2009)

[12] Die englischen original Bezeichnungen lauten: Stacks, Cards, Backgrounds, Fields und Buttons. In Becker u. Dörfler (1991) werden Knöpfe auch als Tasten bezeichnet.

[13] In Becker u. Dörfler (1991) wird dafür auch die englische Originalbezeichnung „Stack" verwendet. Gelegentlich wird auch von „Stackware" gesprochen.

Textfelder

Textfelder in HyperCard ähneln den aus HTML bekannten Text-Input Feldern eines Formulars. Benutzer eines HyperCard-Stapels können in Textfelder Text eingeben, der dann automatisch gespeichert ist. Aus HyperTalk kann auf Textfelder zugegriffen werden, um beispielsweise den Inhalt als Sortierkriterium zu nutzen oder aber um den Inhalt zu ändern.

Textfelder können mehrere Zeilen haben. Die Zeilen sind dann, wie bei Linienpapier, durch dünne horizontale Striche voneinander getrennt. In HyperTalk gibt man in einem solchen Fall an, auf welche Zeile des Textfeldes man zugreifen möchte (siehe Auflistung 2.2).

Wie auch bei Knöpfen kann für Textfelder das Aussehen und ein Name sowohl über einen Eigenschaften-Dialog also auch programmatisch festgelegt werden. Zusätzlich ist es möglich Schriftgröße und Stil zu ändern.

Hintergründe

Hintergründe sind ein ebenso mächtiges wie komplexes Konzept in HyperCard, welches im Folgenden erläutert wird.

Oft wird HyperCard als eine Art Datenbank verwendet. Ein beliebtes Beispiel hierfür ist ein persönliches Adressbuch. In diesem Fall enthält der Stapel viele Karten, die sich nur im Inhalt der Textfelder unterscheiden. Dies wird erreicht, indem alle Textfelder und Knöpfe auf dem Hintergrund einer Karte platziert werden. Dann können einfach neue Karten erzeugt werden, die denselben Hintergrund haben und somit auch dieselben Textfelder und Knöpfe.

Hintergründen sind mit Karten eng verbunden und existieren immer nur in Verbindung mit einer Karte. Jede Karte hat genau einen Hintergrund, wobei es aber in einem Stapel Karten mit unterschiedlichen Hintergründen geben kann. Um einen neuen Hintergrund zu erzeugen, muss eine neue Karte mit einem leeren Hintergrund erzeugt werden. Wenn die letzte Karte, die einen bestimmten Hintergrund hat, gelöscht wird, verschwindet auch der entsprechende Hintergrund und so geschieht es, dass Hintergründe oft aus Versehen gelöscht werden (Becker u. Dörfler, 1991, S. 173).

Um einen Hintergrund zu bearbeiten, muss man auf eine Karte, die diesen Hintergrund hat, wechseln und in den Hintergrund-Bearbeiten Modus schalten. Welcher Modus gerade aktiv ist, lässt sich lediglich am Aussehen der Menüleiste erkennen. Deshalb passiert es gerade Anfängern, dass sie vergessen, in den richtigen Modus umzuschalten (Becker u. Dörfler, 1991, S. 35).

In relationalen Datenbanken hat jede Tabelle ein Relationenschema, in dem die Anzahl, die Namen und die Typen der Tabelleneinträge definiert sind. Dies kann als Metapher

für das Verhältnis von Hintergrund (Relationenschema) zu Karte mit diesem Hintergrund (Tabelleneintrag) dienen[14]. Ebenso wie in einer Datenbank alle Einträge einer bestimmten Tabelle abgerufen werden können, kann in HyperTalk über alle Karten mit einem bestimmten Hintergrund iteriert werden.

Als weitere, ähnliche Metapher kann das Verhältnis zwischen Klasse und Instanz dienen, das oft in der objektorientierten Programmierung zu finden ist. Im Gegensatz zu Klassen, die Vererbungs-Hierarchien bilden können, haben Hintergründe keinerlei Beziehung zueinander. Ein Hintergrund hat keinen weiteren Hintergrund. Nur eine Karte kann einen Hintergrund haben.

Doch wie ist das Verhältnis von Karte zu Hintergrund im Detail? Was kann, um in dem Gedankenbild zu bleiben, instanzspezifisch sein? Auf einer Karte mit einem bestimmten Hintergrund befinden sich alle Knöpfe und Textfelder dieses Hintergrunds. Dabei haben die Knöpfe und Textfelder, mit zwei Ausnahmen, auf allen Karten mit dem Hintergrund dieselben Eigenschaften (Coulouris u. Thimbleby, 1993, S. 55f). Hintergrund-Knöpfe teilen sich nicht die Eigenschaft `hilite`, die festlegt, ob ein Knopf hervorgehoben dargestellt wird. Die zweite und entscheidende Ausnahme besteht bei Textfeldern. Der Text eines Textfeldes wird je Karte gespeichert, ist also „instanzspezifisch“.

Weitere instanzspezifische Eigenschaften für Hintergrund-Knöpfe und Hintergrund-Textfelder sind nicht vorgesehen. So ist es also nicht möglich, einem Hintergrund-Textfeld auf einer bestimmten Karte eine abweichende Position zuzuordnen. Das Verschieben oder anderweitige Editieren von Hintergrund-Textfeldern ist aber ohnehin nur im Hintergrund-Bearbeiten Modus möglich.

Unabhängig vom Hintergrund können auf jeder Karte beliebige Elemente platziert werden. So kann beispielsweise in einem Adressen-Stapel auf der Karte, die die Adresse von Petra Mustermann enthält, eine Textfeld hinzugefügt werden, in dem die Lieblings-Schokolade von Petra vermerkt wird. Dieses Textfeld taucht dann einzig und allein auf dieser speziellen Karte auf. Solche Objekte werden gelegentlich als Vordergrund bezeichnet.

Mit diesem Mechanismus kann auch „instanzspezifisches“ Verhalten simuliert werden (Goodman, 1998a, S. 124). Beispielsweise wird in einem Adressen-Stapel ein Hintergrund-Knopf zum Löschen der aktuellen Karte definiert. Da uns Petra Mustermann aber sehr am Herzen liegt, soll das leichtfertige Löschen ihrer Karte verhindert werden. Zu diesem Zweck wird ein Knopf ohne Funktionalität über dem Hintergrund-Knopf platziert. So ist der Hintergrund-Knopf nicht mehr anklickbar. Der Karten-spezifische Knopf im Vordergrund überlagert den Hintergrund-Knopf.

[14] Tatsächlich ist HyperCard keine relationale Datenbanken (siehe 2.2.4).

Graphik

Zusätzlich zu den fünf vorgestellten Bausteinen einer HyperCard-Anwendung gibt es noch ein weiteres Konzept, das jedoch nicht als Objekt bezeichnet werden kann. HyperCard erlaubt das direkte Malen auf Karten in der Art, die aus MacPaint und ähnlichen Programmen bekannt ist. Dies liegt sicher nicht zuletzt daran, dass MacPaint ebenso wie HyperCard von Bill Atkinson entwickelt wurde.

Der Benutzer kann zwischen 15 Werkzeugen, darunter Bleistift, Pinsel, Radiergummi, Sprühdose, Lasso, Rechteck und Oval sowie 32 Pinselformen und 40 Mustern wählen. Die Muster dienen als Ersatz für Farben, da die damaligen Bildschirme nur schwarz und weiß anzeigen konnten.

Wie auch in MacPaint ist das Zeichnen in HyperCard pixelorientiert. Je nachdem, ob sich der Benutzer im Hintergrundmodus befindet oder nicht, wird das Gemalte Teil des Hintergrundbildes oder des Kartenbildes. Zeichnungen im Kartenbild überlagern Zeichnungen im Hintergrundbild.

Oft werden die statischen Texte einer Karte mit Hilfe des Werkzeuges *Text* in das Hintergrundbild gemalt. Der Text liegt dann jedoch nicht mehr als Objekt vor und kann deshalb nicht ohne Weiteres nachträglich editiert werden. Auch ist die Verwendung des Textes in HyperTalk nicht möglich.

Simple Animationen, wie das Rotieren eines Sternes, sind aufgrund der Objektorientierung in Morphic einfach möglich. In HyperCard ist dies nur kompliziert und über Umwege möglich. Um einen solchen Effekt in HyperCard zu realisieren, macht man sich zunutze, dass Malen nicht nur mit Hilfe der Maus durch den Anwender erfolgen kann, sondern auch in einer HyperTalk-Methode. In einer HyperTalk-Methode kann mit Turtle-Grafik gezeichnet werden (Coulouris u. Thimbleby, 1993, S. 95ff). Um einen so gezeichneten Stern zu rotieren, entfernt man ihn auf die gleiche Art unter Verwendung des Radiergummis o.ä., muss dabei jedoch darauf achten, nicht versehentlich etwas anderes ebenfalls zu löschen, und lässt dann einen rotierten Stern zeichnen.

2.2.3 Stufen der Verwendung

Nachdem nun die Bausteine einer HyperCard-Anwendung bekannt sind, wird im Folgenden aufgezeigt, wie HyperCard verwendet werden kann und welche Kenntnisse Benutzer dafür benötigen. Dazu wird eine Aufteilung in vier Stufen vorgenommen[15]. Dabei wird deutlich, dass HyperCard für Benutzer jeder Stufe interessante und nützliche Möglichkeiten bietet. Insbesondere Anfänger können schnell erste eigene Anwendungen erstellen.

[15] Eine ähnliche Aufteilung findet sich auch bei Goodman (1998a, S. xxxviii), der die Fähigkeiten von HyperCard-Benutzern in drei Level aufteilt.

Stufe eins

HyperCard wurde immer in Verbindung mit einigen fertigen Stapeln, wie z.B. dem Adressbuch-Stapel, ausgeliefert. So können Anwender erste Erfahrungen in der Verwendung von HyperCard sammeln, ohne überhaupt wissen zu müssen, welche Möglichkeiten ihnen HyperCard bietet.

Stufe zwei

Anwender, die um die Möglichkeiten von HyperCard wissen, können erste eigene Stapel erstellen, ganz ohne dabei programmieren zu müssen (Becker u. Dörfler, 1991, S. 37). So bietet es sich beispielsweise an HyperCard zum Erstellen von Präsentationen zu verwenden (Becker u. Dörfler, 1991, S. 205). Dabei kann, muss jedoch nicht, in HyperTalk programmiert werden. Um Effekte für Karten- bzw. Folienübergänge einzustellen, bietet HyperCard einen speziellen Dialog an, der den dafür benötigten HyperTalk-Code automatisch generiert (Becker u. Dörfler, 1991, S. 42).

Das Erstellen von Stapeln ganz ohne Programmieren, wird dadurch unterstützt, dass es möglich ist, Knöpfe inklusive ihrer Funktionalität zu kopieren. HyperCard liefert extra einen Stapel mit, der Kopiervorlagen für häufig verwendete Knöpfe enthält (Becker u. Dörfler, 1991, S. 32).

Stufe drei

Die nächste Stufe ist die Verwendung von HyperTalk (siehe 2.2.5). Hat ein Anwender einen eigenen Stapel erstellt, kann es vorkommen, dass er Anforderungen hat, die nicht ohne Programmierung gelöst werden können. Dies kann beispielsweise die Erstellung einer Übersicht sein auf der alle Karten, die einen bestimmten Hintergrund haben, zusammengefasst werden. Diese ist keine ausgefallene Anforderung. Deshalb kann der Anwender sich in anderen Stapeln anschauen, wie dort diese Anforderung gelöst wurde. Er kann auch den HyperTalk-Quelltext in seinen Stapel kopieren und entsprechend anpassen. Dies ist eine gute Art einen Einstieg in die HyperTalk-Porgrammierung zu finden (Becker u. Dörfler, 1991, S. 184).

Stufe vier

Die fortgeschrittenste Verwendung von HyperCard ist die Erstellung von externen Funktionen und Kommandos (XCMD). Diese werden außerhalb von HyperCard und in einer beliebigen Sprache, beispielsweise C oder Pascal, geschrieben und als Maschienencode in Stapel eingebunden. Anschließend stehen sie als normale HyperTalk Funktionen bzw. Kommandos zur Verfügung.

Ihre Verwendung bietet sich an, um Aufgaben zu erledigen, die außerhalb der Möglichkeiten von HyperCard sind, wie z.B. das Ansprechen bestimmter Hardware. Des Weiteren können XCMDs verwendet werden, um bestimmte Berechnungen schneller auszuführen als es in HyperTalk möglich wäre. Ebenso können XCMDs verwendet werden, um den Quellcode zu verschleiern.

Die Erstellung von XCMDs ist jedoch komplex und riskant (Coulouris u. Thimbleby, 1993, S. 47) und deshalb für Endanwender nicht empfehlenswert.

2.2.4 Sichtweisen auf HyperCard

Nachdem zuvor die Bausteine einer HyperCard-Anwendung sowie die unterschiedlichen Stufen der Verwendung von HyperCard aufgezeigt wurden, versucht dieser Abschnitt die Frage zu beantworten, was HyperCard *ist*. Dazu werden unterschiedliche Sichtweisen auf HyperCard aufgezeigt. In Becker u. Dörfler (1991, S. 260ff) werden verschiedene Verwendungen von HyperCard aufgelistet. Die Sätze beginnen dabei immer mit „HyperCard ist ein [...]“. Dies dient als Anregung für den folgenden Abschnitt.

HyperCard ist ein Zeichenprogramm

Ohne sich mit der restlichen Funktionsweise von HyperCard auseinandersetzen zu müssen, können Anwender die Zeichenfunktionalität (siehe 2.2.2) von HyperCard nutzen. Die Benutzerschnittstelle zum Malen entspricht dabei weitestgehend dem, was Anwender bereits aus anderen Zeichenprogrammen kennen. Auch das Verwenden von Bilddateien aus anderen Programmen ist problemlos möglich.

Durch die Zeichenfunktionalität kann der Anwender direkt das Design eines Stapels nach seinen Vorstellungen anpassen. Rahmen und Ähnliches können einfach gezeichnet werden.

Zeichnungen in HyperCard sind pixelorientiert und keine Objekte. Dies hat, wie oben erläutert, Nachteile, insbesondere bei der nachträglichen Manipulation. Gleichzeitig kann die Pixelorientierung jedoch auch als vorteilhaft angesehen werden, da Endanwender dieses Konzept bereits kennen und somit kein neues erlernen müssen.

HyperCard ist eine Datenbank

In einem Stapel können beliebige Daten persistent gespeichert und später wieder abgerufen werden. Das Erstellen eines Hintergrunds kann mit der Definition einer Tabelle verglichen werden. Dann entspricht eine Karte, die diesen Hintergrund hat, einem Eintrag in der Tabelle. Wie oben bereits erwähnt, ist es möglich via HyperTalk über alle Karten eines Hintergrunds zu iterieren.

Aktuelle, erfolgreiche, relationale Datenbankmanagementsystem (DBMS) unterstützen Anfragen mit Hilfe der deklarativen Sprache SQL. Damit können alle Einträge, die einem bestimmten Kriterium entsprechen, einfach abgerufen werden. Auch Anfragen, die zwei oder mehr Tabellen miteinander in Verbindung setzen, sind ohne Weiteres möglich.

HyperCard ist jedoch keine relationale Datenbank (Goodman, 1998a, S. xxxiiiff und Seite 84). Der Fokus von HyperCard liegt hingegen auf dem Durchstöbern und Betrachten einzelner Karten. Zum Wiederfinden bestimmter Informationen hat HyperCard eine Suchfunktion eingebaut. Diese Suchfunktion orientiert sich an Suchfunktionen, wie sie aus Textverarbeitungsprogrammen bekannt sind. Der Anwender gibt in eine Dialogbox den Suchbegriff ein und bestätigt. Daraufhin wird auf die nächste Karte mit einem Textfeld, das den Suchbegriff enthält, gewechselt und der gefundene Suchbegriff markiert. Erneutes Bestätigen in der Dialogbox wiederholt diesen Vorgang, sodass nacheinander alle Funde durchgegangen werden können.

Im Gegensatz zu Anfragen per SQL ist es dabei nicht möglich, die Suche auf ein bestimmtes Attribut, beispielsweise den Namen, zu beschränken. Wird also in einem Adressbuch der Eintrag zu Dieter Deutschland gesucht und als Suchbegriff „Deutschland“ eingegeben, müssen nacheinander alle Karten durchgegangen werden, in denen das Wort „Deutschland“ vorkommt, also auch die Karten in denen das Wort nicht als Nachname sondern in der Adresse vorkommt.

Aufgrund der heutzutage bestehenden Erfahrung mit datenzentrierten Webseiten ist eine der Komponenten, die Endanwender häufig fordern, eine Übersicht-Detail Verbindung (Rode u. a., 2006, S. 9). Ein Beispiel für eine Übersicht-Detail Verbindung ist eine Seite auf der alle Mitarbeiter einer Abteilung tabellarisch angezeigt werden. Ein Klick auf einen Eintrag führt zu einer Detailansicht des betreffenden Mitarbeiters. Eine solche Darstellung wird auf Webseiten oft auch zum Anzeigen von Suchergebnissen verwendet.

HyperCard bietet weder eine reine Übersichts- noch eine Übersicht-Detail-Komponente an. Es gibt dafür weder ein spezielles Objekt noch einen Assistenten, der eine entsprechende Karte nach Anwenderangaben erzeugt. Stattdessen muss der Anwender ein imperatives HyperTalk-Skript schreiben, das eine Übersichtskarte erzeugt. Oft kann er dafür jedoch bestehende Skripte aus anderen Stapeln kopieren und anpassen.

Anstelle einer Übersichts-Komponente bietet HyperCard aber zwei Möglichkeiten eine Übersicht mehrerer Karten auszudrucken. Die erste Möglichkeit ist eine leicht verständliche Druckfunktion, die es ermöglicht ausgewählten Karten einfach auszudrucken. Dabei können problemlos mehrere Karten auf ein Blatt gedruckt werden.

Zusätzlich zu dieser einfachen Methode gibt es den Dialog *Bericht drucken*. Ein Bericht fasst mehrere Karten mit gleichem Hintergrund zusammen. Der Benutzer kann dabei auswählen, welche Hintergrund-Textfelder gedruckt werden. Ebenso bestimmt er je Hintergrund-Textfeld die Schriftart sowie die relative Position. So können Berichte im

Stile von Etiketten oder Tabellen erstellt werden. Das Ergebnis kann der Benutzer bereits vor dem Drucken in einer Vorschau kontrollieren.

HyperCards Druckfunktion war sehr beliebt und wurde zum Drucken von Adressetiketten (Becker u. Dörfler, 1991, S. 253), Diskettenaufklebern (Coulouris u. Thimbleby, 1993, S. 20ff), Vortragsfolien und vielem mehr verwendet.

HyperCard ist eine Softwareentwicklungsumgebung

> „HyperCard is Apple Computer's application-building tool for the Macintosh“ (Coulouris u. Thimbleby, 1993, S. V)

HyperCard wird zum Entwickeln von Anwendungen verwendet. Je nach Verwendungsstufe (siehe 2.2.3) wird dabei unterschiedlich viel programmiert. Das Hauptfenster von HyperCard, das die aktive Karte anzeigt, und die Werkzeugpaletten zum Zeichnen sowie zum Bearbeiten von Textfeldern und Knöpfen können als GUI-Editor angesehen werden.

Des Weiteren enthält HyperCard zum Bearbeiten des HyperTalk-Codes einen Skript-Editor. Ab Version 2.0 enthält HyperCard zusätzlich einen Debugger (Günther, 1990).

Sowohl der Skript-Editor als auch der Debugger sind als XCMD und nicht in HyperTalk realisiert. Im Gegensatz zu Lively Kernel (siehe 2.1.3) ist HyperCard also nicht selbsttragend.

HyperCard ist ein Autorensystem

HyperCard erlaubt Lehrkräften und anderen Experten das Erstellen von Lernsoftware und ähnlichen Programmen. Die Erstellung der Inhalte, das heißt die Eingabe von Texten, das Einfügen von Bilddateien oder das Zeichnen von Grafiken, erfolgt nach dem Prinzip „What You See Is What You Get“.

Es ist sogar möglich Lernsoftware ganz ohne Verwendung einer Programmiersprache zu erstellen (siehe 2.2.3). Für anspruchsvollere Programme ist die Verwendung von HyperTalk jedoch nötig.

HyperCard ist ein Multimediasystem

HyperCard kann nicht nur mit Bildern und Texten umgehen. Auch Klänge lassen sich direkt mit HyperCard abspielen. Ebenso kann durch HyperTalk-Skripte „bewegte Grafik“ (Becker u. Dörfler, 1991, S. 5) erzeugt werden. Zusätzlich kann via XCMD externe Hardware angesprochen werden und so beispielsweise ein CD-Player aus HyperCard heraus gesteuert werden. Dies nutzte auch die erste kommerzielle[16] Multimedia CD-

[16] Amy Virshup, The Teachings of Bob Stein, 1996, `http://www.wired.com/wired/archive/4.07/stein_pr.html` (Abgerufen am 13.10.2009).

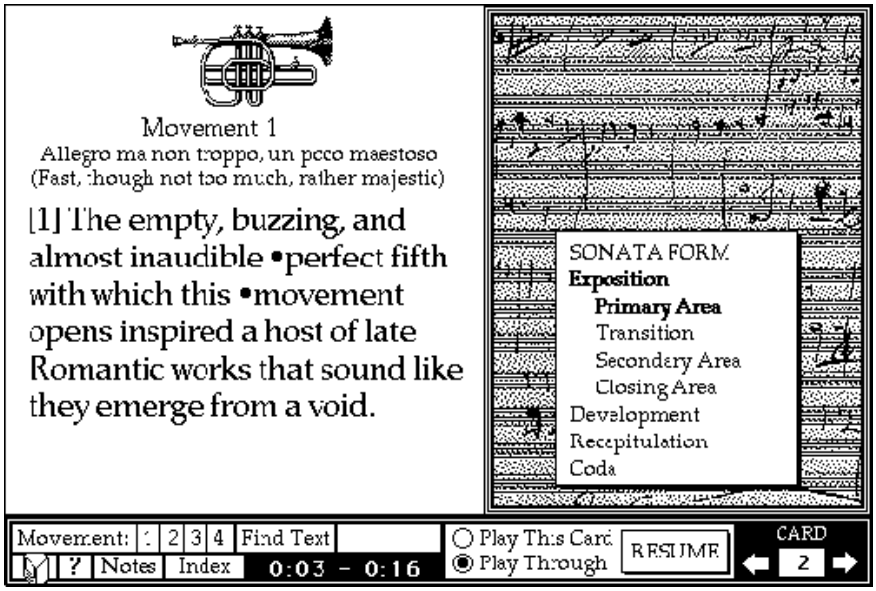


Abbildung 2.1: Screenshot des HyperCard-Stapel *CD Companion to Beethoven's Symphony No. 9* (Quelle: `http://www.futureofthebook.org/blog/archives/Hyper_beethoven.gif` (Abgerufen am 13.10.2009))

ROM *CD Companion to Beethoven's Symphony No. 9*[17]. Sie enthielt einen HyperCard-Stapel, der eine mitgelieferte Audio-CD steuerte und dem Benutzer Informationen zu dem Gehörten präsentierte[18] (siehe Abbildung 2.1).

Mit Hilfe von XCMD kann auch, was für die damaligen Verhältnisse fortschrittlich war, ein „Tonfilm" (Becker u. Dörfler, 1991, S. 272) von einem „Bildplattenspieler" (Becker u. Dörfler, 1991, S. 11) in eine Karte integriert werden.

Jedoch unterstützt HyperCard keine Farben. Bilder sind nur in schwarz/weiß und mit 72 dpi möglich (Becker u. Dörfler, 1991, S. 12).

2.2.5 HyperTalk

Ein Ziel von HyperCard ist es, dem Endanwender die volle Kontrolle über die Möglichkeiten seines Rechners zu geben (Wheeler, 2004, S. 3). Um dies zu erreichen, entwickelte Bill Atkinson für HyperCard eine neue Programmiersprache namens HyperTalk, die als Zielgruppe insbesondere Programmierneulinge hat.

[17] Robert Winter, Bob Stein, CD Companion to Beethoven's Symphony No. 9, The Voyager Company, 1989.

[18] Institute for the Future of the Book, The Beethoven's Ninth Symphony CD Companion, 2005, `http://web.archive.org/web/20071212204932/http://www.futureofthebook.org/next/text/2005/10/the_beethovens_ninth_symphony.html` (Abgerufen am 13.10.2009)

Umgangssprachlichkeit

Um dieser Zielgruppe gerecht zu werden, ist HyperTalk an der englischen Umgangssprache orientiert. Dies wird oft als Grund dafür angegeben, dass Programmieren in HyperTalk einfach sei.

> „HyperTalk's simple English-like expressions [...] make programs [...] simple to write." (Stanley, 1992, S. V)

> „[HyperTalk] ist einfach zu erlernen, weil es große Ähnlichkeiten mit der (englischen) Umgangssprache hat." (Becker u. Dörfler, 1991, S. 224)

Es wäre jedoch ein Trugschluss anzunehmen, dass beliebige umgangssprachliche, englische Anweisungen ein lauffähiges HyperTalk-Skript darstellen. HyperTalk folgt einer wohldefinierten Grammatik[19].

Der Vorteil der Umgangssprachlichkeit liegt in der guten Lesbarkeit von HyperTalk-Skripten. Als gute Beispiele dienen Auflistung 2.1 und 2.2.

```
on mouseUp
        go to next card
end mouseUp
```

Auflistung 2.1: Aus Becker u. Dörfler (1991, S. 37)

```
put the first word of the third line of field
'hello' into field 'goodbye'
```

Auflistung 2.2: Aus Wheeler (2004, S. 3)

Beide Code-Schnipsel bestehen hauptsächlich aus englischen Worten und können ohne große Kenntnisse von HyperTalk nachvollzogen werden.

Da Anwender sich einfach den Code anderer Stapel anschauen können, gibt es eine große Anzahl von leicht verfügbaren Code-Beispielen. Die hohe Verfügbarkeit von Beispielen, die wegen der guten Lesbarkeit leicht nachvollziehbar sind und die gegebenenfalls kopiert und angepasst werden können, unterstützen das Erlernen von HyperTalk sehr.

Die Umgangssprachlichkeit in HyperTalk wird u.a. durch eine enorme Menge an Schlüsselworten erreicht (Wheeler, 2004, S. 3). Des Weiteren werden Klammern und ähnliche für die englische Sprache ungewöhnliche Sonderzeichen weitestgehend vermieden. Eine Ausnahme bilden hierbei Funktionsaufrufe (siehe 2.2.5) sowie das Abschlusszeichen ¬ (siehe Auflistung 2.3), das benötigt wird, wenn lange Befehle auf zwei Zeilen aufgeteilt werden sollen (Becker u. Dörfler, 1991, S. 44).

[19] Für eine BNF der Grammatik siehe: DeVoto, Jeanne A. E.: HyperTalk BNF. `http://www.jaedworks.com/hypercard/scripts/hypertalk-bnf.html` (Abgerufen am 28.8.2009)

Außerdem wird die Umgangssprachlichkeit durch einigen „syntaktischen Zucker“ ermöglicht. So kann anstelle von `1` auch `one` oder `first` geschrieben werden (Wheeler, 2004, S. 3).

Zusätzlich haben viele Kommandos optionale Teile. Beispielsweise kann bei `go to next card` das `to` ohne Änderung der Bedeutung weggelassen werden oder ein `the` vor `next` eingefügt werden (Goodman, 1998b, S. 464).

Die Lesbarkeit wird auch durch die Verwendung der impliziten lokalen Variable `it` erhöht, sie enthält das Ergebnis des letzten Kommandos. Dies demonstriert folgender Beispiel-Schnipsel:

```
answer "Nach welchem Kriterium sortieren?:" ¬
with "Tiername" or "Klasse" or "Abbrechen"
if it is "Tiername"
then [...]
```

Auflistung 2.3: Aus Becker u. Dörfler (1991, S. 69)

Nachrichtenaustausch

HyperTalk ist eine prozedurale, imperative Sprache mit Schleifen, bedingten Anweisungen und Variablen. Sie funktioniert nach dem Prinzip des Nachrichtenaustauschs (Goodman, 1998a, S. 393). Nachrichten werden dabei entweder vom System erzeugt oder von HyperTalk-Skripten (Wheeler, 2004, S. 4). Nachrichten, die das System erzeugt, sind beispielsweise Reaktionen auf Benutzereingaben wie `mouseUp`.

Zusätzlich kann der Benutzer eines Stapels Code in die *Message Box* eingeben. Die so eingegebenen HyperTalk-Befehle werden sofort ausgewertet und stellen eine zusätzliche Quelle für Nachrichten dar.

Objektorientierung

HyperTalk ist insofern objektorientiert, als dass Nachrichten an Objekte versendet werden. Diese Objekte sind Instanzen der fünf HyperCard-Objekt-Klassen (siehe 2.2.2).

In einem HyperCard-Stapel gibt es keinen zentralen Ort, an dem sich der ganze HyperTalk-Code befindet. HyperTalk-Skripte sind viel mehr immer einem Objekt angeheftet (Goodman, 1998a, S. 384).

Im Gegensatz zu heutzutage geläufigen objektorientierten Sprachen gibt es nur Objekte, die den fünf vordefinierten Klassen entsprechen. Der Programmierer kann keine eigenen Klassen definieren. Auch zusammengesetzte Datentypen[20] gibt es nicht. Als Ersatz werden speziell formatierte Strings verwendet. Beispielsweise liefert die Funktion `date` einen String in der Form `"3/4/94"` zurück (Goodman, 1998b, S. 741).

[20] Zusammengesetzte Datentypen werden in C als `struct` und in Pascal als `record` bezeichnet.

Kommandos und Funktionen

HyperTalk unterscheidet zwischen Kommandos und Funktionen. Dies lässt sich damit erklären, dass HyperTalks logische Struktur sich an Pascal orientiert (Wheeler, 2004, S. 1).

Sowohl Kommandos als auch Funktionen können als Reaktion auf eine Nachricht ausgeführt werden. Der Hauptunterschied ist, dass Funktionen ein Ergebnis zurückliefern. Dabei handelt es sich immer um einen String (Goodman, 1998b, S. 741).

Ein weiterer Unterschied besteht bei der Aufrufsyntax. Kommandos verzichten zur Erhöhung der Lesbarkeit komplett auf Klammerung der Parameter. Bei Funktionen mit mehr als einem Argument hingegen ist Klammerung Pflicht. Bei Funktionen mit weniger als zwei Argumente kann die Klammerung weggelassen werden. Dann ist das Voranstellen von `the` erforderlich. Folgende Beispiele sollen dies Verdeutlichen.

```
the date
date()
the sin of 50
sin(50)
max(10,30,40)
```

Auflistung 2.4: Klammerung bei Funktionen. Aus Goodman (1998b, S. 742)

Zu diesen technischen Unterschieden kommt noch ein semantischer: Funktionen sollen keine Seiteneffekte haben (Goodman, 1998b, S. 740).

2.2.6 Kritische Bewertung

Nachdem nun alle wesentlichen Bestandteile von HyperCard erläutert wurden, wird in diesem Abschnitt abschließend HyperCard kritisch bewertet.

Vernetzung

HyperCard-Stapel sind an einen Computer gefesselt. Sie sind nicht dafür gebaut über das Internet von mehreren Benutzern gleichzeitig abgerufen zu werden. HyperCard-Stapel wurden zumeist auf Disketten verteilt.

Eine gewissen Kollaboration lässt sich erreichen, indem die Stapel-Datei auf einem Netzwerklaufwerk gespeichert wird. Doch auch dann kann auf den Stapel nur nacheinander von mehreren Personen zugegriffen werden.

Bill Atkinson selbst äußerte sich 2002 in einem Interview[21] zu dem verpassten Potential von HyperCard so:

[21] Kahney, Leander: HyperCard: What Could Have Been. `http://www.wired.com/gadgets/mac/commentary/cultofmac/2002/08/54370`. Version: August 2002. (Abgerufen am 1.9.2009)

> “ ‘I grew up in a box-centric culture at Apple. If I’d grown up in a network-centric culture, like Sun, HyperCard might have been the first Web browser.’ [. . .]
>
> ‘You don’t transfer someone’s website to your hard drive to look at it. You browse it piecemeal.... It’s much more powerful than a stack of cards on your hard drive.’
>
> ‘With a 100-year perspective, the real value of the personal computer is not spreadsheets, word processors or even desktop publishing,’ he added. ‘It’s the Web.’ ”

Moduslastigkeit

Neben der gerade bemängelten Mehrbenutzerfähigkeit ist ein weiter Kritikpunkt die Moduslastigkeit von HyperCard. Will ein Anwender einen Knopf verschieben oder anderweitig bearbeiten, muss er erst das Knopfwerkzeug auswählen. Ebenso muss zum Bearbeiten von Textfeldern das Texfeldwerkzeug ausgewählt werden (Becker u. Dörfler, 1991, S. 26f). Wenn der Anwender das falsche Werkzeug aktiviert hat, kann dadurch jedoch kein Schaden angerichtet werden. Ist beispielsweise das Texfeldwerkzeug ausgewählt und der Anwender versucht einen Knopf zu verschieben, passiert nichts. Anders sieht dies jedoch beim Verwechseln von Vorder- und Hintergrundmodus aus. Will der Anwender beispielsweise ein neues Textfeld auf einer bestimmten Karte einfügen, befindet sich aber im Hintergrundmodus, so wird das neue Textfeld auf allen Karten mit diesem Hintergrund eingefügt. Wie bereits erwähnt, wird der aktuelle Modus nur unzureichend visualisiert.

Wie auch Morphic, vermeidet HyperCard eine Unterscheidung zwischen Editier- und Ausführmodus. Dadurch wird der Anwender von der Last, sich den aktuellen Modus merken zu müssen, befreit (Maloney u. Smith, 1995, S. 25). Durch die Einführung des Hintergrundmodus in HyperCard muss sich der Anwender jedoch wieder einen Modus merken.

Des Weiteren fühlt sich HyperCard nicht so gradlinig an, da es sowohl zum Bearbeiten von Knöpfen als auch zum Bearbeiten von Textfeldern spezielle Werkzeuge verwendet.

Hintergrund-Metapher

Aber nicht nur der Hintergrund*modus* ist kritisch zu sehen, auch die Umsetzung der Hintergrund-Metapher insgesamt ist nicht optimal. Klar ist jedoch, dass HyperCard einen großen Teil seiner Mächtigkeit durch das Hintergrund-Konzept bezieht. Trotzdem oder gerade deswegen ist das Hintergrund-Konzept, mit all seinen Feinheiten und der starken Abhängigkeit von Hintergrund zu Karte, nicht leicht erlernbar. Damit steht es in Kontrast zu der Einfachheit der meisten anderen Konzepte in HyperCard. Des

Weiteren erscheint es unnatürlich, dass der komplette Inhalt des Hintergrunds auf allen Karten identisch ist und nur der Inhalt der Textfelder anders behandelt wird.

Graphik – kein Objekt

Knöpfe, Textfelder, Hintergründe, Karten und Stapel sind richtige Objekte. Sie haben Eigenschaften, die sowohl vom Benutzer direkt als auch programmatisch verändert werden können. Ebenso können sie HyperTalk-Skripte enthalten. Anders verhält es sich jedoch mit der Grafik: Sie fällt aus diesem Konzept heraus. Dies führt zu den bereits erwähnten Nachteilen, die jedoch nicht unüberwindbar sind. Die Pixelorientierung der Graphik ist jedoch, vor allem für HyperCard-Anfänger von Vorteil, da das Prinzip leicht verständlich und mit hoher Wahrscheinlichkeit bereits bekannt ist.

Primitive Suche

Aufgrund der heutigen Allgegenwärtigkeit des Internets benutzen Endandwender das Web oft zum Suchen. Sie suchen nach Büchern, nach Autos, nach Wohnungen und vielem mehr. Deshalb haben sie viel Erfahrung mit spezialisierten Suchformularen und Ergebnisseiten. HyperCards eingebaute Suchfunktion orientiert sich im Gegensatz dazu an der aus Texteditoren. Wie auch bei der Graphik hat sich Bill Atkinson für die einfachere Alternative entschieden, so dass Anfänger sich leichter zurecht finden können.

Von HyperCard lernen

Der Erfolg von HyperCard liegt nicht zuletzt in seiner großen Verbreitung begründet. Jahrelang wurde jeder neue Macintosh Rechner mit einer Version von HyperCard ausgeliefert. So war die Hemmschwelle, HyperCard auszuprobieren, gering.

Bemerkenswert ist, dass HyperCard-Anfängern schnelle Erfolge ermöglicht werden. Dies wird durch Verwendung bereits geläufiger Konzepte sowie verständlicher Metaphern erreicht. Anfänger können bestehende Anwendungen nutzen und kleine Veränderungen vornehmen. Darüber hinausgehend können Anfänger bereits erste eigene Anwendungen erstellen, ohne in HyperTalk programmieren zu müssen. Dies wird durch die Möglichkeit mit Hilfe der Hintergrund-Metapher Vorlagen definieren zu können erreicht und dadurch, dass Knöpfe aus anderen Stapeln mitsamt ihres HyperTalk-Skriptes kopiert werden können, unterstützt. Eine Wiederverwendung per Copy-and-Paste ist zwar nach Sofware-Engineering Gesichtspunkten nicht wünschenswert, für kleinere Anwendungen aber scheinbar ausreichend.

Trotz der Beschränkung auf wenige Grundbausteine ist HyperCard jedoch mächtig und eröffnet viele Anwendungsmöglichkeiten. Viele Anforderungen, die über die Möglichkeiten von Benutzern in Stufe zwei hinausgehen, können mit Hilfe von HyperTalk-

Skripten realisiert werden. Professionelle Entwickler können überdies HyperTalk via XCMDs beliebig erweitern.

Zusammenfassend kann man sagen, dass HyperCard die Forderung von Alan Kays „Simple things should be simple, complex things should be possible.“[22] erfüllt.

[22] Leuf, B. und Cunningham, W. The Wiki way: quick collaboration on the Web, Addison-Wesley Longman Publishing Co., Inc., 2001

Kapitel 3

Das Konzept von WebCards

3.1 Von HyperCard zu WebCards

WebCards ermöglicht Anwendern das kollaborative Erstellen und Bearbeiten von datenzentrierten, graphischen Webanwendungen. Um dies möglichst einfach zu gestalten, werden einige Metaphern und Konzepte aus HyperCard übernommen und wenn nötig angepasst.

Ähnlich wie in HyperCard ist eine WebCards-Anwendung ein Stapel von Karten. Im Gegensatz zu HyperCard sind die Klassen der Objekte, die sich auf ein Karte befinden können, nicht vorher festgelegt. Einer Karte können beliebige Morphs hinzugefügt werden. Dies kann via Drag-and-Drop passieren oder auch programmatisch.

WebCards verzichtet auf spezielle Werkzeuge oder Modi zum Bearbeiten von Morphs, sondern erlaubt dem Benutzer Morphs so zu bearbeiten, wie er es aus Lively Kernel gewohnt ist. Zusätzlich stellt WebCards sogenannte Halo-Menüs zur Verfügung, wie sie auch in Lively Fabrik (Lincke u. a., 2009) verwendet werden. Die Halos werden angezeigt, wenn sich der Mauszeiger über dem betreffenden Morph befindet. Sie bieten die Möglichkeit den Morph per Klick zu verschieben oder zu entfernen. Des Weiteren ermöglichen sie das Anzeigen einer Undo-Liste (siehe 3.4) sowie eines Dialoges zum Bearbeiten der wichtigsten visuellen Eigenschaften des Morphs.

Änderungen an einem WebCards-Stapel müssen nicht durch Aufrufen einer speziellen Speicherfunktion persistiert werden, dies geschieht automatisch.

3.1.1 Aufteilung des Hauptfensters

Das Hauptfenster von WebCards (siehe Abbildung 3.1) zeigt links einen Lagermorph. Dieser enthält Morphs, die auf einer Karte platziert werden können. Beim Klick auf

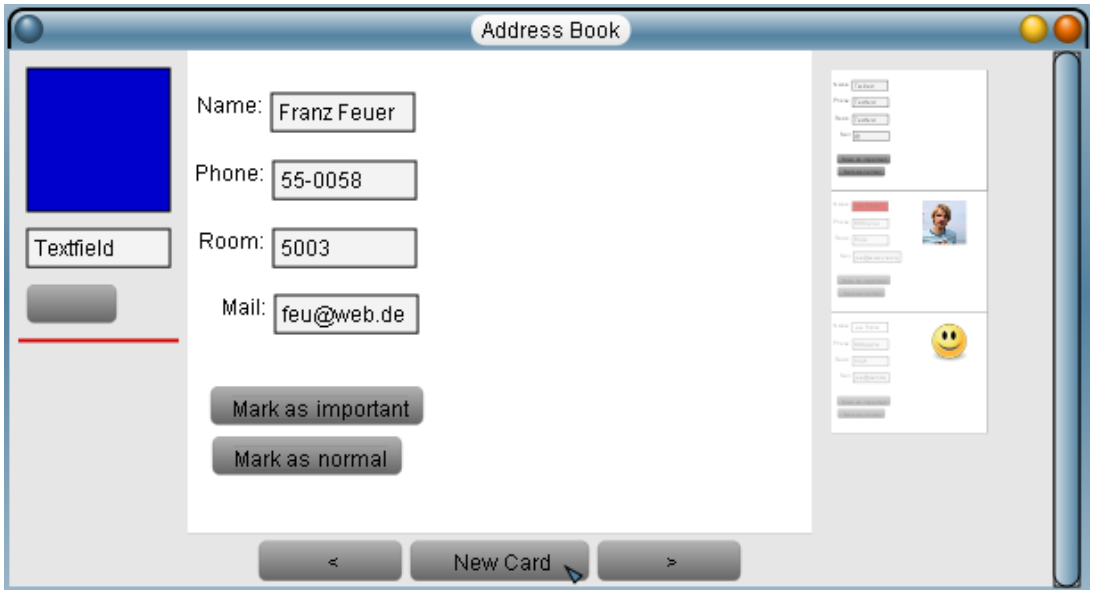


Abbildung 3.1: Hauptfenster von WebCards

einen Morph aus dem Lager wird dieser dupliziert. Der neue Morph folgt solange dem Mauszeiger, bis er mit einem weiteren Klick auf der Karte abgelegt wird. Der Lagermorph enthält einige spezielle Mophs, darunter einen Button-Morph sowie einen Textfeld-Morph. Des Weiteren können auch andere Morphs auf dem Lager abgelegt werden, die dann ebenso per Klick kopiert und platziert werden können.

In der Mitte des Hauptfensters ist die aktuelle Karte zu sehen. Darunter befinden sich drei Knöpfe. Der mittlere dient dem Erzeugen neuer Karten, die anderen beiden wechseln die aktuelle Karte. Dabei zeigt der rechte Knopf die nächste Karte im Stapel an, der linke die vorherige.

Auf der rechten Seite des Hauptfensters befindet sich die Voransicht: Sie zeigt untereinander alle Karten des aktuellen Stapels in einer verkleinerten Version an.

3.1.2 Abgrenzung von Domain- und UI-Daten

In einer klassischen Webanwendung lassen sich Domain-Daten gut von UI-Daten abgrenzen. Am Beispiel eines Adressbuches in einer webbasierten E-Mail-Anwendung wird dies deutlich. Ein Benutzer kann einen neuen Adressbucheintrag anlegen, indem er das entsprechende HTML-Formular ausfüllt und anschließend den Speicher-Knopf betätigt.

Die in das Formular eingegebenen Daten sind die Anwender-Daten bzw. Domain-Daten. Diese werden typischerweise in einer Datenbank gespeichert. Das Aussehen des Formulars hingegen kann von dem Benutzer nicht beeinflusst werden. Dieses wurde von

einem Anwendungsentwickler festgelegt. Bei dem Design des Formulars handelt es sich um die UI-Daten.

Eine solche Unterscheidung fällt in WebCards jedoch aus mehreren Gründen schwer. Zunächst dient WebCards der benutzerschnittstellengetriebenen Web-Entwicklung. Das heißt, WebCards wird verwendet, um UIs zu erzeugen. UIs sind also die Domain-Daten von WebCards. Des Weiteren gibt es in WebCards keine Unterscheidung von Entwicklungs- und Ausführungsmodus; auch dies erschwert eine Abgrenzung von Domain- und UI-Daten.

Die Abgrenzung wird abschließend dadurch unmöglich gemacht, dass Anwender in Morphic und somit auch in WebCards, jeden beliebigen Morph einfach verändern können. So kann die Position oder die Farbe eines Textfeldes genauso einfach wie der Inhalt geändert werden. Solche Änderungen sind jedoch nicht unbedingt Änderungen am Design, sondern können auch Anwenderdaten darstellen. Beispielsweise kann die Farbe eines Textfeldes eine Dringlichkeitsstufe repräsentieren. Ebenso kann die Position eines Textfeldes, das beispielsweise die schriftliche Kurzfassung einer Aufgabe enthält, die Zuordnung dieser Aufgabe zu einer Person versinnbildlichen.

Deshalb werden alle visuellen Eigenschaften eines Morphs, wie Größe, Farbe, Form, Rotation, Skalierung und Position, in WebCards als Anwenderdaten betrachtet.

3.2 Das Konzept Hintergrund

Ein wichtiges Konzept aus HyperCard sind Hintergründe. Sie können zunächst verwendet werden, um auf mehren Karten die gleichen Elemente zu platzieren und so ein einheitliches Design aller Karten zu erreichen. Darüber hinaus können Hintergründe verwendet werden, um Vorlagen für gleichartige Datensätze zu definieren. Wie oben beschrieben, können sich in HyperCard auf Hintergründen Textfelder befinden. Auf jeder Karte, die diesen Hintergrund hat, können diese Textfelder andere Inhalte haben.

Die Idee, dass Benutzer Vorlagen anlegen können, geht zurück auf Sketchpad (Sutherland, 1963). In Sketchpad können Benutzer sogenannte „Master drawings“ anlegen, die als Vorlage für Instanzen dienen. Solche Instanzen können in Position, Skalierung und Rotation vom „Master drawing“ abweichen.

3.2.1 Definieren eines Hintergrunds

In HyperCard sind Hintergründe stark von Karten abhängig (siehe 2.2.2). Ein Hintergrund existiert nur, wenn es mindestens eine Karte mit diesem Hintergrund gibt. Um einen Hintergrund zu bearbeiten, muss man auf eine Karte mit diesem Hintergrund wechseln und dann einen speziellen Modus aktivieren.

WebCards behandelt Hintergründe weitestgehend wie normale Karten. Um einen Hintergrund zu erzeugen, muss lediglich eine normale Karte erzeugt werden. Auf dieser können dann, wie auf jeder Karte, Morphs platziert werden. Anderen Karten kann diese Karte dann als Hintergrund zugewiesen werden. Um den Hintergrund zu bearbeiten, muss kein spezieller Modus verwendet werden. Der Benutzer muss dazu einzig auf die Karte wechseln, die als Hintergrund definiert wurde.

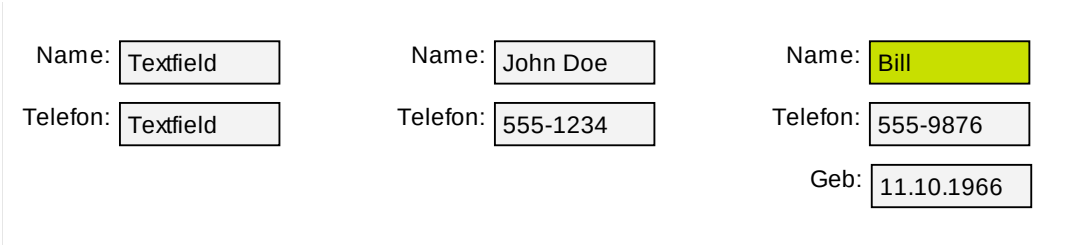
Aus Gründen der Einfachheit ist es nicht möglich, dass eine Karte, die einen Hintergrund definiert, selbst einen Hintergrund hat. So werden unter anderem Zyklen von vornherein ausgeschlossen und das Verhalten bei Änderungen an einem Hintergrund wird für den Benutzer vorhersagbarer. Gleichzeitig verringert diese Einschränkung jedoch auch die Mächtigkeit des Hintergrundkonzepts. Einige Fälle, in denen es aus Gründen der Wiederverwendung wünschenswert wäre, dass Hintergründe ebenfalls Hintergründe haben können, ließen sich damit abdecken, dass Karten mehrere Hintergründe gleichzeitig haben können. Dies ist für eine spätere WebCards Version angedacht.

3.2.2 Änderungen an Hintergrund-Morphs

In Lively Kernel kann ein Anwender jederzeit die Position und andere Eigenschaften eines Morphs verändern. Auf einer Karte mit einem bestimmten Hintergrund können Morphs, die auf dem Hintergrund definiert wurden (im Folgenden Hintergrund-Morphs genannt), genauso einfach wie normale Morphs verändert werden.

Es gibt viele Möglichkeiten, was bei Änderungen von Eigenschaften eines solchen Hintergrund-Morphs passieren soll. Dies hängt von dem gedanklichen Modell, der Metapher, ab. Ein Hintergrund kann als Folienmaster, wie er aus Microsoft PowerPoint bekannt ist, verstanden werden. Des Weiteren kennen viele Anwender auch das Ebenen-Konzept aus Bildbearbeitungsprogrammen. Für das Verhältnis von Hintergrund zu Karten mit diesem Hintergrund gibt es auch aus dem Bereich der Programmierung einige gedankliche Bilder. Der Hintergrund kann als Prototyp oder als Klassendefinition und die Karte mit dem Hintergrund als Instanz begriffen werden. Ebenso kann der Hintergrund als Klasse und die Karte mit diesem Hintergrund als Subklasse angesehen werden. Des Weiteren kann das aus Datenbanken bekannte Verhältnis von Tabellendefinitionen zu Tabelleneintrag als gedankliches Modell dienen.

Die Metapher Folienmaster betont besonders das einheitliche Design. Bei dieser Metapher sollen alle Morphs, die auf einem Hintergrund definiert wurden, auf allen Karten mit diesem Hintergrund gleich sein. Um dies zu erreichen, könnte man versuchen das Verändern von Hintergrund-Morphs nur direkt auf der Karte, die den Hintergrund definiert, zu erlauben und nicht auf Karten mit diesem Hintergrund. Alternativ ist es auch denkbar, dass Hintergrund-Morphs direkt auf Karten mit dem entsprechenden Hintergrund bearbeitet werden können, und dass diese Veränderungen sich dann auf alle Karten mit diesem Hintergrund auswirken. Dies ist etwas direkter und der Anwender muss nicht in einen speziellen Modus o.ä. wechseln. Allerdings besteht die Gefahr,



(a) Hintergrund/Vorlage

(b) Karte mit Hintergrund

(c) weitere Karte mit Hintergrund

Abbildung 3.2: Hintergrund, sowie Karten mit diesem Hintergrund

dass ein Benutzer sich nicht bewusst ist, dass es sich bei einem bestimmten Morph um einen Hintergrund-Morph handelt und sich die Änderungen an diesem Morph auf alle Karten mit diesem Hintergrund auswirken.

Das Konzept, dass alle Morphs eines Hintergrunds auf allen Karten mit diesem Hintergrund gleich sind, hat einen entscheidenden Nachteil. Zwar kann so ein einheitliches Design erzielt werden, es ist aber nicht möglich Vorlagen für ähnliche Datensätze wie Adressbucheinträge zu definieren.

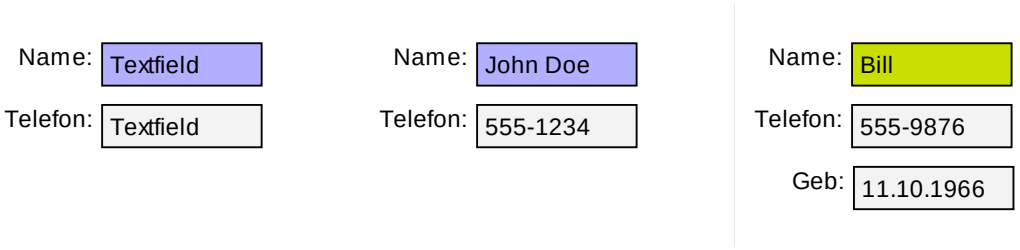
Um mit Hintergründen sowohl ein einheitliches Design als auch Vorlagen für ähnliche Datensätze zu schaffen, wird der Inhalt von Textfeldern in HyperCard gesondert behandelt. Der Inhalt von Textfeldern ist die einzige Eigenschaft, die je Karte mit diesem Hintergrund von der Vorlage abweichen kann.

Auch in WebCards sollen Hintergründe beiden Zwecken dienen. Bei der Entwicklung von WebCards stellte sich die Frage, welche Eigenschaften eines Morphs, zusätzlich zum Textinhalt, von der Vorlage abweichen dürfen. Oder anders ausgedrückt: Was sind Domain-Daten, was sind UI-Daten? Wie in 3.1.2 beschrieben, werden in WebCards alle visuellen Eigenschaften eines Morphs als Anwenderdaten betrachtet. Sie dürfen deshalb von der Vorlage abweichen.

In Abbildung 3.2(a) ist ein Hintergrund zu sehen, der als Vorlage für zwei Karten dient. Abbildung 3.2(b) zeigt eine Karte, die diesen Hintergrund hat, die Textfelder wurden ausgefüllt. Auch die Karte auf Abbildung 3.2(c) hat diesen Hintergrund. Auch hier wurden die Textfelder ausgefüllt. Zusätzlich wurde die Farbe des ersten Feldes geändert. Des Weiteren wurde dieser Karte ein drittes Textfeld hinzugefügt.

3.2.3 Änderungen an der Vorlage

Während der Entwicklung von WebCards stellte sich weiterhin die Frage, wie sich Änderungen an der Vorlage auf bereits existierende Karten mit diesem Hintergrund auswirken sollen. Die simpelste und somit für den Benutzer vielleicht auch am einfachsten



(a) Hintergrund/Vorlage (b) Karte mit Hintergrund (c) weitere Karte mit Hintergrund

Abbildung 3.3: Änderung am Hintergrund

zu verstehende Möglichkeit ist, dass sich Änderungen an der Vorlage auf bereits bestehende Karten gar nicht auswirken, sondern nur für neue Karten mit diesem Hintergrund ihre Wirkung entfalten. Insbesondere dann, wenn das Hintergrundkonzept verwendet wird um ein einheitliches Design zu erzielen, ist dieses Verhalten jedoch unpraktisch. In diesem Fall ist es sinnvoll, wenn sich das Verschieben eines Hintergrund-Morphs auf allen Karten mit diesem Hintergrund auswirkt. Dies lässt sich erreichen, indem alle Änderungen an Vorlage-Morphs ebenfalls direkt auf allen Karten mit diesem Hintergrund ausgeführt werden.

Das direkte Übernehmen aller Änderungen an Vorlage-Morphs hat jedoch auch seine Nachteile, da dies das einheitliche Aussehen betont und der Verwendung von Hintergründen für Datensätze kaum Rechnung zollt. Dies wird durch das Weiterdenken des Beispiels aus Abbildung 3.2 deutlich: Dort wurde bereits auf einer Karte die Farbe des Namenfelds geändert. Wenn nun die Farbe des Namenfelds in der Vorlage geändert wird, so würde ein direktes Übernehmen dieser Änderung auch die Farbe des Namenfelds auf dieser Karte überschreiben. Ebenso würde ein Ändern des Inhalts von „Textfield“ auf „Hier Namen eingeben“ sämtliche bereits im Adressbuch befindlichen Namen überschreiben.

In WebCards werden deshalb die visuellen Eigenschaften eines Vorlage-Morphs als Standardwerte betrachtet, die in Karten mit diesem Hintergrund überlagert werden können. Eine Änderung eines solchen Standardwertes in der Vorlage schlägt sich also nur dann in einer bereits existierenden Karten mit diesem Hintergrund nieder, wenn dort der Standardwert noch nicht ersetzt wurde.

Für das Beispiel aus Abbildung 3.2 bedeutet das, dass sich auch die Farbe des Namensfeldes auf der ersten Karte ändert, wenn die Farbe des Namensfeldes in der Vorlage geändert wird. Die Farbe auf der zweiten Karte bleibt jedoch gleich, da hier bereits der Standardwert ersetzt wurde (siehe Abbildung 3.3).

Dieses Verhalten trägt beiden Hauptanwedungen von Hintergründen, dem Erreichen eines einheitlichen Designs sowie der Verwendung als Vorlage für gleichartige Datensätze, Rechnung. Dieses Verhalten ist jedoch, im Vergleich zu den vorher genannten Mög-

lichkeiten, komplex. Insbesondere, wenn eine visuelle Eigenschaft eines Hintergrund-Morphs zwar geändert, dies aber für den Anwender nicht ersichtlich ist, kann es passieren, dass einer Änderung am Vorlage-Morph nicht das erwartete Ergebnis liefert. In diesem Fall würde sich nämlich eine Änderung dieser Eigenschaft nicht auf den betroffenen Hintergrund-Morph auswirken.

Eine solche nicht sichtbare Änderung, die dafür sorgt, dass der Standartwert zwar überschrieben ist, dies aber nicht für den Anwender ersichtlich ist, kann die Verschiebung um nur einen Pixel oder ähnliches sein.

Um nicht sichtbaren Änderungen zu vermeiden, könnte man immer dann, wenn eine visuelle Eigenschaft eines Hintergrund-Morphs auf einen Wert gesetzt wird, der ähnlich dem Standardwert ist, dem Benutzer anbieten wieder den Standardwert zu verwenden. Es ist aber nicht sinnvoll in einem solchen Fall automatisch, also ohne den Benutzer zu fragen, den Standardwert zu benutzen, da das Überschreiben nicht unbedingt eine Art manueller Zurücknahme einer Aktion darstellt, sondern auch ein bewusstes Festschreiben eines Wertes sein kann.

3.3 Kollaboration

3.3.1 Klassische Kollaborationsmöglichkeiten im Web

Klassische Webseiten erlauben keinerlei Zusammenarbeit. Die Webseite wird erstellt und anschließend auf einen Webserver geladen. Mit Hilfe von Webbrowsern kann die Seite danach betrachtet, aber nicht bearbeitet werden.

Wiki Systeme hingegen erlauben das Bearbeiten durch den Betrachter der Webseite. Sie instrumentalisieren dazu den Formularmechanismus von HTML (Ebersbach u. Glaser, 2005). Der Benutzer verändert dabei den Inhalt der Webseite nicht direkt, sondern betätigt zuerst einen „Bearbeiten“-Knopf, um dann in einem Formular den Inhalt der Webseite zu editieren. Wikis sind hauptsächlich für die Bearbeitung von Texten ausgelegt. Die Formatierung des Textes erfolgt dabei über eine spezielle Syntax. Nach dem erfolgten Editieren speichert der Benutzer seine Änderungen ab; Diese sind sodann wirksam. Anschließend können andere Benutzer die Seite ebenfalls bearbeiten, allerdings ist ein gleichzeitiges Bearbeiten nicht möglich. Dies wird als asynchrone Kollaboration bezeichnet (Ohshima u. a., 2007, S. 1).

3.3.2 Asynchrone Kollaboration ohne Modi

Lively Kernel verzichtet bewusst auf eine Unterscheidung von Bearbeiten- und Ausführen-Modus, um so dem Benutzer das Arbeiten zu erleichtern.

Um in Systemen ohne diese Unterscheidung asynchrone Kollaboration zu ermöglichen, bietet es sich an, dass ein Anwender nach erfolgter Benutzung, was eine Bearbeitung

beinhalten kann, entscheiden muss, ob er die potentiellen Änderungen abspeichern möchte. Die Entscheidung, ob eine Bearbeitung erfolgt, wird also nicht im Vorhinein durch den Wechsel in den Bearbeiten-Modi getroffen, sondern im Nachhinein durch das Speichern.

Wenn so asynchrone Kollaboration ermöglicht wird, kann es passieren, dass zwei Anwender, ausgehend von der gleichen Version, Änderungen vornehmen. Wenn beide Anwender ihre Änderungen speichern wollen, liegt ein Konflikt vor.

Manche Wiki-Systeme vermeiden Konflikte, indem sie verhindern, dass mehr als ein Benutzer je Artikel in den Bearbeiten-Modus wechselt. Dies ist in Systemen ohne gesonderten Bearbeiten-Modus jedoch nicht möglich.

3.3.3 Kollaboration in WebCards

Konflikte, wie sie bei asynchroner Kollaboration auftreten können, werden in WebCards vermieden, indem WebCards synchrone Kollaboration ermöglicht. Das heißt, mehrere Anwender können zur gleichen Zeit die gleiche Karte eines Stapels betrachten und auch bearbeiten.

Die Vorteile synchroner Kollaboration gehen jedoch weit über das Vermeiden von Konflikten heraus. Systeme, die synchronen Kollaboration ermöglichen, erlauben es, dass Menschen unabhängig von ihrem aktuellen Aufenthaltsort zur gleichen Zeit zusammenarbeiten. Dabei sehen alle Teilnehmer der Kollaboration die Ergebnisse der Aktionen der anderen und können auf diese reagieren.

WebCards ermöglicht es, Webanwendungen synchron zu erstellen und auch zu nutzen. Das heißt, eine Gruppe von Menschen kann gemeinsam ein auf ihre Bedürfnisse angepasste Anwendung erstellen und anschließend benutzen. Da keine Unterscheidung zwischen Ausführen und Bearbeiten vorgenommen wird, kann die Anwendung jederzeit an die sich ändernden Anforderungen angepasst werden.

Um mit Hilfe von WebCards synchron kollaborieren zu können, müssen die Anwender keinen speziellen Modus o.ä. verwenden. Es genügt, wenn sie sich gleichzeitig auf derselben Karte befinden. Die Gruppe der Bearbeiter einer Karte kann sich kontinuierlich ändern, ohne dass die Benutzer davon beeinflusst werden.

Die Kollaboration via WebCards beschränkt sich auf den Inhalt des Stapels. Das heißt, bei Verwendung von WebCards werden nur Änderungen an Morphs, die Teil des Stapels sind, mit den potentiellen anderen Nutzern des Stapels geteilt. Andere Morphs in der Welt, in der sich WebCards befindet, werden nicht geteilt.

Auch eine asynchrone Kollaboration ist möglich, in dem Sinne, dass erst ein Benutzer eine Karte editiert und zu einem späteren Zeitpunkt ein weiterer. Allerdings ist es nicht möglich, dass ein Benutzer eine Karte oder einen Morph auf einer Karte zur alleinigen Nutzung sperrt. Es kann jederzeit ein weiterer Benutzer hinzukommen.

Wenn sich mehrere Benutzer auf einer Karte befinden, können diese, ohne dafür in einen speziellen Modus wechseln zu müssen, alle Morphs, die sich auf dieser Karte befinden, verändern und auch neue Morphs der Karte hinzufügen. Dabei wird weder die gesamte Karte noch ein bestimmter Morph für die Veränderung durch einen einzelnen Benutzer gesperrt; WebCards verhält sich insofern optimistisch im Sinne von Kung u. Robinson (1981).

Es ist also möglich, dass auf einer Karte in einem Adressbuch-Stapel ein Benutzer den Namen und ein anderer Benutzer gleichzeitig die Telefonnummer ändert. Dies kann für beide Benutzer überraschend sein. Um den Überraschungseffekt abzumildern, soll WebCards in einer späteren Version die *Workspace Awareness* (Greenberg u. a., 1996) der Bearbeiter eines Stapels erhöhen. Eine einfach zu realisierende Idee ist das Anzeigen welche Anwender aktuell welche Karte benutzen.

Wie eben erwähnt, können zwei Benutzer unterschiedliche Morphs auf derselben Karte gleichzeitig bearbeiten. Darüber hinaus können auch zwei Benutzer denselben Morph gleichzeitig verändern. Beispielsweise kann es passieren, dass ein Benutzer den Inhalt eines Textfeldes ändert und ein anderer gleichzeitig die Hintergrundfarbe dieses Textfeldes. In diesem Fall ist als Ergebnis der Inhalt und die Hintergrundfarbe des Textfeldes geändert. Wenn beide Benutzer gleichzeitig die Hintergrundfarbe des Textfeldes ändern, so kann sich natürlich nur eine Änderung durchsetzen. In einem solchen Fall können alle Benutzer mit Hilfe der Undo-Liste nachvollziehen, was passiert ist und gegebenenfalls bestimmte Änderungen rückgängig machen. Es erscheint sinnvoll, dass gerade in einem solchen Fall die kollaborierenden Benutzer sich mit Hilfe von Instant Messaging oder Voice Over IP (VoIP) verständigen können. WebCards bietet dafür jedoch keine direkte Unterstützung. Lediglich per Textfeldern auf der Karte könnten die Benutzer Nachrichten austauschen.

3.4 Rückgängig machen

Die Möglichkeit, Aktionen Rückgängig zu machen (undo), ist eine sehr nützliche Eigenschaft (Sun, 2002, S. 1). Sie ermöglicht es dem Benutzer die Funktionalität des Programms auszuprobieren, ohne dabei Gefahr zu laufen wichtige Daten zu verlieren. Ebenso kann der Benutzer verschiedene Alternativen ausprobieren.

3.4.1 Undo-Modelle

Viele Anwendungen mit Undo-Funktion folgen dabei einem linearen Modell (Cass u. a., 2006, S. 1). Angenommen, es wurden die Aktionen $A_1, A_2, \ldots A_{i-1}, A_i, A_{i+1}, \ldots A_n$ ausgeführt, und der Benutzer möchte Aktion A_i zurücknehmen, dann ist das Ergebnis im linearen Modell $A_1, A_2, \ldots A_{i-1}$. Die Aktionen $A_i, A_{i+1}, \ldots A_n$ sind zurückgenommen.

Ein Modell, das dem Benutzer wesentlich mehr Möglichkeiten zum Experimentieren einräumt, ist das selektive Modell (Cass u. a., 2006, S. 1). Bei gleicher Ausgangssituation wie zuvor, liefert das Zurücknehmen von A_i als Ergebnis $A_1, A_2, \ldots A_{i-1}, A_{i+1}, \ldots A_n$. Cass u. a. (2006, S. 2) weist jedoch darauf hin, dass hierbei nicht auf mögliche Abhängigkeiten der Aktionen $A_{i+1}, \ldots A_n$ zu A_i geachtet wird. Beispielsweise kann die Aktion A_i das Erzeugen eines Morphs sein und A_{i+1} das Setzen der Farbe dieses Mophs.

Das kaskadierende selektive Modell ist eine Erweiterung des selektiven Modells, dass mögliche Abhängigkeiten berücksichtigt. Laut Cass u. a. (2006, S. 8) ist dieses Modell, zumindest im Zusammenhang mit Standard-Anwendungen wie Zeichenprogrammen, das anwenderfreundlichste Modell.

3.4.2 Undo in kollaborativen Anwendungen

Bereits in nicht kollaborativen Anwendungen ist ein selektiver Undo-Mechanismus sehr nützlich, in kollaborativen Anwendungen wird er jedoch geradezu notwendig (Cass u. a., 2006, S. 2), da sich die Fehler eines einzelnen Benutzers auf alle Benutzer auswirken und gleichzeitig der Funktionsumfang, den es zu entdecken gilt, größer ist als bei Anwendungen, die für Einzelnutzer entworfen sind (Sun, 2002).

Deshalb ist WebCards Undo-Funktionalität selektiv. So kann ein Benutzer jederzeit zunächst unbemerkte Aktionen von anderen Benutzern zurücknehmen, ohne dabei spätere Aktionen ebenfalls zurücknehmen zu müssen.

Insbesondere beim gleichzeitigen Arbeiten auf der gleichen Karte kann es, wie oben beschrieben zu ungewollten Ergebnissen kommen. Wie auch Jupiter (Nichols u. a., 1995, S. 8) vertraut WebCards im Falle von solchen Kollisionen auf die Fähigkeiten der Anwender diese zu bemerken und selbständig zu beheben. In solchen Fällen kann der Undo-Mechanismus bemüht werden, um den gewünschten Zustand zu erreichen.

3.4.3 Undo in WebCards

Unabhängig davon, ob ein Benutzer alleine eine Karte bearbeitet, oder mehrere Benutzer kollaborieren, kann der Benutzer jederzeit Aktionen zurücknehmen. Die Aktionen werden dafür nicht, wie es aus klassischen Textverarbeitungsprogrammen bekannt ist, an einer Zentralen stelle aufgelistet, sondern sind je Morph abrufbar.

Für jeden Morph kann der Benutzer eine Undo-Liste öffnen. In dieser werden die Aktionen angezeigt, die diesen Morph betreffen (siehe Abbildung 3.4). Die Aktionen können unabhängig voneinander ausgewählt werden und zurückgenommen werden. Bereits zurückgenommene Aktionen werden speziell dargestellt und könne wiederhergestellt werden (redo).

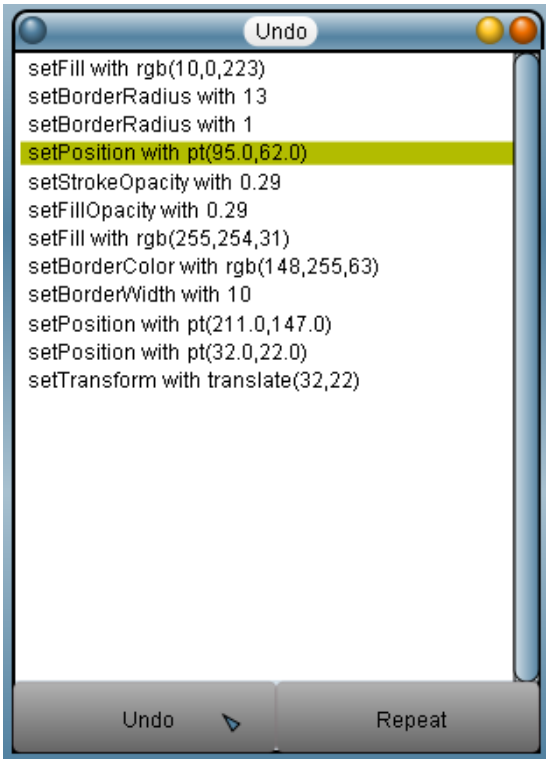


Abbildung 3.4: Undo-Liste

3.5 Programmieren in JavaScript

Als ein Erfolgsfaktor von HyperCard kann die speziell für Programmierneulinge entwickelte Programmiersprache HyperTalk gelten. Da WebCards insbesondere das Erstellen von Webanwendungen ohne die Verwendung einer Programmiersprache erleichtern soll, wurde darauf verzichtet, eine spezielle Programmiersprache zu verwenden. Anstatt dessen kann der Anwender, wie es in Lively Kernel üblich ist, Skript in JavaScript schreiben.

Die Verwendung von JavaScript hat den Vorteil, dass im Vergleich zu einer speziell für WebCards entwickelten Sprache viele Anleitungen und Referenzen zu der Sprache vorhanden sind. Des Weiteren kann ein Anwender sein zu JavaScript erworbenes Wissen leichter außerhalb von WebCards anwenden.

Kapitel 4

Implementierung von WebCards

Die Architektur von WebCards ist durch zwei Entscheidungen geprägt: erstens die Verwendung einer zentralen Datenbank und zweitens die durchgehende Verwendung von Kommando-Objekten für die Implementierung. Im nächsten Abschnitt wird die Verwendung von CouchDB (Anderson u. a., 2009) als Datenbank für WebCards erläutert.

Im darauf folgenden Abschnitt wird gezeigt, wie bei der Erweiterung von Lively Kernel um einen Undo-Mechanismus und die Möglichkeit zur synchroner Kollaboration das Problem gelöst wurde, dass Änderungen an Morphs sowohl über die UI als auch durch Skripte möglich sind.

Des Weiteren wird erläutert, wie unter Verwendung von Kommando-Objekten Undo, synchrone Kollaboration, asynchrone Kollaboration, Persistenz sowie ein Vorlagen-Konzept implementiert wurden. Abschließend wird bewertet, welche Vor- und Nachteile sich aus der gleichzeitigen Verwendung von Kommando-Objekten zur Implementierung mehrerer Anforderungen ergeben.

4.1 Zentrale Datenbank

Einer der Vorteile von Webanwendungen ist, dass sie die Daten des Benutzers, seien es E-Mails, Textdokumente oder anderes, auf einem Server speichern und so den Zugriff von jedem Computer mit Internetzugang aus ermöglichen. Um diesen Vorteil auch in Webanwendungen, die mit WebCards erstellte wurden, zu erzielen, werden die mit WebCards erstellten Stapel mit Hilfe einer Datenbank zur Verfügung gestellt.

Zunächst war angedacht, wie bei klassischen Webanwendungen, lediglich dezidierte Domain-Daten auf einem Datenbankserver zu speichern. Doch wie in 3.1.2 beschrieben, ist es nicht möglich in Morphic UI-Daten von Domain-Daten zu trennen, ohne dabei die Anwendungsmöglichkeiten von WebCards stark einzuschränken. Deshalb wer-

den alle visuellen Eigenschaften der Morphs, die den Stapel bilden, in der Datenbank gespeichert.

Bei ersten Überlegungen wurde als Datenbank eine relationale Datenbank angedacht, da solche sich bei klassischen Webanwendungen bewährt haben. Schnell stellte sich jedoch heraus, dass relationale Datenbanken im vorliegenden Fall nicht uneingeschränkt geeignet sind, da Objekte gespeichert werden sollen, bei denen nicht von vornherein feststeht, welche Felder sie haben. Bei der Suche nach anderen Möglichkeiten wurde CouchDB als eine vielversprechende Alternative ausgemacht. Eine vollständige Evaluierung, bei der CouchDB mit anderen potentiellen Kandidaten[1] verglichen wird, wurde jedoch nicht durchgeführt. Im Folgenden Abschnitt wird nun ein kurzer Überblick zu CouchDB gegeben.

4.1.1 CouchDB

Apache CouchDB[2] ist eine in Erlang geschriebene Datenbank. Es handelt sich jedoch um keine relationale Datenbank, die per SQL angesprochen werden kann, sondern um eine schemafreie Datenbank. CouchDB ist dokumentenbasiert und speichert Dokumente, die in JavaScript Object Notation (JSON)[3] vorliegen. Die Dokumente werden dabei ohne Hierarchie, also flach, in der Datenbank abgelegt. Jedes Dokument wird durch eine eindeutige ID identifiziert. Diese ID kann vor dem Speichern selbst festgelegt werden, andernfalls wird von CouchDB ein Universally Unique Identifier (UUID)[4] vergeben.

Die Create, Read, Update, Delete (CRUD) Operationen zum Erzeugen, Lesen, Ändern und Löschen von JSON-Dokumenten werden durch eine HTTP API, die der Representational State Transfer (REST) (Fielding, 2000) Architektur folgt, realisiert.

Abfragen, die über das simple Lesen eines einzelnen Dokumentes hinausgehen, sind mit CouchDB auch möglich, dazu wird der MapReduce Framework (Dean u. Ghemawat, 2004) verwendet. Dabei muss eine Map-Funktion sowie optional eine Reduce-Funktion an die Datenbank übergeben werden. Die beiden Funktionen werden als JavaScript-Funktionen per HTTP übergeben.

Die tiefgreifende Nutzung von JavaScript, JSON und HTTP prädestinierten CouchDB als Datenbank für WebCards.

[1] Alternativen wären: Persevere (`http://www.persvr.org/` [Abgerufen am 7.10.2009]), ChaiDB (`http://sf.net/projects/chaidb/` [Abgerufen am 7.10.2009]), MongoDB (`http://www.mongodb.org/` [Abgerufen am 7.10.2009]), SQLite(`http://www.sqlite.org/` [Abgerufen am 7.10.2009]), HSQLDB (`http://hsqldb.org/` [Abgerufen am 7.10.2009]).

[2] `http://couchdb.apache.org/`

[3] `http://tools.ietf.org/html/rfc4627`

[4] `http://tools.ietf.org/html/rfc4122`

4.1.2 Morphs zu JSON konvertieren

Um ein Objekt in der CouchDB Datenbank speichern zu können, muss es erst zu einem String, der der JavaScript Object Notation entspricht, umgewandelt werden. Dazu kann die Methode `JSON.stringify` verwendet werden. Die Standard-Implementierung stammt von Douglas Crockford, sie ist als Open Source erhältlich[5]. Neuerdings bieten auch einige aktuelle Browser diese Methode von Hause aus an[6].

Ein Morph aus Lively Kernel kann jedoch nicht einfach der Methode `JSON.stringify` übergeben werden. Dies liegt daran, dass JSON standardmäßig keinerlei Referenzierung unterstützt. Deshalb versucht `JSON.stringify` alle Unter-Objekte in die JSON-Repräsentation zu übernehmen. Da ein Morph in der Variablen `owner` immer auf seinen Eltern-Morph zeigt und im Array `submorphs` auf seine Kinder verweist, führt dies dazu, dass versucht wird alle Morphs, die sich in einer Welt befinden, zu serialisiert. Noch fataler sind jedoch zirkuläre Verweise zwischen Morphs oder anderen Objekten, diese führen dazu, dass `JSON.stringify` in eine Endlosschleife gerät.

Zur Lösung dieser Probleme wurde für WebCards die Klasse `Relaxer` eingeführt. Sie hat u.a. die Methode `objToRelaxedJso`[7], welche für ein gegebenes Objekt ein neues Objekt erzeugt, das danach problemlos mit `JSON.stringify` umgewandelt werden kann.

Es ist auch denkbar auf ein Objekt als Zwischenrepräsentation zu verzichten und direkt einen JSON-String auszugeben. Dies wäre jedoch schwieriger zu entwickeln gewesen und hätte die Flexibilität gekostet, das Ergebnis von `objToRelaxedJso` einfach verändern zu können. Durch die Zwischenrepräsentation ist dies jedoch sehr einfach möglich, wovon auch reger Gebrauch gemacht wird. Außerdem beleibt fraglich, ob das direkte Erzeugen eines JSON-Strings tatsächlich wesentlich effizienter wäre als die Verwendung einer Zwischenrepräsentation, die vom nativen `JSON.stringify` in einen JSON-String umgewandelt wird.

Die Methode `objToRelaxedJso` erfüllt mehrere Zwecke. Zunächst sorgt sie dafür, dass alle Morphs als einzelne Dokumente in der CouchDB gespeichert werden können. So können Morphs einzeln abgerufen und verändert werden, ohne dass dabei die Dokumentenrepräsentation anderer Morphs beeinflusst wird. Dazu werden im Objekt, das als Ergebnis zurückgeliefert wird, anstelle der Verweise auf andere Morphs Referenz-Objekte verwendet. Ein solches Referenz-Objekte ist in Auflistung 4.1 zu sehen. Der wichtigste Teil dieses Objektes ist die Variable `$ref`. Sie enthält die ID des CouchDB-

[5] `http://www.json.org/json2.js`

[6] Firefox 3.5: `https://developer.mozilla.org/en/Using_JSON_in_Firefox`, WebKit 528+: `https://bugs.webkit.org/show_bug.cgi?id=20031`, Internet Explorer 8: `http://blogs.msdn.com/ie/archive/2008/09/10/native-json-in-ie8.aspx`

[7] Der Methoden Name endet bewusst auf „Jso“ und nicht auf „Json“, da die Methode keinen String der der JavaScript Object Notation (JSON) entspricht, sondern ein einfaches JavaScript Object zurückliefert.

Dokumentes des ursprünglichen Morphs. Des Weiteren enthält das Referenz-Objekt die Lively Kernel interne ID des Morph sowie die Klasse des Morphs.

```
{
 "$ref": "d2c0cc2b645ab48b8108cb697d6bc6ed",
 "$id": "d2c0cc2b645ab48b8108cb697d6bc6ed:Card",
 "$type": "Card"
}
```

Auflistung 4.1: Referenz-Objekt

Die Lively-Kernel-ID des Morph muss nicht zwangsläufig der ID des entsprechenden Dokuments entsprechen. Standardmäßig vergibt LivelyKernel die IDs aller Morphs fortlaufend. Dies könnte im Fall von Kollaboration jedoch schnell zu Kollisionen führen. Um dies zu vermeiden, wird jedem Morph, der Teil eines Stapels ist, eine neue universelle ID zugeteilt. Es bietet sich an, dafür dieselbe UUID zu verwenden, die CouchDB für das entsprechende Dokumente vergibt. Deshalb sind, bis auf wenige Ausnahmen, die Variablen `$ref` und `$id` im Referenz-Objekt redundant.

Die Referenzierung von Morphs sorgt zwar dafür, dass jeder Morph in einem eigenen JSON-Dokument gespeichert wird, sie löst das Problem der zirkulären Referenzen jedoch nicht vollständig. Es ist immer noch möglich, dass Objekte, die keine Morphs sind, zirkuläre Referenzen haben.

Um dieses Problem zu lösen, bietet es sich an, Pfad-Ausdrücke als Referenzen zu verwenden[8]. Jedes Objekt wird bei Aufruf von `objToRelaxedJso(obj)` nur einmal serialisiert. Wird an andere Stelle in `obj` ein Objekt abermals referenziert, wird an dieser Stelle im Ergebnis-Objekt ein Referenz-Objekt erzeugt, dieses Referenz-Objekt hat als `$ref` jedoch keine ID sondern einen Pfad-Ausdruck. Dieser beginnt immer mit `$`, wobei das `$` für den Parameter `obj` steht. Ein Pfad-Ausdruck ist ein gültiger JavaScript Ausdruck, beispielsweise `$.shape.colors[0]`. Das heißt, wenn das Dollarzeichen durch den Namen einer Variable, die auf `obj` verweist, ersetzt wird, kann der Pfad-Ausdruck via `eval` zu dem ursprünglichen Wert ausgewertet werden[9].

Durch die Verwendung von Referenzobjekten ist es möglich, die Eigenschaften beliebiger JavaScript-Objekte abzuspeichern – mit einer Ausnahme: Funktionen.

Funktionen sind in JavaScript first-class Objekte. Das heißt, sie können genauso wie andere Eigenschaften gesetzt und gelesen werden. Um ein Objekt vollständig zu serialisieren, müssten also auch die Funktionen abgespeichert werden. JSON erlaubt jedoch nicht das Speichern von Funktionen. Als einfache Umgehung dieser Beschränkung bietet es sich an Funktionen als Strings abzuspeichern. Tatsächlich liefert der Aufruf von `toString` an einer Funktion den Quellcode dieser als String (ECMA International,

[8] Zyp, Kris. JSON Referencing Proposal and Library, Oktober 2007, `http://www.json.com/2007/10/19/json-referencing-proposal-and-library/` (Abgerufen am 10.07.2009)

[9] WebCards verwendet beim Parsen tatsächlich `eval`, da dies eine einfache Implementierung ist. Dies birgt jedoch die Gefahr, dass so ungewollt Code ausgeführt wird.

1999, 15.3.4.2). Funktionen können in JavaScript jedoch Closures bilden. Der durch die Closure gespeicherte Definitionskontext wird bei dem Ergebnis von `toString` nicht berücksichtigt und es gibt auch keine andere Möglichkeit auf diesen per Reflexion zuzugreifen. Es ist somit für die meisten Funktionen in Lively Kernel nicht möglich sie nach JSON zu serialisieren.

Grundsätzlich ist JavaScript nicht klassenbasiert, sondern verwendet Prototypen. Wie bereits in 2.1.1 erwähnt, verwendet Lively Kernel durchgängig die Prototype Bibliothek um JavaScript mit einem Klassensystem zu ergänzen, deshalb ist es möglich an jedem Morph und anderen Objekten die Klasse zu erfragen. Beim Konvertieren von Objekten verzichtet der `Relaxer` deshalb komplett darauf Funktionen umzuwandeln, sie werden einfach weggelassen. Anstatt dessen wird in der Variable `$type` der Name der Klasse des Objektes gespeichert. So kann beim Wiederherstellen eines Objektes aus seiner JSON-Repräsentation ein Objekt der angegebenen Klasse erzeugt werden.

Da es in JavaScript auch bei Verwendung der Prototype Bibliothek immer möglich ist die Funktionen eines Objektes zu verändern, kann jedoch nicht garantiert werden, dass das wiederhergestellte Objekt die gleichen Funktionen wie das Original hat. Da die Entwickler von Lively Kernel konsequent das Klassensystem verwendet haben, stellte dies in der Praxis jedoch kein Problem dar.

Somit lassen sich JavaScript-Objekte mit all ihren Eigenschaften, einschließlich der Klassen-Methoden, zu JSON umwandeln. Ein so abgespeicherter und später wiederhergestellter Morph hat jedoch nicht das gleiche Aussehen. Dies ist fatal, da aus Sicht eines Benutzers der Morph vor allem durch sein Aussehen gekennzeichnet ist. Der Grund für den Verlust der visuellen Eigenschaften des wiederhergestellten Morphs liegt in der SVG-Basierung der Morphic-Implementierung.

Ein Großteil der visuellen Eigenschaften eines Morphs ist nicht als JavaScript-Eigenschaften abgelegt sondern direkt im SVG-Knoten. Die Farbe eines Morph kann beispielsweise per `getFill` abgerufen und per `setFill` geändert werden. Beide Methoden greifen dazu via DOM direkt auf den SVG-Knoten zu.

Um trotzdem alle relevanten Eigenschaften in die JSON-Repräsentation zu übernehmen, wurden die Methoden `afterToJso` und `afterRestoreFromJso` eingeführt. Jede Klasse kann sie bei Bedarf implementieren. Wird ein Objekt einer Klasse, die diese Methoden hat, umgewandelt, so wird nach dem Umwandeln `afterToJso` aufgerufen. Als Parameter wird das erzeugte Objekt, das die Zwischenrepräsentation darstellt, übergeben. An dieses Objekt kann dann beispielsweise die Eigenschaft `$fill` mit dem Ergebnis von `getFill` gesetzt werden. Beim Wiederherstellen wird abschließend die Methode `afterRestoreFromJso` aufgerufen. In ihr kann dann der Wert von `$fill` als Parameter für einen Aufruf von `setFill` verwendet werden.

Durch die eben erläuterten Maßnahmen ist es möglich Morphs und andere Objekte zu JSON umzuwandeln. Diese JSON-Repräsentation kann dann in einer CouchDB abgespeichert werden.

4.2 Kommando-Objekte

Das Kommando ist ein klassisches Design Patter. Sein Ziel beschreiben Gamma u. a. (1995, S. 233) wie folgt:

> „Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undo operations.“

Unter Loggen verstehen Gamma u. a. das persistente Speichern der Kommando-Objekte um im Falle eines Systemausfalls diese zur Systemwiederherstellung zu verwenden (Gamma u. a., 1995, S. 236).

Des Weiteren erwähnen Gamma u. a. dass, falls der Adressat eines Kommandos sich Adressraum-unabhängig darstellen lässt, Kommando-Objekte über Prozessgrenzen hinweg transportiert werden könnten (Gamma u. a., 1995, S. 235).

Bei der Implementierung von WebCards wurden Kommando-Objekte ausgiebig verwendet. Im Folgenden wird gezeigt, wie unter Verwendung von Kommando-Objekten Undo, synchroner Kollaboration, asynchroner Kollaboration, Persistenz und die Hintergrund-Metapher implementiert werden.

4.2.1 Erzeugung der Kommando-Objekte

In GUI-Bibliotheken wie Swing werden durchgehend Kommando-Objekte verwendet, um auf Benutzeraktionen, wie das Betätigen eines Buttons, zu reagieren. Wird für die Implementierung eines Systems eine solchen Bibliothek verwendet, werden also von Anfang an Kommando-Objekte verwendet. Dies ist bei Lively Kernel nicht der Fall.

Des Weiteren ist WebCards als System gedacht, das Web-Entwicklung durch Endanwender ermöglicht. Ein Endanwender kann in WebCards einfach einen Button erstellen, der bei Betätigung die Farbe eines anderen Morphs ändert. Dank Lively Kernel ist es sogar möglich, dass der Endanwender einen Befehl, wie das Ändern der Farbe, einfach in ein Textfeld eingibt und den Befehl dann direkt ausführt.

Da Kommando-Objekte als Grundlage für einige Funktionalitäten verwendet werden, ist es nötig, dass die entsprechenden Kommando-Objekte immer erzeugt werden, egal auf welche der möglichen Arten ein Morph manipuliert wird, sei es über die grafische Benutzerschnittstelle von Lively Kernel, durch vom Endanwender erzeugte Skripte oder per direktem Befehl.

Eine mögliche Realisierung ist es, dem Anwender spezielle Werkzeuge zum Manipulieren der Morphs an die Hand zu geben, die Kommando-Objekte für die jeweilige Aktion erzeugen. Des Weiteren kann man dem Benutzer eine spezielle API zur Verfügung stellen, mit deren Hilfe er Morphs verändern kann, wobei jeweils Kommando-Objekte erzeugt werden.

Dieser Ansatz hat jedoch einige Nachteile: Zunächst ist es schwer, den Benutzer zur Verwendung der Werkzeuge zu bringen. Dies lässt sich eventuell durch weitestgehende Unterbindung der anderen, standardmäßigen Möglichkeiten erreichen. Noch schwerer ist es dem Benutzer die neue zusätzliche API nahe zu bringen, da sich die Verwendung der Original-API nicht ohne Weiteres verhindern lässt. Vor allem hat dieser Ansatz den Nachteil, dass er sich für den Anwender, der bereits Erfahrungen mit der Benutzung von Lively Kernel hat, unnatürlich anfühlt.

Die Lösung in WebCards besteht darin, dass die betreffenden Funktionen, wie `setFill`, durch Funktionen ersetzt werden, die neben dem Ausführen der Originalfunktion auch ein entsprechendes Kommando-Objekt erzeugen und weiterleiten. So werden Kommando-Objekt bei jedem Aufruf der entsprechenden Funktion erzeugt, unabhängig davon, ob der Aufruf durch ein vom Benutzer geschriebenes Skript oder von der grafischen Benutzerschnittstelle initiiert wird.

Es ist jedoch nicht sinnvoll die Methoden-Definitionen in den Klassen zu ersetzen, da dies erstens einen starken Eingriff in Lively Kernel bedeuten würde und zweitens Kommando-Objekte nur von Morphs, die Teil eines WebCards-Stapels sind, zur Ermöglichung von Kollaboration sowie der weiteren über Kommando-Objekte erzielten Funktionalität verwendet werden sollen.

Der erste Ansatz basierte darauf, die Methode `addMorph` in der Klasse `Card` zu überschreiben. Dies ist die Methode, die aufgerufen wird, um einen Morph einer Karte hinzuzufügen. In der überschriebenen Methode wurden, neben dem Aufruf der Super-Methode, die Methoden des übergebenen Morphs, die Kommando-Objekte erzeugen sollen, ersetzt. Wie bereits erwähnt, sind Funktionen in JavaScript Objekte. Deshalb können die Methoden von Objekten einfach durch andere ersetzt werden. Jeder Ersatz-Methode wurde dabei Zugriff auf die jeweilige Original-Methode gewährt. Die Ersatz-Methode kann so die Original-Methode aufrufen und zusätzlich ein entsprechendes Kommando-Objekt erzeugen.

Diese Implementierung funktionierte zwar grundsätzlich, der dafür zuständige Code war jedoch verworren und unübersichtlich, da er einige komplexe Fallunterscheidungen enthielt. Dies beginnt damit, dass abhängig von der Klasse des Morph unterschiedliche Methoden ersetzt werden müssen. Des Weiteren kann nicht für alle Methoden eine generische Ersatz-Methode verwendet werden, da je nach Original-Methode unterschiedliche Kommando-Objekte auf unterschiedliche Art und Weise erzeugt werden müssen. Zusätzlich reicht es nicht die `addMorph`-Methode der Karten zu überschreiben, da Morphs auch indirekt Teil eines Stapels werden können, indem sie einem Morph, der bereits Teil einer Karte ist, hinzugefügt werden. Es müssen also auch die `addMorph`-Methoden aller Kind-Morphs mit einer speziellen Methode ersetzt werden. Eine weitere Schwierigkeit liegt darin, dass Morphs nur wenn sie Teil eines Stapels sind, Kommando-Objekte erzeugen sollen. Deshalb müssen die Ersatz-Methoden eines Morphs wieder durch die Original-Methoden ersetzt werden, wenn der Morph nicht mehr Teil eines Stapels ist.

Bei dieser Implementierung gibt es noch ein zusätzlich zu betrachtendes Problem: An manchen Stellen im Programm, beispielsweise beim Ausführen eines Kommando-Objektes, soll lediglich die Original-Methode und nicht die Ersatz-Methode ausgeführt werden. Dazu wurde im Rahmen des ersten Ansatzes ein zusätzlicher Parameter eingeführt. Die Ersatz-Methode überprüft, ob zusätzlich zu den Argumenten für die Original-Methode als letztes Argument `true` übergeben wurde. In diesem Fall wird kein Kommando-Objekt erzeugt, sondern nur die Original-Methode aufgerufen. Bei Methoden, die ansonsten immer mit einer festen Anzahl von Parametern aufgerufen werden, stellt dies kein Problem dar. Bei Methoden mit variabler Anzahl von Parametern, wie `setTextString`, kann dies jedoch zu Verwirrungen führen.

Um die Qualität des Quelltextes zu verbessern, wurde dieser erste Ansatz durch einen kontextorientierten (Hirschfeld u. a., 2008) ersetzt. Dies führte zu wesentlich lesbarerem und weniger fehleranfälligen Quelltext. Die größten Verbesserungen bestehen darin, dass nun keine komplexe Fallunterscheidung zum Ersetzen der Methoden nötig ist und dass die Entscheidung, ob die Original-Methode, oder die Methode, die zusätzlich ein passendes Kommando-Objekt erzeugt, aufgerufen wird, nicht mehr durch die fehleranfällige Verwendung eines weiteren Parameters getroffen wird.

4.2.2 Implementierung der Kommando-Objekte

Bei den meisten verwendeten Kommando-Objekten handelt es sich um Kommandos mit wenig eigener Intelligenz. Sie rufen lediglich eine bestimmte Methode mit einem bestimmten Parameter auf, um so den Wert einer Eigenschaft eines Morphs zu setzen. Beispielsweise ruft ein Kommando-Objekt, das die Farbe eines Morphs auf Rot setzt, die Methode `setFill` an diesem Morph mit dem Parameter, der die Farbe Rot repräsentiert, auf. Auch die Methode zum Rückgängig-Machen ist nicht komplizierter. Sie ruft die gleiche Methode auf. Allerdings wird als Parameter der Wert, den die zu ändernde Eigenschaft vorher hatte, verwendet.

Für diese Kommando-Objekte wurde die generische Klasse `SetGetCommandObject` geschrieben, die als Instanz-Variablen das Ziel-Objekt, den Namen der Methode, den Parameter zum Ausführen der Aktion sowie den Parameter zum Zurücknehmen der Aktion hat. Instanzen von `SetGetCommandObject` werden beispielsweise für `setPosition`, `setFill`, `setRotation` und `setExtent` verwendet.

Eine weitere generische Klasse von Kommando-Objekten ist `AddRemoveCommandObject`. Der Unterschied zu `SetGetCommandObject` liegt darin, dass zum Zurücknehmen der Aktion derselbe Parameter, aber eine andere Methode verwendet wird. Diese Klasse wir beispielsweise für `addCard` und `addMorph` verwendet.

Nachdem bis jetzt erläutert wurde, wann und wie Kommando-Objekte erzeugt werden, wird im Folgenden die Funktionalität, die durch Kommando-Objekte implementiert ist, untersucht. Dabei werden insbesondere die Vor- und Nachteile dieser Implementierung aufgezeigt und ihre Grenzen beleuchtet.

4.2.3 Rückgängig-Machen mit Hilfe von Kommando-Objekten

Kommando-Objekte bieten sich als Implementierungsgrundlage für einen Mechanismus zum Rückgängig-Machen von Aktionen geradezu an. Dabei wird für jedes Kommando eine `undo`-Methode implementiert, diese macht den Effekt der Aktion rückgängig. Um also die letzte durchgeführte Aktion rückgängig zu machen, muss lediglich von dem zugehörigen Kommando-Objekt die `undo`-Methode aufgerufen werden.

Um ein lineares Undo-Modell (siehe 3.4.1) zu implementieren, bei dem es möglich ist die letzten k Aktionen zurückzunehmen, müssen die entsprechenden Kommando-Objekte aufbewahrt werden. Angenommen die Aktionen folgender Kommando-Objekte wurden ausgeführt: $K_1, K_2, \ldots K_{i-1}, K_i, K_{i+1}, \ldots K_n$, wobei K_n das neuste Kommando-Objekte ist. Soll Aktion i rückgängig gemacht werden, wird nacheinander die `undo`-Methode von $K_n, K_{n-1}, \ldots K_{i+1}, K_i$ aufgerufen.

Wie in 3.4.3 beschrieben, verwendet WebCards ein selektives Undo-Modell. Angenommen der Anwender möchte bei gleicher Ausgangslage K_i zurücknehmen, dann wäre ein naiver Implementierungsansatz einfach die `undo`-Methode von K_i aufzurufen. Folgendes Beispiel zeigt, warum dies nicht zum gewünschten Ergebnis führt. Für das Beispiel nehmen wir an, dass für Morph M_1 folgenden Kommandos ausgeführt wurden:

1. setColor(red)
2. setColor(blue)
3. addMorph(M_2)
4. setColor(green)

wobei setColor(green) das letzte ausgeführte Kommando darstellt[10]. Nun soll setColor(blue) zurückgenommen werden. Das gewünschte Ergebnis dabei ist, dass M_1 als Farbe grün behält, da setColor(green) als letztes Kommando ausgeführt wurde und somit die Farbe des Morphs festlegt. Das alleinige Aufrufen von `undo` am Kommando-Objekt 2 würde jedoch dazu führen, dass M_1 rot ist, da Kommando-Objekt 2 als Parameter für das Zurücknehmen der Aktion die Farbe, die M_1 vor dem Erzeugen von Kommando-Objekt 2 hatte, also rot, verwendet.

Ein verbesserter Algorithmus könnte nach dem Aufruf von `undo` am Kommando-Objekt 2 die Kommando-Objekte 3 und 4 erneut auszuführen. Dies würde dazu führen, dass alle Attribute von M_1, die durch Kommando-Objekt der Klasse `SetGetCommandObject` bestimmt werden, den korrekten Wert hätten. Kommando-Objekte, die nur im Zusammenspiel den Wert eines Attribute von M_1 bestimmten, wie `addMorph` oder `translateBy`, können jedoch nicht einfach nochmals ausgeführt werden.

[10] Im Gegensatz zu dieser Darstellung wird in der Benutzerschnittstelle von WebCards das neueste Kommando-Objekt in der Undo-List oben angezeigt.

Der Algorithmus, den WebCards verwendet, ruft deshalb auf den Kommando-Objekten 4, 3 und 2 `undo` auf und führt im Anschluss die Kommando-Objekte 3 und 4 wieder aus.

Allgemein wird, um K_i aus der Kommando-Objekt-Reihe $K_1, K_2, \ldots K_{i-1}, K_i, K_{i+1}, \ldots K_n$ zurückzunehmen, nacheinander bei $K_n, K_{n-1}, \ldots K_{i+1}, K_i$ die `undo`-Methode aufgerufen. Danach werden $K_{i+1}, \ldots K_{n-1}, K_n$ wieder ausgeführt.

Optimierungsmöglichkeiten

Die verwendete Implementierung ist unabhängig von der Aktion, die die Kommando-Objekten realisieren, und verzichtet weitestgehend auf Optimierungen.

Bei einer Reihe von Kommando-Objekten der Klasse `SetGetCommandObject`, die dasselbe Attribut, wie beispielsweise Farbe oder Position, an demselben Morph setzen, ist einzig das neuste Kommando-Objekt für den Wert des Attributs zuständig. Dies könnte als Grundlage für einige Optimierungen dienen.

Für `AddRemoveCommandObject` gilt diese Regel jedoch nicht. So sind selbstverständlich alle per `addMorph` hinzugefügten Morphs sichtbar und nicht bloß der letzte.

Die Methode `translateBy` addiert den als Parameter übergebenen Vektor zu der aktuellen Position des betreffenden Morphs hinzu. Für sie wurde eine spezielle Kommando-Klasse eingeführt. Im Gegensatz zu Kommandos der Klasse `SetGetCommandObject` sind alle Kommandos für den endgültigen Wert des betreffenden Attributs verantwortlich. Ein weiterer Unterschied ist, dass die Reihenfolge der Kommando-Objekte dieser Klasse irrelevant ist. Dies könnte für Optimierungen verwendet werden.

Für Optimierungen müssen also die Eigenschaften aller Kommando-Klassen beachtet werden. Da das Rückgängig-Machen bereits ohne Optimierungen schnell genug erschien, wurde auf Optimierungen verzichtet.

Zusätzlich zu Optimierungen des Algorithmus sind auch noch einige Verbesserungen an der Benutzersschnittstelle möglich. So könnten in der Undo-Liste (siehe Abbildung 3.4) die Kommando-Objekte der Klasse `SetGetCommandObject`, die nicht für den aktuellen Wert einer Eigenschaft verantwortlich sind, ausgegraut dargestellt werden, da es unwahrscheinlich ist, dass ein Anwender ein solches Kommando-Objekt zurücknehmen möchte, da dies keinen sichtbaren Effekte hätte.

Als weitere Verbesserung könnten Kommando-Objekte, je weiter ihre Erstellung in der Vergangenheit liegt, zu immer größeren logischen Einheiten zusammengefasst werden, da es immer unwahrscheinlicher wird, dass der Benutzer die einzelnen Aktionen zurücknehmen möchte, und so die Undo-Liste übersichtlicher wird. Auch ein Filtermechanismus, mit dessen Hilfe Benutzer die in der Undo-Liste angezeigt Kommando-Objekte filtern könnten, um beispielsweise nur Lösch-Aktionen angezeigt zu bekommen, ist denkbar.

Kaskadierendes selektives Modell

Wie bereits erwähnt (3.4.1), können Aktionen untereinander Abhängigkeiten aufweisen. Da der Benutzer in WebCards einzelne Aktionen selektiv zurücknehmen kann, müssen diese Abhängigkeiten auch beachtet werden. Die Aktionen wurden jedoch so gewählt, dass fast keine Abhängigkeiten zwischen ihnen bestehen. Die Ausnahme bildet das Erzeugen eines Morphs. Alle Aktionen, die einen Morph verändern, sind von seiner Erzeugung abhängig. Das Erzeugen ist jedoch keine gesonderte Aktion. Erst das Hinzufügen via `addMorph` wird als Kommando-Objekt repräsentiert. Das Erzeugen ist somit nicht gesondert rücknehmbar.

Wenn ein Anwender das Hinzufügen eines Morphs zurücknimmt, kann er die Undo-Liste (siehe 3.4.3) für den entsprechenden Morph nicht mehr öffnen, da die Undo-Liste über das Halo-Menü des Morphs geöffnet wird. Das Kaskadieren passiert so für den Benutzer auf natürliche und nachvollziehbare Weise.

WebCards besitzt noch kein zentrales Undo-Menü, in dem die Kommando-Objekte Morph-übergreifend angezeigt werden. In einem solchen müssten nach dem Zurücknehmen des Hinzufügen eines Morphs alle Kommando-Objekte, die diesen Morph als Ziel haben, ausgeblendet werden.

4.2.4 Persistenz und asynchrone Kollaboration mit Hilfe von Kommando-Objekten

Wie in 4.1 beschrieben, können neben Morphs auch andere Objekte als JSON in einer CouchDB abgespeichert werden. Dies gilt natürlich auch für Kommando-Objekte. WebCards speichert jedes Kommando-Objekt unmittelbar nach dem Erzeugen in der zentralen Datenbank ab. Dazu wird das Kommando-Objekt zu JSON umgewandelt und anschließend per HTTP an den Datenbank-Server gesendet.

Laut Sun u. Chen (2002, S. 2) machen Benutzer ihre Beurteilung der Qualität eines Programms stark von der Antwortzeit des Programms abhängig. Deshalb werden die Kommando-Objekte per asynchronem `XMLHttpRequest`[11] an die Datenbank gesendet.

Wenn ein Morph per `addMorph` einem Stapel hinzugefügt wird, so wird ein entsprechendes Kommando-Objekt erzeugt, das den Morph als Instanz-Variable hat. Beim Serialisieren des Kommando-Objektes wird überprüft, ob der Morph bereits in der Datenbank gespeichert wurde. Sollte dies nicht der Fall sein, so wird der Morph umgehend an die Datenbank gesendet.

Jeder Morph wird also nur beim erstmaligen Hinzufügen zum WebCards-Stapel gespeichert. Anschließende Änderungen werden durch das Speichern der Kommando-Objekte, die diese Änderungen erzeugt haben, persistiert. Es wird also nicht nur der Morph als solcher gespeichert, sondern auch seine Geschichte.

[11] `https://developer.mozilla.org/En/Using_XMLHttpRequest` (Abgerufen am 8.10.2009)

Da die Kommando-Objekte automatisch zur Datenbank gesendet werden, muss der Benutzer den Stapel nicht explizit abspeichern. Egal wann WebCards geschlossen wird, der aktuelle Stapel ist gespeichert. Dies ist insbesondere beim unbeabsichtigten Beenden von Vorteil.

Wenn der Ersteller des Stapels oder ein anderer Anwender den Stapel neu lädt, werden alle Kommando-Objekte auf einmal geladen, um nicht für jedes Kommando-Objekt eine neue HTTP-Anfrage starten zu müssen. Die Kommando-Objekte werden dann nacheinander ausgeführt. Dies passiert auch bei größeren Mengen von Kommando-Objekten in akzeptabler Zeit. Wie bereits in 4.2.3 beschrieben, müssten nicht alle Kommando-Objekte ausgeführt werden, auf solche Optimierungen wurde jedoch verzichtet. Die Änderungen werden so ausgeführt, dass der Browser keine Zwischenergebnisse anzeigt, sondern erst nach Anwendung aller betreffenden Kommando-Objekte den aktuellen Zustand. So bemerkt der Anwender, der gerade den Stapel aufgerufen hat, nicht, dass alle Kommando-Objekte nacheinander ausgeführt werden.

Die jetzige Implementierung hat den Nachteil, dass die Datenbank immer voller wird, je intensiver der Stapel benutzt wird. Die Datenbank wird auch dann voller, wenn die Anzahl der Morphs gleich bleibt oder sogar abnimmt. Es ist deshalb sinnvoll, WebCards so zu erweitern, dass es möglich ist, die Menge der gespeicherten Kommando-Objekte zu reduzieren. Dies könnte entweder automatisch oder per administrativen Befehl passieren. Dazu könnten beispielsweise alle Kommandos, die einen bestimmten Morph verändern, auf diesen angewendet werden. Anschließend würde der Morph erneut abgespeichert und die verwendeten Kommando-Objekte gelöscht.

4.2.5 Synchrone Kollaboration mit Hilfe von Kommando-Objekten

Die eben beschriebene zentrale Speicherung der Morphs und Kommando-Objekte ermöglicht asynchrone Kollaboration. Darüber hinaus erlaubt WebCards synchrone Kollaboration. Das heißt, dass eine Gruppe von Menschen zur gleichen Zeit das gleiche Objekte nicht nur betrachten, sondern auch editieren kann (Ellis u. a., 1991). Die Benutzer befinden sich dabei an unterschiedlichen Orten und sind jeweils über ihren Webbrowser mit dem Internet verbunden.

Architektur: zentraler Koordinator

Die Anwender laden jeweils wie in 4.2.4 beschrieben den Stapel. Änderungen, die sie erzeugen, werden als Kommando-Objekte an die Datenbank gesendet. Alle WebCards-Instanzen, die gerade diesen Stapel geladen haben, werden daraufhin von CouchDB über das neue Kommando-Objekt informiert. Bis auf die Instanz, die das Kommando-Objekt erzeugt hat, laden alle das Kommando-Objekt aus der Datenbank und führen es aus.

Das Informieren aller laufenden WebCards-Instanzen wurde zunächst dadurch realisiert, dass jede Instanz in einem festen zeitlichen Abstand beim Server nachfragt. Die aktuelle Version von WebCards verwendet die in CouchDB 0.10.0a806985 eingeführte Möglichkeit, sich über Änderungen von CouchDB informieren zu lassen (Anderson u. a., 2009, Kapitel 25). Dazu stellt WebCards eine asynchrone HTTP-Anfrage an CouchDB, die erst beantwortet wird, wenn es eine Änderung gab[12]. Im Vergleich zur ersten Implementierung verringert dies die Anzahl an HTTP-Anfragen erheblich und verkleinert zusätzlich den Zeitraum zwischen dem Ankommen des Kommando-Objekt beim Server und dem Informieren der anderen WebCards-Instanzen.

Synchrone Kollaboration wird in WebCards also mit Hilfe eines zentralen Koordinators (Nichols u. a., 1995) erreicht und nicht wie in Croquet (Smith u. a., 2003) durch ein Peer-to-Peer Protokoll. Peer-to-Peer Protokolle haben zwar den Vorteil, dass sie gut skalieren, aus einem Webbrowser heraus sind Peer-to-Peer Verbindungen jedoch nicht ohne Weiteres möglich.

Konsistenzerhaltung

Konsistenzerhaltung ist eine fundamentale Aufgabe in Systemen, die synchrone Kollaboration erlauben (Sun u. Chen, 2002). Viele Systeme tun dies, indem sie pessimistisch vorgehen. Das heißt, sie führen Änderungen erst aus, wenn sie sich, beispielsweise durch ein Semaphore, davon überzeugt haben, dass es zu keinem Konflikt kommen wird. Ein solches Vorgehen hat den Nachteil, dass die Änderung immer erst mit einer Verzögerung ausgeführt wird. Insbesondere bei Verbindungen über Netzwerke mit einer hohen Latenz und einer geringen Bandweite würde dies zu nicht akzeptablen Antwortzeiten führen (Nichols u. a., 1995).

Deshalb verwendet WebCards einen optimistischen Algorithmus. Das heißt, wenn ein Benutzer eine visuelle Eigenschaft an einem Morph ändert, wird diese Änderung auf der Version des Stapels dieses Benutzers sofort ausgeführt. Das entsprechende Kommando-Objekt wird asynchron an die CouchDB gesendet, von wo aus es an alle Besucher des Stapels weitergeleitet wird.

Beim optimistischen Umgang mit Nebenläufigkeit kann es zu Konflikten kommen. Ein Konflikt liegt vor, wenn zwei Benutzer gleichzeitig am gleichen Objekt das gleiche Attribut ändern. Oberstes Ziel von WebCards ist in solchen Fällen, dass alle Anwender das gleiche Ergebnis sehen.

Das Jupiter System (Nichols u. a., 1995), das ebenfalls einen zentralen Koordinator hat und Nebenläufigkeit optimistisch behandelt, verwendet dazu die von Ellis u. Gibbs (1989) vorgeschlagene Operations-Transformation[13], die auch in Google Wave[14] ver-

[12] Dieses Vorgehen wird als *Long Polling*, *HTTP server push* oder *Comet* bezeichnet

[13] engl.: Operational Transformation

[14] David Wang, Alex Mah: Google Wave Operational Transformation, `http://www.waveprotocol.org/whitepapers/operational-transform` (Abgerufen am 8.10.2009)

wendet wird. Dabei erzeugt der zentrale Koordinator für jeden Anwender des Systems eine individuelle Operation, nach deren Anwendung sein System in Einklang mit den restlichen Systemen ist.

Wenn in WebCards mehrere Benutzer gleichzeitig die Farbe eines Morphs ändern, so reicht es nicht, wenn alle Benutzer anschließend die gleiche Farbe sehen. Zusätzlich müssen alle WebCards-Instanzen alle Kommando-Objekte in derselben Reihenfolge vorliegen haben, damit alle Anwender mit Hilfe der Undo-Liste (siehe 3.4.3) nachvollziehen können, was passiert ist.

In WebCards werden Kollisionen dadurch aufgelöst, dass CouchDB allen Objekten in einer Datenbank eine fortlaufende Nummer zuteilt. Diese Sequenz-Nummer legt fest, in welcher Reihenfolge die Kommandos auszuführen sind. Da alle Kommandos in WebCards immer nur einen Morph beeinflussen, werden die Kommandos intern je Morph durchnummeriert.

Wenn auf einem Morph eine Änderung durchgeführt wird, wird für das entsprechend erzeugte Kommando-Objekt optimistischerweise angenommen, dass es das nächste Kommando-Objekt ist, das diesen Morph betrifft, und somit die Nummer des Kommandos um eins höher ist, als die des letzten Kommando-Objekts, das diesen Morph verändert hat.

Ein Kommando-Objekt, das an CouchDB gesendet wurde, wird an alle WebCards-Instanzen, die den betreffenden Stapel anzeigen, weitergeleitet, auch an die Instanz, die das Kommando-Objekt gesendet hat. Alle Instanzen bestimmen für jedes vom Server kommende Kommando-Objekt als Erstes anhand der Sequenz-Nummer die Nummer des Kommandos in Bezug auf den Ziel-Morph. Anschließend wird überprüft, ob es bereits ein Kommando-Objekt gibt, dem diese Nummer zugeteilt wurde. Wenn dies nicht der Fall ist, so handelt es sich um ein komplett neues Kommando-Objekt. Sollte jedoch bereits ein Kommando-Objekt diese Nummer haben, so liegt dies an der eben beschriebenen optimistischen Vorgehensweise. Nun gibt es zwei Möglichkeiten: Einerseits kann es sich bei dem bereits vorhandenen Kommando-Objekt mit dieser Nummer und dem empfangenen Kommando-Objekt um das gleiche Objekt handeln; dann war die optimistische Annahme korrekt. Das Kommando-Objekt muss nicht ausgeführt werden. Andererseits kann es sich auch um ein anderes Objekt handeln. Das bedeutet, dass die optimistische Annahme falsch war und ein anderer Benutzer ein Kommando-Objekt erzeugt hat, das vor dem momentan mit dieser Nummer bezeichneten Objekt auszuführen ist. In diesem Fall werden alle Kommando-Objekte, die in Bezug auf den betreffenden Morph eine Nummer tragen, die größer oder gleich der Nummer des neuen Kommandos ist, rückgängig gemacht. Anschließend wird das neue Kommando ausgeführt. Danach werden, abermals optimistisch, alle Kommandos, die im vorherigen Schritt rückgängig gemacht wurden, wieder ausgeführt. Die ihnen optimistisch zugeteilte Nummer wird um eins erhöht.

Dieser Algorithmus sorgt also dafür, dass Kommando-Objekte, die einen bestimmten Morph manipulieren, bei allen Teilnehmern in der gleichen Reihenfolge ausgeführt wer-

den. Der Algorithmus geht davon aus, dass ein Kommando-Objekt immer nur einen bestimmten Morph verändert. Des Weiteren ist es erforderlich, dass die `undo`-Methode den Effekt des Kommando-Objektes exakt zurücknimmt.

Wenn ein Kommando-Objekte immer genau ein Attribut eines bestimmten Morphs manipulieren würde, so könnte der Algorithmus so angepasst werden, dass die Kommando-Objekte nicht je Morph, sondern je Attribut durchnummeriert werden. Dies hätte den Vorteil, dass die optimistische Ausführung eines Kommando-Objektes mit einer größeren Wahrscheinlichkeit nicht zurückgenommen werden müsste. Die für diese Anpassung nötige Forderung erfüllen zwar die meisten Kommando-Objekte in WebCards, aber nicht alle. Deshalb werden die Kommando-Objekte je Morph gruppiert.

4.2.6 Grafische Vererbung mit Hilfe von Kommando-Objekten

Wie in 3.2 ausführlich beschrieben, bietet WebCards die Möglichkeit, eine Karte als Vorlage für andere Karten zu verwenden.

Wird einer Karte ein Hintergrund zugewiesen, so wird die Karte, die als Vorlage dient, mit Hilfe der Kopier-Funktionalität von Lively Kernel dupliziert. Zusätzlich speichert WebCards eine Verbindung zwischen Original und der Kopie, also zwischen der Vorlage und der Karte mit Hintergrund, ab.

Änderungen an Hintergrund-Morphs, also den Morphs, die sich als Hintergrund auf einer Karte befinden, werden genau so wie Änderungen an normalen Morphs behandelt.

Komplizierter wird es, wenn an der Vorlage Änderungen vorgenommen werden (siehe 3.2.3). Zusätzlich zum Senden eines Kommando-Objektes und den weiteren in den vorherigen Abschnitten beschriebenen Maßnahmen, muss die Änderung auf die Hintergrund-Morphs angewendet werden. Aus den in 3.2.3 beschriebenen Gründen wird jedoch nicht einfach das Kommando auf den Hintergrund-Morph angewendet. Das gewünschte Verhalten ist nämlich, dass Vorlagen Standardwerte definieren, die in Karten mit Hintergrund überlagert werden können.

In einer ersten Implementierung wurde zum Erreichen dieses Verhaltens bei Änderungen am Vorlage-Morph der entsprechende Hintergrund-Morph gelöscht, eine neue Kopie des Vorlage-Morphs erstellt und auf diese die Änderungen, die am Hintergund-Morph durchgeführt wurden, angewendet. Diese Implementierung hat jedoch den entscheidenden Nachteil, dass die Identität des Morphs verloren geht. Das heißt, wenn ein Benutzer beispielsweise ein *Style Panel* für einen Hintergrund-Morph geöffnet hat und ein andere Benutzer den Vorlage-Morph verändert, so kann der erste Benutzer das *Style Panel* dann nicht mehr benutzen, da es sich auf den gelöschten Morph bezieht. Ein solches Verhalten ist für den Benutzer nicht verständlich, da es für ihn den Anschein hat, es handele sich um denselben Morph.

Eine alternative Implementierung, bei der die Identität des Morphs erhalten bleibt, könnte ohne Kommando-Objekte auskommen, indem der Zugriffsmechanis-

mus auf Objekt-Eigenschaften verändert wird. Dazu müssten die get-Methoden des Hintergrund-Morphs so angepasst werden, dass sie erst kontrollieren, ob der angefragte Wert in diesem Hintergrund-Morph überschrieben wurde oder ob der Wert des Vorlage-Morphs abgerufen und zurückgeliefert werden muss. Dieser Implementierungsansatz wurde jedoch nicht gewählt, da er einen starken Eingriff in den bestehenden Lively Kernel Quellcode bedeutet hätte. Darüber hinaus ist eine Veränderung des Zugriffsmechanismus auf die Objekt-Eigenschaften nicht ausreichend, da in der aktuellen, auf SVG basierenden Implementierung von Lively Kernel nicht die von den get-Methoden zurückgelieferten Werte für das Aussehen eines Morphs verantwortlich sind, sondern die im entsprechenden DOM-Knoten gespeicherten Werte.

Bei der aktuellen Implementierung wird ein allgemeingültiger Algorithmus verwendet, der, wie der erste Ansatz, auf Kommando-Objekten basiert und keine Änderungen an bereits bestehenden oder zukünftigen Morph-Klassen erforderlich macht. Im Gegensatz zum ersten Ansatz erhält dieser Algorithmus die Identität des Morphs und löst so die Probleme des ersten Ansatzes. Dies wird erreicht, indem im Falle einer Änderung an einem Vorlage-Morph alle Änderungen am entsprechenden Hintergrund-Morph rückgängig gemacht werden. Anschließend wird die Änderung des Vorlage-Morphs auf den Hintergrund-Morph angewendet. Abschließend werden alle zuvor zurückgenommenen Änderungen wiederhergestellt.

Auf diese Weise wird erreicht, dass alle Kommando-Objekte, die durch Änderungen am Vorlage-Morph erzeugt wurden, logisch vor den Kommando-Objekten, die durch Änderungen am Hintergrund-Morph erzeugt wurden, ausgeführt werden. Änderungen am Hintergrund-Morph können somit Änderungen am Vorlage-Morph überlagern.

Optimierungsmöglichkeiten

Die aktuelle Implementierung hat Optimierungspotential, da immer alle Kommando-Objekte erst rückgängig gemacht und anschließend wieder ausgeführt werden, auch wenn dies nicht nötig ist. Bei solchen Optimierungen müssten die in 4.2.3 beschriebenen semantischen Eigenschaften jeder Kommando-Klasse berücksichtigt werden.

Die Verwendung der Kommando-Objekte zum Erreichen des Vorlage-Verhaltens, führt dazu, dass nicht einfach alle Kommando-Objekte eines Hintergrund-Morphs gelöscht werden können, da dann Änderungen an der Vorlage nicht mehr behandelt werden könnten. Trotzdem ist es nicht nötig alle Kommando-Objekte des Hintergrund-Morphs aufzubewahren. Bei einer Reihe von Kommando-Objekten der Klasse `SetGetCommandObject`, die alle das gleiche Attribut an dem betreffenden Hintergrund-Morph verändern, ist immer nur das letzte Kommando-Objekt für den Wert verantwortlich. Es wäre also möglich, die anderen Kommando-Objekte zu löschen.

Abweichungen vom mentalen Modell des Nutzers

Der verwendete Algorithmus basiert auf der Ausführungsreihenfolge von Kommando-Objekten. Damit die durch den Algorithmus erzielten Ergebnisse für Anwender verständlich sind, dürfen die Kommando-Objekte sich nur auf einzelne, logisch unabhängige Eigenschaften eines Morphs beziehen. Diese Forderung erfüllen Kommando-Objekte, die als Aktion den Aufruf der Methode `setVertices` haben nur bedingt, wie das folgende Beispiel zeigt. Solche Kommandos werden beim Ändern der Form eines Morphs erzeugt. Wenn ein Nutzer an einem Hintergrund-Morph, der beispielsweise die Form eines fünfzackigen Sterns hat, einen der Zacken länger zieht, so führt dies dazu, dass Änderungen der Form des Vorlage-Morph sich nicht mehr auf den Hintergrund-Morph auswirken. Ein Anwender könnte jedoch den Stern mental in fünf Zacken eingeteilt haben und deshalb erwarten, dass er nur die Form einer Zacke überschrieben hat und sich Änderungen am Vorlage-Morph, die die Form der anderen vier Zacken betreffen, auch weiterhin am Hintergrund-Morph auswirken.

Falls es tatsächlich zu Abweichungen vom mentalen Modell des Nutzers kommt, so kann der Nutzer mit Hilfe der Undo-Liste nachvollziehen, wie das System arbeitet.

4.2.7 Verwendung von Kommando-Objekten für viele Anforderungen gleichzeitig

Wie in den vorherigen Abschnitten gezeigt, verwendet WebCards Kommando-Objekte für mehrere Anforderungen gleichzeitig. Darunter befinden sich ein Undo-Mechanismus, Persistenz, Kollaboration und grafische Vererbung.

Diese Entscheidung hat den Vorteil, dass sich für den Anwender ein einheitliches Bild ergibt. Dieselben Änderungen an einem Morph, die zurückgenommen werden können, werden auch automatisch abgespeichert und sind auch bei allen anderen Besuchern des Stapels nach nur minimaler Verzögerung sichtbar. Dieselben Eigenschaften sind es auch, die in Hintergründen überlagert werden können.

Die Undo-Liste (siehe 3.4.3) sorgt dafür, dass der Benutzer leicht nachvollziehen kann, wann welche Kommando-Objekte erzeugt werden.

Ein weiterer Vorteil dieser Implementierung ist, dass WebCard einfach erweitert werden kann. Wenn beispielsweise die Klasse Morph um eine Methode zum Setzen des z-Index erweitert würde, so müsste lediglich der Layer der Morph-Klasse so erweitert werden, dass für diese Methode ein Kommando-Objekt erzeugt wird. Diese Erweiterung würde lediglich drei Zeile umfassen, es gleichzeitig aber ermöglichen, dass Änderungen des z-Index automatisch persistiert und synchronisiert werden, ebenso könnten dann diese Änderungen zurückgenommen werden und würden sich in die Hintergrund-Metapher einfügen.

Bei der Implementierung von WebCards stellte sich heraus, dass die Verwendung der gleichen Kommando-Klassen und Objekte für all diese Anforderungen neben diesen Vorteilen auch Nachteile hat, da es zu einer starken Kopplung der einzelnen Anforderungen kommt, die sowohl bei der Auswahl der Kommandos als auch beim Aufbewahrungszeitraum der Kommando-Objekte berücksichtigt werden muss.

Auswahl der Kommandos

Aufgrund der starken Kopplung müssen bei der Auswahl der Methoden, die durch Kommando-Objekte repräsentiert werden sollen, die Belange aller Anforderungen beachtet werden.

Aus der Verwendung der Kommandos für die Undo-Funktionalität ergibt sich die Anforderung, dass die Kommandos möglichst nah an dem mentalen Modell des Benutzers sind. So erwartet ein Benutzer, dass eine Änderung, die für ihn eine logische Einheit bildet, auch als eine Aktion in der Undo-Liste auftaucht.

Aus der Verwendung der Kommandos zur synchronen Kollaboration ergibt sich die Anforderung, dass es für das Rückgängigmachen eines Kommandos ebenfalls ein Kommando-Objekt geben muss, damit das Rückgängigmachen an alle Teilnehmer propagiert werden kann. Des Weiteren erfordert der Algorithmus, der zum optimistischen Umgang mit Nebenläufigkeit verwendet wird, dass die Kommandos sich immer nur auf einen Morph beziehen.

Aufbewahrungszeitraum

Nicht nur bei der Auswahl der Aktionen auch bei der Frage, wie lange Kommando-Objekte aufbewahrt werden, müssen die Belange aller Anforderungen berücksichtigt werden.

Bei der Undo-Funktionalität verhält es sich so, dass Kommando-Objekte mit zunehmenden Alter immer unwichtiger für den Benutzer werden. Als Erleichterung für den Benutzer könnten Aktionen, je länger sie zurück liegen, zu immer größeren logischen Einheiten gruppiert werden.

Aufgrund der Verwendung der Kommando-Objekte zur Persistierung können Kommando-Objekte erst dann gelöscht werden wenn die Änderungen, die sie kapseln, auf eine andere Weise, beispielsweise wie in 4.2.4 beschriebene, persitiert wurden.

Die Verwendung der Kommando-Objekte zur grafischen Vererbung erfordern hingegen, wie in 4.2.6 beschrieben, dass bestimmte Kommando-Objekte nicht gelöscht werden.

Im Rahmen der synchronen Kollaboration könnte zusätzlich die Anforderung aufkommen, dass alle Teilnehmer die gleichen Kommandos in der Undo-Liste sehen, der Aufbewahrungszeitraum der Kommandos also bei allen Teilnehmern gleich sein soll.

Kapitel 5

Evaluierung

In den vorherigen Kapiteln wurden die Konzepte und die Implementierung von WebCards vorgestellt. In diesem Kapitel werden diese nun bewertet, indem zwei mit WebCards erstellte Beispielanwendungen die Konzepte mitsamt ihrer Implementierung evaluiert.

5.1 Beispielanwendung: Planung im Team

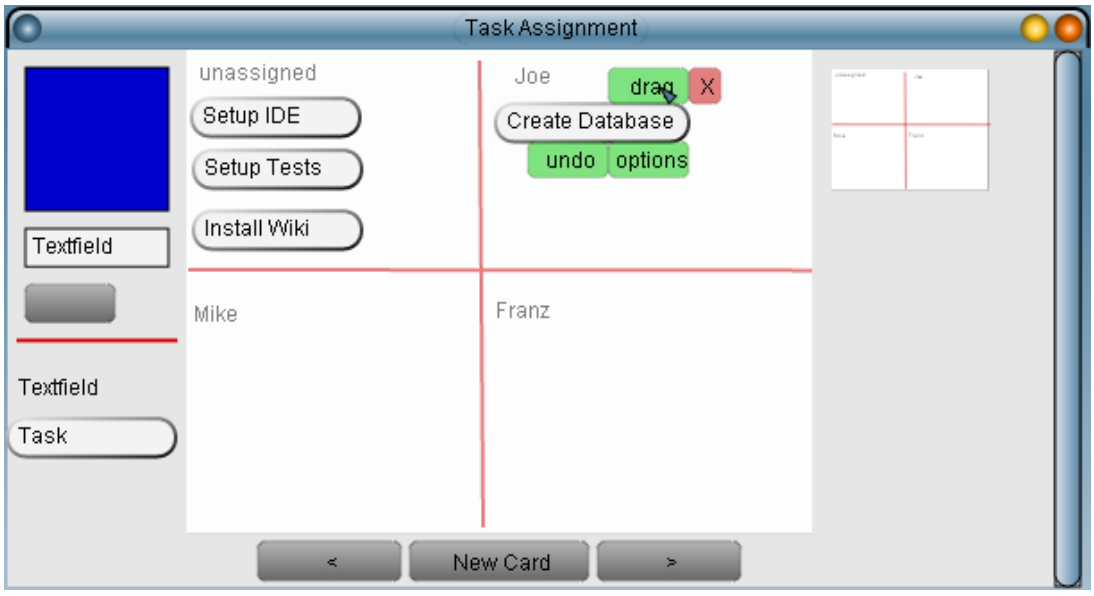


Abbildung 5.1: Screenshot der Beispielanwendung eins

Um in einem Team die Aufgaben, die in der nächsten Woche erledigt werden sollen, zu planen, wird eine kleine Anwendung mit WebCards erstellt. Als Erstes wird die erste Karte mit zwei Linien in vier gleichgroße Bereiche eingeteilt. Danach wird in jeden Bereich ein Textfeld, das den Namen eines der drei Team-Mitglieder enthält platziert. Diese Textfelder sollen möglichst dezent sein, deshalb wird über den Options-Dialog der Hintergrund und der Rand des ersten Textfeldes auf durchsichtig gestellt. Damit diese Einstellungen nicht für jedes der vier Textfelder einzeln durchgeführt werden müssen, wird das erste so veränderte Textfeld auf dem Lagermorph abgelegt, so können per Mausklick Kopien von diesem Morph erzeugt werden. Insgesamt werden vier Kopien erstellt. Eine für jedes Team-Mitglied und eine für noch nicht zugeteilte Aufgaben. Die vier Bereiche werden so beschriftet.

Da diese Aufteilung genau so jede Woche verwendet werden soll, wird diese Karte als Vorlage verwendet. Eine neue Karte, auf der die Zuteilung für die erste Woche vorgenommen werden soll, wird per Klick erzeugt. Über das Kontext-Menü wird der neuen Karte die erste Karte als Hintergrund zugewiesen.

Nun können die Aufgaben hinzugefügt werden: Für jede Aufgabe wird ein Textmorph mit einem hervorstechenden Rahmen verwendet. Auch hier wird der Lagermorph zum einfachen Duplizieren des speziell formatierten Morphs verwendet.

Alle drei Team-Mitglieder öffnen nun mit ihrem Browser den WebCards-Stapel und ziehen per Drag-and-Drop Aufgaben in ihren Bereich und erstellen neue Aufgaben.

5.2 Beispielanwendung: Adressbuch

Bei der zweiten mit WebCards realisierten Anwendung handelt es sich um ein Adressbuch. Die erste Karte wird als Vorlage für die Einträge im Adressbuch dienen. Auf der Karte werden vier Textfelder aus dem Lagermorph platziert. Über den Options-Dialog wird jedes der vier Textfelder beschriftet. Es gibt vier Felder, je eins für Namen, Telefonnummer, Raumnummer und E-Mail-Adresse. Eine erste Version der Vorlage für die Adressbuch-Einträge ist nun fertig.

Es wird eine neue Karte angelegt, der die eben erstellte Vorlage zugewiesen wird. Dann werden die Felder ausgefüllt. Ebenso wird mit zwei weiteren Karten verfahren. Das Adressbuch hat nun vier Karten. Davon dient eine als Vorlage und drei als Einträge.

Beim Anlegen der drei Adressbucheinträge hat sich gezeigt, dass das Eingeben des „@“-Zeichens im Webbrowser Chrome auf dem Betriebssystem Windows XP kompliziert ist. Um das Anlegen neuer Einträge im Adressbuch zu erleichtern wird in der Vorlage im Textfeld, das für die E-Mail-Adresse vorgesehen ist, ein „@“-Zeichens eingegeben. Da in den drei bereits vorhandenen Adressbucheinträgen die E-Mail-Adresse schon eingetragen ist, ändert sich an den vorhandenen Einträgen nichts. Beim Anlegen neuer Einträge steht nun im Feld für die E-Mail-Adresse nicht mehr das wenig hilfreiche Wort „Textfeld“, sondern ein „@“-Zeichen.

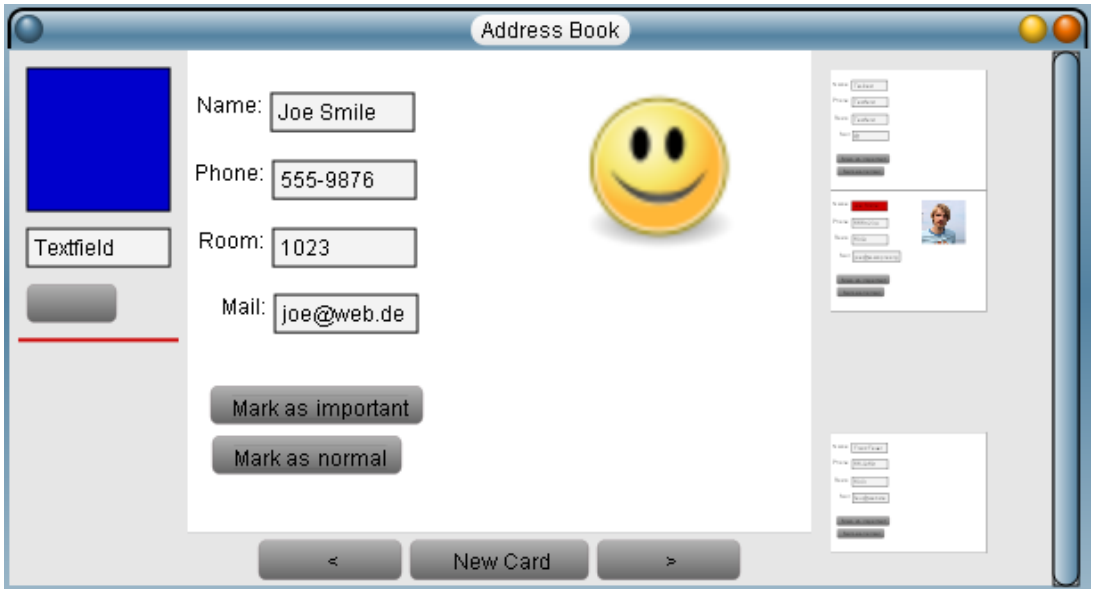


Abbildung 5.2: Screenshot der Beispielanwendung zwei

Um sich leichter an die im Adressbuch eingetragenen Personen erinnern zu können, soll zusätzlich noch ein Foto der betreffenden Person angezeigt werden. Dazu muss als Erstes die Platzierung der Textfelder geändert werden, da ansonsten nicht ausreichend Platz vorhanden ist. Um dies zu erreichen, wird auf der Vorlage per Drag-and-Drop, erst die Position des Textfeldes für die Raumnummer und anschließend das Textfeld für die E-Mail-Adresse verschoben. Die Änderung an der Vorlage wirkt sich sofort auf alle bereits existierenden Einträge im Adressbuch aus. Nun ist auf der rechten Hälfte aller Karten genug Platz für ein Foto.

Um externe Bilder in Lively Kernel einzubinden, gibt es den `ImageMorph`. Ihm muss die URL eines Bildes übergeben werden, das dann angezeigt wird. Die Fotos müssen also auf einen Webserver hochgeladen werden. Anschließend werden sie per `ImageMorph` auf den entsprechenden Karten platziert.

Abschließend wird das Adressbuch so erweitert, dass besonders wichtige Personen im Adressbuch visuell hervorgehoben werden können. Die Idee ist, dass bei wichtigen Personen das Namensfeld einen roten Hintergrund hat und bei anderen einen weißen. Um dies zu erreichen, wird auf der Vorlage-Karte ein Knopf aus dem Lagermorph platziert und mit Hilfe des Options-Dialogs angepasst. Als Beschriftung wird „Mark as important“ eingetragen. Als Funktion, die beim Betätigen des Knopfs ausgeführt werden soll, wird `this.owner.submorphs[0].setFill(Color.red)` eingetragen. Auf die gleiche Art wird ein Knopf angelegt, der das Namensfeld weiß färbt.

Die beiden so angelegten Knöpfe stehen sofort auf allen Karten zur Verfügung. Jetzt wird auf die zweite Karte gewechselt und per Klick auf ‘Mark as important“ das Namensfeld dieser Karte rot gefärbt.

5.3 Erkenntnisse aus den Beispielanwendungen

5.3.1 Hintergrund-Metapher

In der ersten Beispielanwendung wird das Hintergrundkonzept verwendet, um nicht für jede Wochenplanung die entsprechende Karte immer wieder aufs Neue aufteilen zu müssen. Dadurch wird auch ein einheitliches Aussehen aller Wochenplanungen erreicht.

Im zweiten Beispiel, dem Adressbuch, wird das Hintergrundkonzept hauptsächlich verwendet, um eine Vorlage für gleichartige Adressbucheinträge zu erzeugen. Des Weiteren wird das Hintergrundkonzept verwendet, um nachträglich Änderungen am Aussehen aller Adressbucheinträge vorzunehmen.

Die Hintergrund-Metapher erleichtert also das Erstellen von simplen Anwendungen, indem sie es ermöglicht, sowohl Vorlagen für Datensätze zu definieren als auch ein einheitliches Design zu realisieren. Dabei überlässt das Hintergrundkonzept dem Benutzer die Entscheidung, welche visuellen Eigenschaften der verwendeten Morphs als UI- und welche als Domain-Daten genutzt werden.

5.3.2 Kollaboration

In dem Adressbuch-Beispiel erstellt ein einzelner Anwender das Adressbuch. Das so erstellte Adressbuch könnte sowohl von einer Gruppe von Benutzer verwendet werden als auch nur von dem Ersteller allein. WebCards bietet jedoch keine Möglichkeit, festzulegen, wer einen Stapel benutzen darf. Falls der Ersteller des Adressbuch-Stapels beschließt, den Stapel gemeinsam mit Anderen nutzen zu wollen, so kann er dies ohne weiteren Aufwand tun.

Das Beispiel, in dem mit Hilfe eines WebCards-Stapel die Aufgabenverteilung in einem Team vorgenommen wird, zeigt, wie einfach es durch die in WebCards integrierten Mechanismen ist, eine Karte in WebCards synchron zu bearbeiten.

WebCards ermöglicht es, Webanwendungen einfach synchron erstellen und nutzen zu können.

5.3.3 Rückgängig machen

Falls im WebCards-Stapel, der zur Aufgabenverteilung genutzt wird, eine Aufgabe, die bereits zugeteilt war, aus Versehen in den Bereich eines anderen Team-Mitglieds

geschoben wird, muss nicht umständlich geklärt werden, wem die Aufgabe eigentlich zugeteilt war. Es wird einfach die Undo-Liste des Textfeldes, das die Aufgabe repräsentiert, geöffnet und die letzte Positionsänderung zurückgenommen.

Auch in dem Adressbuch-Stapel ist die Undo-Funktionalität nützlich. Falls die Telefonnummer einer bestimmten Person aktualisiert werden soll, unbeabsichtigterweise jedoch die Telefonnummer einer anderen Person geändert wird, so ist die überschriebene Telefonnummer nicht verloren sondern kann über die Undo-Liste wiederhergestellt werden.

Die Undo-Funktionalität von WebCards ermöglicht es den Benutzern eines Stapels unbeschwert Änderungen am Stapel vorzunehmen, ohne dabei Gefahr zu laufen, wichtige Daten zu verlieren.

5.3.4 Schlussfolgerung

Durch die beiden Beispielanwendungen sind ein paar Schwächen von WebCards erkennbar. WebCards konzentriert sich insbesondere darauf, das Erstellen von nützlichen Webanwendungen zu erleichtern, bei deren Erstellung kein Quelltext geschrieben werden muss. Deshalb erfordert das Schreiben der JavaScript-Funktionen für die beiden Knöpfe in der zweiten Beispielanwendung Wissen über JavaScript im Allgemeinen und Morphic und Lively Kernel im Speziellen. Des Weiteren ist es nicht möglich aus einer bestehenden Karte eine Vorlage einfach heraus zu extrahieren. Eine Karte kann nur komplett als Vorlage verwendet werden. Deshalb ist es oft sinnvoll, sich von Vornherein zu überlegen, welcher Teil eines Stapels durch Vorlagen realisiert werden soll.

Trotz einigem Verbesserungspotential, ermöglicht WebCards simple, aber nützliche Webanwendungen auf einfache Art zu erstellen, die synchron bearbeitet und benutzt werden können, und kommt so dem Ziel „Simple things should be simple“ näher.

Kapitel 6

Verwandte Arbeiten

In diesem Kapitel werden vier verwandte Arbeiten vorgestellt und verglichen. Die ersten zwei Systeme sind insbesondere aufgrund ihrer Ähnlichkeit zu HyperCard von Interesse. Die verbleibenden zwei werden in Hinblick auf die durch sie ermöglichte synchrone Kollaboration untersucht.

6.1 HyperCard ähnliche Systeme

6.1.1 TileStack

Anfang 2008 wurde auf der Macworld Messe TileStack vorgestellt[1]. Dabei wurde das Ziel verkündet, HyperCard wiederzubeleben.

Technische Grundlagen

TileStack[2] läuft komplett im Browser und braucht kein Plugin. Die Objekte eines TileStack-Staples werden durch native HTML-Elemente dargestellt. Auf Implementierungsebene werden diese Elemente durch Veränderung des Document Object Models (DOM) via JavaScript manipuliert.

Anwender von TileStack programmieren jedoch nicht in JavaScript, sondern in *Speak*[3]. Speak ist nahezu identisch mit HyperTalk (siehe 2.2.5). Damit die von Anwendern geschriebenen Speak-Skripte im Browser laufen, werden sie zu JavaScript übersetzt[4].

[1] http://www.tuaw.com/2008/01/18/show-floor-video-tilestack-aims-to-bring-hypercard-stacks-into/ (Abgerufen am 16.10.2009)
[2] `http://tilestack.com/` (Abgerufen am 16.10.2009)
[3] `http://tilestack.com/help/` (Abgerufen am 16.10.2009)
[4] `http://tilestack.com/faq/` (Abgerufen am 16.10.2009)

TileStack als Erweiterung von HyperCard

Wie gerade erwähnt entspricht die Skript-Sprache von TileStack der von HyperCard. Auch ansonsten ist TileStack sehr ähnlich zu HyperCard. Dies geht soweit, dass, zumindest theoretisch[5], alte HyperCard-Stapel in TileStack importiert werden können.

TileStack hat sämtliche Konzepte von HyperCard übernommen und lediglich kleine Ergänzungen durchgeführt. So stehen die Möglichkeiten, die der `XMLHttpRequest` in JavaScript bietet, auch in Speak zur Verfügung[6]. Des Weiteren gibt es neben den aus HyperCard bekannten Knöpfen und Textfeldern noch die aus HTML-Formularen bekannten Checkboxen, Radio-Buttons und Dropdown-Listen. TileStack verzichtet jedoch darauf, dem Anwender die Möglichkeit zum Zeichnen zu geben. Anwender müssen Bilder mit externen Werkzeugen erstellen und können diese anschließend in TileStack importieren.

Verwirrende Modi

Die Art, wie Elemente bearbeiten werden, ist in TileStack leicht verändert worden. Um einen Stapel zu bearbeiten, sei es um die Position eines Textfeldes oder das Skript eines Knopfes zu ändern, muss der Anwender in den Bearbeiten-Modus wechseln. In diesem Modus können Textfelder und Knöpfe, anders als in HyperCard, ohne spezielles Werkzeug per Drag-and-Drop verschoben werden. Wie auch HyperCard hat TileStack einen speziellen Modus zum Bearbeiten von Hintergründen.

Ob der normale Bearbeiten-Modus oder der zum Bearbeiten des Hintergrund aktiv ist, kann nur schwer unterschieden werden. Diese Tatsache wird noch dadurch verschärft, dass auch im normalen Bearbeiten-Modus Hintergrund-Textfelder und -Knöpfe bearbeitet werden können. Nur zum Hinzufügen von neuen Elementen zum Hintergrund muss der spezielle Modus verwendet werden.

Dies kann dazu führen, dass Anwender unbeabsichtigterweise die Position eines Elements, das im Hintergrund definiert wurde, auf allen Karten mit diesem Hintergrund ändern. Dies ist insbesondere wegen der fehlenden Undo-Funktionalität ärgerlich. Genauso leicht passiert es, dass ein neues Element unbeabsichtigterweise auf einer Karte und nicht auf dem Hintergrund definiert wird.

Jeder für sich allein

Auch in Bezug auf Kollaboration scheint sich TileStack an seinem Vorbild zu orientieren.

[5] Bei einem Test war das Ergebnis des Imports nicht funktionsfähig.
[6] `http://tilestack.com/help/topics/request/` (Abgerufen am 16.10.2009)

Jeder angemeldete Nutzer kann eigene Stapel erstellen. Wenn er es wünscht, kann er einen so erstellten Stapel veröffentlichen. Dann können auch andere Anwender den Stapel benutzen. Sie können den Stapel auch verändern. Solche Änderungen können jedoch nicht gespeichert werden. Ein Anwender, der einen fremden Stapel verändert hat, kann lediglich eine Kopie des veränderten Stapels zu seinen eigenen Stapeln hinzufügen. Wenn er diesen anschließend veröffentlicht, gibt es zwei Versionen des Stapels, wobei die Originalversion in jedem Fall erhalten bleibt.

Damit nicht für jede minimale Änderung eines Stapels eine Kopie von diesem angelegt werden muss, bietet TileStack die Möglichkeit, globale Variablen eines Stacks durch HTTP GET Parameter zu setzen. Dies nutzt beispielsweise ein Stapel, der das Cover des zuletzt gehörten Lieds eines Last.fm[7] Nutzers anzeigt. Der Last.fm-Benutzername kann dem Stack dabei in der URL übergeben werden. Der Aufruf von `http://tilestack.com/stacks/Latest_Last.fm_Track/?Guser=Musterfrau` setzt die globale Variable `user` auf `"Musterfrau"`. WebCards hingegen unterstützt das Übergeben von Variablen über die URL nicht.

Die eingeschränkten Kollaborations-Möglichkeiten sorgen dafür, dass TileStack sich gut zum Erstellen und Veröffentlichen von Geschicklichkeitsspielen sowie Widgets, die bestimmte Daten anzeigen, eignet. Auch persönliche Datensammlungen, wie sie mit HyperCard gerne erstellt wurden, lassen sich gut realisieren. Allerdings lassen sich mit TileStack keine kollaborativ genutzten Anwendungen erstellen. So kann ein Adressbuch nicht von mehreren Anwendern benutzt werden; jeder muss auf einer eigenen Kopie arbeiten.

WebCards hingegen legt den Schwerpunkt auf Anwendungen, die kollaborativ genutzt werden, und ist nur schlecht für Anwendungen geeignet, bei denen Benutzer unabhängig voneinander Objekte manipulieren sollen, wie es beispielsweise bei Geschicklichkeitsspielen der Fall ist.

HyperCard plus etwas Internet

TileStack lässt HyperCard tatsächlich wieder auferstehen. Dabei wird HyperCard um einige aktuelle Ideen, wie die Möglichkeit asynchrone HTTP-Anfragen zu stellen und Variablen über die URL zu übergeben, erweitert.

Gleichzeitig wird jedoch auch die Moduslastigkeit von HyperCard übernommen. Der Anwender muss sich merken in welchem Modus er ist und muss zwischen Bearbeiten und Ausführen umschalten. Dies ist wenig direkt und lebendig. Auch bei der Hintergrundmetahper bleibt TileStack HyperCard treu: Nur der Inhalt von Textfeldern kann instanzspezifisch sein.

[7] `http://www.last.fm/` (Abgerufen am 16.10.2009)

TileStack erweitert HyperCard um die eben genannten Ideen und bringt so HyperCard und das Web näher zusammen. Gleichzeitig bleibt TileStack aber in Bezug auf Kollaborations-Unterstützung hinter den Möglichkeiten.

6.1.2 BookMorph in Etoys

„Etoys ist ein Autorensystem für Kinder, das mit den OLPC XO Laptops ausgeliefert wird“ (Freudenberg u. a., 2009). Aufgrund der Konzentration auf Kinder als Zielgruppe wird Etoys als „Einstiegsstufen-Komponente“ von Squeak bezeichnet (Allen-Conn u. Rose, 2003).

Die Benutzer-Schnittstellen von Etoys und Lively Kernel haben eine starke Ähnlichkeit, da beide Systeme Morphic (siehe 2.1.2) als UI-Framework verwenden.

Etoys bietet einen `BookMorph`, der in der deutschen Übersetzung einfach als „Buch“ bezeichnet wird. Ein Buch entspricht weitestgehend einem Stapel in WebCards, die Seiten des Buches den Karten im Stapel. Auf einer Seite des Buches können, so wie in WebCards, Morphs abgelegt werden.

Für die Seitenwechsel können per Menü, ähnlich wie in HyperCard, einfach optische Effekte und Klangeffekte festgelegt werden.

Suchfunktion

Werden Textfelder auf den Seiten des Buches abgelegt, so können diese über das Menü des `BookMorph` durchsucht werden. Das Durchsuchen ist dabei dem aus HyperCard sehr ähnlich (siehe 2.2.4 und 2.2.6). Das heißt, bei Suchen kann, wie bei der Suchfunktion eines Texteditors, lediglich ein Wort eingegeben werden. Es ist nicht möglich beispielsweise in einem Telefonbuch nur die Textfelder, die den Nachnamen enthalten, zu durchsuchen.

Vorlagen-Metapher

Der `BookMorph` bietet die Möglichkeit, eine Seite als Vorlage zu verwenden. Dazu muss der Benutzer sich auf der als Vorlage zu verwendenen Seite befinden und im Menü unter „Erweitert...“ „Mache dies zur Vorlage für alle neuen Seite“ auswählen. Daraufhin wird eine Kopie der aktuellen Karte als Vorlage für neue Karten abgelegt. Wenn der Benutzer nun eine neue Karte anfordert, wird anstelle einer leeren Karte eine Kopie der Vorlage erzeugt. Auf diese Weise können einfach Telefonbücher oder ähnliches erstellt werden[8].

[8] John Hinsley, A Morphic Rolodex Tutorial using Direct Manipulation (and Scripting!), `http://wiki.squeak.org/squeak/2102` (Abgerufen am 12.10.2009), 2007

Im Gegensatz zu WebCards geht jedoch die Verbindung zwischen der als Vorlage ausgewählten Karte/Seite und den mit dieser Vorlage erzeugten Karten/Seiten verloren. So ist es bei Verwendung der Vorlage für gleichartige Datensätze nicht möglich, nachträglich Änderungen an der Vorlage durchzuführen. Änderungen, egal ob sie das Design oder die Vorlage für gleichartige Datensätze betreffen, müssen auf jeder Karte von Hand ausgeführt werden. Der Vorlagen-Mechanismus von WebCards ist somit wesentlich mächtiger.

Undo

Im Gegensatz zu WebCards bietet der `BookMorph` keine eigene Unterstützung für das Zurücknehmen von Aktionen. Etoys bietet immerhin eine simple Undo-Funktion an, mit deren Hilfe die letzte Aktion zurückgenommen werden kann. Diese Undo-Funktionalität kann somit auch beim Arbeiten mit dem `BookMorph` genutzt werden.

Kollaboration

Auch für Kollaboration bietet der `BookMorph` keine eigene Unterstützung. Es kann jedoch Kollaborations-Unterstützung von Etoys verwendet werden. So kann zur asynchronen Kollaboration ein Projekt sehr einfach auf einen speziellen Server geladen werden oder ein einzelner Morph in einer Datei gespeichert werden. Für synchrone Kollaboration könnte Nebraska[9] verwendet werden. Nebraska erzeugt auf einem Server eine Welt, die alle Teilnehmer sich teilen. Die Maus- und Tastatur-Eingaben eines Benutzers werden an den Server weitergeleitet. Der Server sendet Befehle an alle Teilnehmer, die den Inhalt der Welt zeichnen.

HyperCard-ähnlich

Der `BookMorph` in Etoys stellt eine gelungene Fortführung von HyperCard da. Der Schwerpunkt liegt dabei insbesondere auf der Einfachheit. Bei der Umsetzung der Hintergrund-Metapher geht dies zulasten der Mächtigkeit.

Ebenso wie ein HyperCard-Stapel bleibt auch ein Buch, das auf Grundlage des `BookMorph` erstellt wurde, hinter den heutigen Möglichkeiten zurück (siehe 2.2.6), da das Buch kaum Gebrauch von den Fähigkeiten des Web macht.

[9] Nebraska, 2007, `http://wiki.squeak.org/squeak/1356` (Abgerufen am 12.10.2009)

6.2 Systeme zur synchronen Kollaboration

6.2.1 TinLizzie WysiWiki

Das TinLizzie WysiWiki System (Ohshima u. a., 2007) versucht die Nachteile heutiger Wiki-Systeme zu überwinden. Als Nachteil wird dabei gesehen, dass Wikis sich fast ausschließlich auf Texte und Bilder konzentrieren, wohingegen interaktive und multimediale Inhalte kaum Beachtung finden. Des Weiteren bemängeln Ohshima u. a., dass das Bearbeiten von Wiki-Artikeln nicht direkt, sondern indirekt, zumeist unter Verwendungen einer speziellen Syntax, erfolgt. Zusätzlich wird kritisiert, dass Wiki-Artikel nur asynchron bearbeitet werden können.

Technische Grundlagen

Um diese Nachteile zu überwinden, wird als Grundlage Tweak[10], das auf Squeak aufsetzt, verwendet. Auf Tweak aufbauend haben Ohshima u. a. eine Etoys-ähnliche Benutzerschnittstelle geschaffen.

Damit dieses System im Browser zur Verfügung steht, muss der Anwender die Squeak VM inklusive Browser-Plugin installiert haben. Lively Kernel hingegen hat es sich als Ziel gesetzt, wie eine gewöhnliche HTML-Webseite, ohne extra Plugins auszukommen. Der Verzicht auf Plugins hat den Vorteil, dass die Hemmschwelle Lively Kernel zu verwenden gesenkt wird.

Das Äquivalent zu einem Wiki-Artikel ist in TinLizzie WysiWiki ein Dokument. Ein Dokument ist ein Tweak-Projekt, das im Open Document Format (ODF) abgespeichert ist. Ähnlich einer Karte in WebCards hat ein Dokument eine feste Breite und eine feste Höhe. Deshalb wurde nicht das Open Document Format für Textdateien, sondern das für Präsentationen verwendet.

Asynchrone Kollaboration

Das asynchrone Bearbeiten eines TinLizzie WysiWiki-Dokuments läuft ähnlich wie das klassische Bearbeiten eines Wiki-Artikels ab. Als Erstes wird das Dokument vom Server herunter geladen. Dann wird aus der Datei das original Tweak-Projekt wiederhergestellt. Anders als bei einem echten Wiki-Artikel, muss anschließend kein Bearbeiten-Knopf betätigt werden, der Anwender kann direkt das Projekt verändern. Wenn der Anwender möchte, kann er sich nach dem Bearbeiten dazu entscheiden, dass er die Veränderungen abspeichern möchte. In diesem Fall wird das veränderte Projekt wieder in eine ODF-Datei umgewandelt. Abschließend wird das Dokument via WebDAV, einer Erweiterung von HTTP, an den Server gesendet.

[10] `http://tweakproject.org/` (Abgerufen am 15.10.2009)

Klassischen Wiki-Syteme haben eine strenge Unterscheidung zwischen Betrachten und Bearbeiten eines Artikels. Beim Abrufen eines Artikels befindet sich der Benutzer im Betrachten-Modus und kann nichts verändern. Will er den Artikel verändern, so muss er in den Bearbeiten-Modus wechseln. TinLizzie WysiWiki hingegen verzichtet aus den gleichen Gründen wie Morphic zunächst auf eine Unterscheidung zwischen Betrachten und Bearbeiten eines Artikels. Wenn ein Anwender ein Dokument verändern möchte, dann benutzt er es auf die gleiche Art, wie ein Anwender, der das Dokument nicht verändern will, mit dem entscheidenden Unterschied, dass er zum Schluss das durch die Benutzung veränderte Dokument abspeichert. Seine Veränderungen werden so dauerhaft. Die Entscheidung, ob ein Anwender ein Dokument bearbeitet oder benutzt, wird also nicht im Vorhinein, sondern im Nachhinein getroffen.

Asynchrone Kollaboration ist in Wiki-Systemen vor allem dann zweckmäßig, wenn man annimmt, dass die meisten Benutzer eines Wikis den Inhalt eines Artikels lediglich lesen wollen. Diese Annahme triff bei Enzyklopädien voll zu. Allerdings werden Wikis auch immer mehr für andere Zwecke verwendet, beispielsweise um in Teams Termine oder anderes zu klären. Bei einer solchen Verwendung steigt die Rate von Schreibzugriffen erheblich an und somit auch die Wahrscheinlichkeit, dass gleichzeitige zwei Änderungen erfolgen, die dann in Konflikt stehen.

Synchrone Kollaboration

Ein Bearbeiter eines Dokuments kann sich jederzeit entscheiden, dass er mit bestimmten anderen Anwendern synchron kollaborieren möchte. In diesem Fall muss er diese Anwender dazu einladen.

WebCards hingegen kennt das explizite Einladen nicht. Alle Benutzer eines Stapels kollaborieren automatisch synchron. So wird vermeiden, dass es zu Konflikten kommt.

Ein weiterer in diesem Zusammenhang relevanter Unterschied zwischen WebCards und TinLizzie WysiWiki ist, dass bei TinLizzie WysiWiki Dokumente voneinander unabhängig sind. So können gleichzeitig mehrere Benutzer unterschiedliche Dokumente bearbeiten. Es ist also mit dem Mittel der asynchronen Kollaboration ein synchrones Arbeiten möglich. In WebCards hingegen sind die Karten eines Stapels, auch aufgrund der Hintergrund-Metapher, nicht unabhängig voneinander. Trotzdem können mehrere Benutzer gleichzeitig unterschiedliche Karten des gleichen Stapels bearbeiten, ohne voneinander wissen zu müssen. Dies ist möglich, da sie, ohne sich gegenseitig einladen zu müssen, automatisch synchron kollaborieren.

Im Gegensatz zu WebCards sind in TinLizzie WysiWiki asynchrone und synchrone Kollaboration komplett unterschiedlich realisiert. Wenn ein Anwender zur synchronen Kollaborieren eingeladen wurde, erhält er die aktuelle Version des Dokuments als ODF-Datei übertragen. So haben alle Teilnehmer die gleiche Ausgangsbasis. Von da an wird eine vereinfachte Form des TeaTime Protokolls aus Croquet (Smith u. a., 2003) verwendet. Bei diesem Protokoll handelt es sich um ein pessimistisches Peer-to-

Peer-Protokol. Die Verwendung eines Peer-to-Peer-Protokols ist möglich, da TinLizzie WysiWiki nicht die vom Browser gemachten Beschränkungen einhält und beispielsweise via `XMLHttpRequest` kommuniziert, sondern die Möglichkeiten der Squeak VM nutzt.

Lebendiges Wiki

TinLizzie WysiWiki erlaubt, ähnlich wie Etoys und Lively Kernel, das Erstellen interaktiver und reichhaltiger Inhalte. Zusätzlich führt das System einen Modus zur expliziten synchronen Kollaboration ein. TinLizzie WysiWiki stellt eine Weiterentwicklung klassischer Wikis dar. Dabei eignet sich TinLizzie WysiWiki besonders für Inhaltssammlungen wie Enzyklopädien, auf die zumeist lesend zugegriffen wird.

6.2.2 Jupiter Kollaborationssystem

Das Jupiter Kollaborationssystem (Nichols u. a., 1995) stellt eine virtuelle Welt zur Verfügung, die synchrone Kollaboration ermöglicht.

In Bezug auf die vorliegende Arbeit ist insbesondere der in Jupiter verwendete Algorithmus zur synchronen Kollaboration interessant und wird deshalb im Folgenden betrachtet. Der in WebCards verwendete Algorithmus kann als eine Vereinfachung dieses Algorithmus angesehen werden. Allerdings führt diese Vereinfachung, wie im Folgenden zu sehen ist, auch zu einer Verringerung der Mächtigkeit des Algorithmus.

Architektur von Jupiter

Wie auch WebCards, verwendet Jupiter einen zentralen Koordinator. Im Gegensatz zu WebCards handelt es sich dabei aber nicht um eine Datenbank, die für vielfältige Zwecke verwendet werden kann, sondern um einen eigens für Jupiter entwickelten Server. Dieser Server beheimatet die von Jupiter zur Verfügung gestellte virtuelle Welt einschließlich des Zustands und des Programmcodes. Jeder Teilnehmer der synchronen Kollaboration hat eine TCP-Verbindung zu dem Server. Der Server kann als aktiver Teilnehmer der durch ihn ermöglichten synchronen Kollaboration angesehen werden, da er den für die virtuelle Welt zuständigen Programmcode selber ausführt und Nachrichten an die mit ihm verbundenen Teilnehmer sendet.

WebCards und Jupiter behandeln das Problem der hohen Latenz und beschränkten Bandbreite auf ähnliche Art und Weise. In beiden Systemen kommunizieren Client und Server auf einer hohen Ebene miteinander. So versendet Jupiter Nachrichten über Änderungen am Zustand von speziell für Jupiter entwickelten Widgets. Im Vergleich zum Versenden von Benutzeraktionen, wie Tastatur- oder Mauseingaben und dem Übertragen von Grafik, muss so seltener und weniger kommuniziert werden.

Ebenso wie WebCards, verwendet Jupiter einen optimistischen Algorithmus. Das heißt, Änderungen eines Nutzers werden auf seiner Instanz des Systems sofort ausgeführt. Dadurch, dass Änderungen sofort, das heißt, ohne zuvor den Server zu kontaktieren, durchgeführt werden, kann der Benutzer die Software wesentlich flüssiger bedienen. Allerdings kann es dazu kommen, dass die optimistischerweise sofort ausgeführten Änderungen mit anderen Änderungen in Konflikt stehen.

Konsistenzerhaltung

Bei der Behandlung von Konflikten unterscheidet sich, wie im Folgenden ersichtlich wird, WebCards von Jupiter. Nichols u. a. verwenden zum Lösen von Konflikten eine Abwandlung der von Ellis u. Gibbs (1989) vorgeschlagenen Operations-Transformation. Dabei bestimmt der Server für zwei gleichzeitig durchgeführte Änderungen, welche Operationen die beiden Initiatoren der Änderungen durchführen müssen um zu einem gemeinsamen Ergebnis zu kommen. Ein Beispiel von Nichols u. a. (1995, S. 5) soll dies verdeutlichen: Angenommen der Text „ABCDE“ wird gleichzeitig bearbeitet. Teilnehmer eins führt die Aktion „lösche Buchstaben vier“ durch, sein Ergebnis ist „ABCE“. Gleichzeitig führt Teilnehmer zwei die Aktion „lösche Buchstaben zwei“ durch, sein Ergebnis ist „ACDE“. Teilnehmer eins muss dann die Aktion „lösche Buchstaben zwei“ durchführen und Teilnehmer zwei die Aktion „lösche Buchstaben drei“. Das gemeinsame Ergebnis ist dann „ACE“.

Insgesamt sind in Jupiter 456 Kombinationen von gleichzeitigen Operationen denkbar. Bei den allermeisten dieser Kombinationen liegt jedoch kein Konflikt vor, da die Operationen sich nicht beeinflussen. Nach Abzug dieser Kombinationen bleiben 65 übrig (Nichols u. a., 1995, S. 8).

Doch auch bei diesen Kombinationen sind noch viele triviale Fälle vorhanden, bei denen beide Operationen das gleiche Attribut eines Widgets setzen. In diesem Fall wird eine der beiden Operationen als erfolgreiche festgelegt. Der Produzent der nicht erfolgreichen Operation muss die erfolgreiche Operationen durchführen. Die erfolglose Operation wird einfach verworfen.

Ein Großteil der dann noch verbleibenden Kombinationen bezieht sich auf das gleichzeitige Bearbeiten eines Textes. Abgesehen von diesen Operationen erzielt der auf der Ausführungsreihenfolge von Kommando-Objekten basierende Algorithmus von WebCards das gleiche Resultat. Dabei hat er den Vorteil, dass er allgemeingültig ist und nicht für jede Kombinationsmöglichkeit von Operationen eine spezielle Operations-Transformation festgelegt werden muss. Zusätzlich hält der in WebCards verwendete Algorithmus nicht nur den momentanen Zustand der Morphs konsistent, sondern auch die gesamte Reihenfolge der Kommando-Objekte und somit die Geschichte des Morphs, die für Undo und andere Funktionalität genutzt wird.

Rein hypothetisch würde der Algorithmus zur Konsistenzerhaltung in WebCards auch für das gleichzeitige Bearbeiten eines Textes funktionieren, wenn es sich bei dem Text

nicht um den Zustand eines Morphs handeln würde, sondern jeder Buchstabe selbst ein Morph wäre.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Mit WebCards wurde im Rahmen dieser Masterarbeit ein auf Lively Kernel basierendes Programm entwickelt, das es Endanwendern ermöglicht, datenzentrierte, graphische Webanwendungen auf einfache Art und Weise zu erstellen und somit dem Ziel „Simple things should be simple“ näher kommt.

Als ein Ideengeber zur Erreichung dieses Ziels diente HyperCard, dessen Vor- und Nachteile im Rahmen dieser Arbeit aufgezeigt wurden. Dabei wurden von HyperCard die Metaphern, die leicht verständlich, gleichzeitig aber auch mächtig genug sind, übernommen, sodass auch in WebCards eine Anwendung ein Stapel von Karten mit Hintergründen ist. Darüber hinaus wurde der Ansatz von HyperCard, Anwendern das Erstellen von simplen, nützlichen Programmen ohne Verwendung einer Programmiersprache zu ermöglichen, umgesetzt.

Die Möglichkeit, in HyperCard ohne Verwendung einer Programmiersprache erste nützliche Anwendungen erstellen zu können, basiert hauptsächlich auf der Hintergrund-Metapher. Deshalb wurde für WebCards eine Hintergrund-Metapher entwickelt, die der Möglichkeit des *live editing* Rechnung trägt und gleichzeitig so ausgestaltet ist, dass sie beiden Verwendungszwecken der Hintergrund-Metapher aus HyperCard gerecht wird, welche einerseits das Erzielen eines einheitlichen Aussehens und andererseits das Anlegen von Vorlagen für Datensätze sind. Die so entwickelte Hintergrund-Metapher geht weit über die Möglichkeiten der Hintergrund-Metapher aus HyperCard hinaus und ermöglicht so eine kreative Nutzung.

Im Gegensatz zu HyperCard ist WebCards explizit auf die Unterstützung von Zusammenarbeit ausgerichtet. Dazu ermöglicht WebCards synchrone Kollaboration und eröffnet so ein breiteres Feld von Anwendungsmöglichkeiten und vermeidet überdies die bei asynchroner Kollaboration auftretenden Konflikte. Die synchrone Kollabora-

tion wird durch einen zentralen Server ermöglicht, da sich dies bei im Webbrowser laufenden Programmen anbietet und die Implementierung erleichtert. Weiterhin wird in WebCards zur synchronen Kollaboration ein optimistischer Algorithmus verwendet, der die erzeugten Kommando-Objekte asynchron versendet. Dadurch hat WebCards eine schnelle Antwortzeit und der Benutzer kann flüssig arbeiten.

Auch die Undo-Funktionalität von WebCards erleichtert dem Benutzer das Arbeiten mit WebCards: Sowohl beim Erstellen von Programmen als auch beim Benutzen kann er sich darauf verlassen, dass Änderungen an Morphs einfach rückgängig zu machen sind. Da die Kommando-Objekte, die für die Undo-Funktionalität verwendet werden, persistent sind, stellt die Undo-Funktionalität eine feingranulare Alternative zur Versionsgeschichte, wie sie in klassischen Wikis verwendet wird, dar. Des Weiteren kann durch die Undo-Liste eines Morphs leicht nachvollzogen werden, wie das Aussehen des betreffenden Morphs zustande gekommen ist.

Im Rahmen der Entwicklung von WebCards wurde Lively Kernel um synchrone Kollaboration, einen Undo- sowie einen Vorlagen-Mechanismus erweitert. Als Grundlage dafür wurde Lively Kernel um Unterstützung für Kommando-Objekte erweitert, die so ausgestaltet ist, dass Kommando-Objekte unabhängig davon erzeugt werden, ob Veränderungen an Morphs über die graphische Benutzerschnittstelle oder durch Skripte durchgeführt werden. Die so erzeugten Kommando-Objekte werden gleichermaßen zur Realisierung von Persistenz, Undo, Kollaboration sowie einer Vorlagen-Metapher verwendet. Die dafür verwendeten Algorithmen wurden erläutert sowie ihre individuellen Vor- und Nachteile diskutiert.

Überdies wurden die Vor- und Nachteile, die sich aus der gleichzeitigen Verwendung derselben Kommando-Objekte für all diese Anforderungen ergeben, erläutert. Zu den Nachteilen gehört vor allem die starke Kopplung der Kommando-Objekte mit all diesen Funktionalitäten, die insbesondere dazu führt, dass nicht beliebige neue Kommando-Klassen eingeführt werden können, sondern dass immer die Belange aller Funktionalitäten berücksichtigt werden müssen. Dazu gehört beispielsweise, dass Kommando-Objekte immer nur genau einen Morph verändern dürfen. Ebenso muss die Aktion eines Kommando-Objektes komplett zurücknehmbar sein, da von der Undo-Methode der Kommando-Objekte in den vorgestellten Algorithmen reger Gebrauch gemacht wird und sich selbst kleine Abweichungen vom Ursprungszustand sonst anhäufen würden.

Diesen Nachteilen stehen starke Vorteile gegenüber. Auf Seiten des Anwenders besteht der erhebliche Vorteil, dass die Anwendung ein einheitliches und somit auch leichter nachvollziehbares Verhalten aufweist. Auch für die Entwicklung von WebCards ist die Verwendung von Kommando-Objekten für die genannten Funktionalitäten von Vorteil, da Algorithmen verwendet werden können, die unabhängig davon sind, für wie viele Arten von Aktionen Kommando-Objekte erzeugt werden und somit sehr leicht Erweiterungen um neue Arten von Aktionen vorgenommen werden können.

7.2 Ausblick

Über eine potentielle Erweiterung um neue Arten von Aktionen hinaus, sind die im Folgenden genannten möglichen Erweiterungen von WebCards um zusätzliche Funktionalitäten interessant. Zu diesen, für Anwender nützlichen, Ergänzungen gehört sicherlich eine Suchfunktion, die es ermöglicht, die Textfelder aller Karten, die eine bestimmte Vorlage haben, nach einem Wort zu durchsuchen. Dabei wäre es sinnvoll, wenn die Suche sich auf einzelne Textfelder beschränken ließe, sodass beispielsweise in einem Adressbuch bei der Suche nach einem Namen verhindert werden kann, dass das Adressfeld ebenfalls durchsucht wird und unerwünschte Treffer liefert.

Basierend darauf, dass nicht nur der aktuelle Zustand, sondern durch die Kommando-Objekte die Geschichte eines Morphs vorhanden ist, könnte dem Benutzer auch die Möglichkeit gegeben werden, bei einer Suche die ehemaligen Werte der Textfelder mit durchsuchen zu lassen. Es müsste untersucht werden, ob es sich dabei um eine hilfreiche Funktionalität handelt oder ob sie den Benutzer verwirrt.

Eine weitere Ergänzung von WebCards könnte eine Übersicht-Detail-Komponente sein, mit deren Hilfe der Inhalt aller Karten, die eine bestimmte Vorlage haben, tabellarisch angezeigt wird. Per Klick auf eine Tabellen-Zeile würde auf die entsprechende Karte gesprungen. Eine solche Komponente kann von Anwendern benutzt werden, um eine schnelle Übersicht über die vorhandenen Einträge, die durch Karten repräsentiert sind, zu gewinnen. Des Weiteren könnte sie auch verwendet werden, um die Suchergebnisse der eben vorgeschlagenen Suchfunktion darzustellen.

Auch die Undo-Liste kann, wie in 4.2.3 erläutert, durch zukünftige Arbeit übersichtlicher gestaltet werden. Darüber hinaus könnte der Nutzen der Undo-Funktionalität erhöht werden, indem nach Möglichkeiten gesucht wird, wie das Löschen von Morphs intuitiver und direkter zurückgenommen werden kann.

Aufgrund der Beschränkung des Umfangs dieser Masterarbeit wurde für WebCards noch kein Funktionalität entwickelt, die einen Überblick darüber verschafft, wer gerade ebenfalls einen bestimmten Stapel bzw. eine bestimmte Karte bearbeitet. Eine solche Funktionalität ist aber, wie in 3.3.3 beschrieben, für den Benutzer sehr wichtig und soll in einer kommenden Version integriert werden.

Um Benutzer dabei zu unterstützen, bereits bekannte Stapel wiederzufinden und neue Stapel zu entdecken, könnten Erweiterungen von WebCards in Betracht gezogen werden. Eine Möglichkeit ist es, dem Benutzer eine Übersicht mit Namen und Vorschaubild über alle auf einem Server vorhandenen Stapel anzuzeigen.

Neben der Erweiterung um komplett neue Funktionalitäten ist es erforderlich auch bereits bestehende Teile des Programms weiter zu entwickeln. Dazu gehört besonders eine Erweiterung der bestehenden Kommando-Objekt-Architektur um eine Löschfunktionalität, da in der jetzigen Version sämtliche Kommando-Objekte immer gespeichert werden. Die Bedingungen, die beim Löschen der Kommando-Objekte eingehalten wer-

den müssen, wurden in 4.2.7 dargestellt. Bei der Realisierung einer, möglicherweise automatischen, Löschfunktionalität muss geklärt werden, wie lange der Zugriff auf die komplette oder teilweise Geschichte eines Objektes möglich sein soll, da diese nach dem Löschen der betreffenden Kommando-Objekte nicht mehr zur Verfügung steht.

Im Rahmen dieser Arbeit wurde insbesondere die Möglichkeit, Webanwendungen ohne Verwendung einer Programmiersprache zu erstellen, untersucht. In kommenden Arbeiten könnte deshalb untersucht werden, wie Endanwender das Erstellen von Skripten erleichtert werden kann und ob dafür eine spezielle Programmiersprache, wie HyperTalk, benötigt wird.

Des Weiteren könnten folgende Arbeiten sich mit der Fragestellung auseinandersetzen, worin genau die Vorteile asynchroner Kollaboration im Vergleich zu synchroner Kollaboration bestehen und wie man diese in WebCards integrieren könnte.

Auf Basis der Kommando-Objekte wurden in WebCards bereits einige Funktionalitäten realisiert, es erscheint spannend noch weitere Funktionalitäten, die sich auf dieser Basis leicht umsetzen lassen, zu entdecken. So könnte beispielsweise eine Zeitleiste entwickeltet werden, auf der ein Benutzer wie in einem Multimedia-Player, einen Schieberegler verschiebt und dadurch zu unterschiedlichen Zeitpunkten in der Geschichte einer Karte gelangt. Dies könnte gut unter Verwendung der Undo-Methoden der Kommando-Objekte implementiert werden. Eine solche Zeitleiste ermöglicht es Benutzern, unkompliziert die Entstehung einer Karte nachzuvollziehen.

Auch ohne diese Erweiterungen ist WebCards ein interessantes und spannendes Programm, das es Endanwendern erlaubt kollaborativ datenzentrierte, graphische Endanwender-Webanwendungen einfach zu erstellen.

Literaturverzeichnis

[Allen-Conn u. Rose 2003] Allen-Conn, Betty J. ; Rose, Kim: *Powerful Ideas in the Classroom – Using Squeak to Enhance Math and Science Learning.* Glendale, Kalifornien : Viewpoints Research Institute, Inc., 2003

[Anderson u. a. 2009] Anderson, Chris ; Lehnardt, Jan ; Slater, Noah: *CouchDB: The Definitive Guide.* Sebastopol, Kalifornien : O'Reilly Media, Inc., 2009

[Becker u. Dörfler 1991] Becker, Karl-Heinz ; Dörfler, Michael: *Wege zu Hypercard: Der Einstieg in eine neue Software-Generation: Für Version 2.* Braunschweig : Vieweg, 1991

[Cass u. a. 2006] Cass, Aaron G. ; Fernandes, Chris S. T. ; Polidore, Andrew: An empirical evaluation of undo mechanisms. In: Mørch, Anders I. (Hrsg.) ; Morgan, Konrad (Hrsg.) ; Bratteteig, Tone (Hrsg.) ; Ghosh, Gautam (Hrsg.) ; Svanaes, Dag (Hrsg.): *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction.* New York : ACM, 2006, S. 19–27

[Coulouris u. Thimbleby 1993] Coulouris, George ; Thimbleby, Harold: *HyperProgramming: Building Interactive Programs with HyperCard.* Boston, Massachusetts : Addison-Wesley Longman Publishing Co., Inc., 1993

[Dean u. Ghemawat 2004] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation.* Berkeley, Kalifornien : USENIX Association, Dezember 2004, 137-150

[Ebersbach u. Glaser 2005] Ebersbach, Anja ; Glaser, Markus: Wiki. In: *Informatik Spektrum* 28 (2005), Nr. 2, S. 131–135

[ECMA International 1999] ECMA International: *ECMA-262: ECMAScript Language Specification.* `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf`. Version: Dezember 1999

[Ellis u. Gibbs 1989] Ellis, Clarence A. ; Gibbs, Simon J.: Concurrency Control in Groupware Systems. In: Clifford, James (Hrsg.) ; Lindsay, Bruce G. (Hrsg.) ; Maier, David (Hrsg.): *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data.* New York : ACM, 1989, S. 399–407

[Ellis u. a. 1991] Ellis, Clarence A. ; Gibbs, Simon J. ; Rein, Gail: Groupware: Some Issues and Experiences. In: *Communications of the ACM* 34 (1991), Nr. 1, S. 39–58

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures.* Irvine, Kalifornien, University of California, Doktorarbeit, 2000

[Freudenberg u. a. 2009] Freudenberg, Bert ; Ohshima, Yoshiki ; Wallace, Scott: Etoys for One Laptop Per Child. In: *Proceedings of the Seventh Annual International Conference on Creating, Connecting and Collaborating through Computing*, 2009. – Noch nicht erschienen

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston, Massachusetts : Addison-Wesley, 1995

[Günther 1990] Günther, Carsten: Zweiter Meilenstein – Das neue Macintosh-HyperCard 2.0. In: *c't* 10 (1990), S. 194

[Goodman 1998a] Goodman, Danny: *The Complete HyperCard 2.2 Handbook - Volume 1.* New York : toExcel, 1998

[Goodman 1998b] Goodman, Danny: *The Complete HyperCard 2.2 Handbook - Volume 2.* New York : toExcel, 1998

[Greenberg u. a. 1996] Greenberg, Saul ; Gutwin, Carl ; Cockburn, Andy: Using Disortion-Oriented Displays to Support Workspace Awareness. In: Sasse, Martina A. (Hrsg.) ; Cunningham, Jim (Hrsg.) ; Winder, Russel L. (Hrsg.): *BCS HCI: People and Computers XI, Proceedings of HCI '96*, Springer, 1996, S. 299–314

[Hirschfeld u. a. 2008] Hirschfeld, Robert ; Costanza, Pascal ; Nierstrasz, Oscar: Context-oriented Programming. In: *Journal of Object Technology* 7 (2008), März - April, Nr. 3, S. 125–151

[Ingalls u. a. 1997] Ingalls, Dan ; Kaehler, Ted ; Maloney, John ; Wallace, Scott ; Kay, Alan C.: Back to the future: the story of Squeak, a practical Smalltalk written in itself. In: *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.* New York : ACM, 1997, S. 318–326

[Ingalls u. a. 2008] Ingalls, Dan ; Palacz, Krzysztof ; Uhler, Stephen ; Taivalsaari, Antero ; Mikkonen, Tommi: The Lively Kernel A Self-supporting System on a Web Page. In: Hirschfeld, Robert (Hrsg.) ; Rose, Kim (Hrsg.): *S3: Self-Sustaining*

Systems, First Workshop, Revised Selected Papers Bd. 5146. Berlin, Heidelberg : Springer-Verlag, Mai 2008 (Lecture Notes in Computer Science), S. 31–50

[Krahn u. a. 2009] Krahn, Robert ; Ingalls, Dan ; Hirschfeld, Robert ; Lincke, Jens ; Palacz, Krzysztof: Lively Wiki - A Development Environment for Creating and Sharing Active Web Content. In: *Proceedings of the International Symposium on Wikis and Open Collaboration (WikiSym)*. Orlando, Florida, 2009. – Noch nicht erschienen

[Kung u. Robinson 1981] Kung, H. T. ; Robinson, John T.: On Optimistic Methods for Concurrency Control. In: *ACM Transactions on Database Systems* 6 (1981), Nr. 2, S. 213–226

[Lincke u. a. 2009] Lincke, Jens ; Krahn, Robert ; Ingalls, Dan ; Hirschfeld, Robert: Lively Fabrik - A Web-based End-user Programming Environment. In: *In Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C5)*. Los Alamitos, Kalifornien, Januar 2009. – Noch nicht erschienen

[Maier u. Dwornitzak 2007] Maier, Andreas ; Dwornitzak, Denis: *Analyse des Sun Labs Lively Kernel*. `http://www.technetbloggers.de/downloads/LivelyKernel_Document.pdf`. Version: Dezember 2007. – nicht veröffentlicht

[Maloney 1995] Maloney, John H.: Morphic: The Self User Interface Framework. Self 4.0 Release Documentation / Sun Microsystems Laboratories. 1995. – Forschungsbericht

[Maloney u. Smith 1995] Maloney, John H. ; Smith, Randall B.: Directness and Liveness in the Morphic User Interface Construction Environment. In: *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*. New York : ACM, November 1995, S. 21–28

[Mikkonen u. Taivalsaari 2007] Mikkonen, Tommi ; Taivalsaari, Antero: Using JavaScript as a Real Programming Language / Sun Microsystems Laboratories. 2007 (TR-2007-168). – Forschungsbericht

[Molina u. a. 2007] Molina, Francis ; Sweeney, Brian ; Willard, Ted ; Winter, André: Building cross-browser interfaces for digital libraries with scalable vector graphics (SVG). In: Rasmussen, Edie M. (Hrsg.) ; Larson, Ray R. (Hrsg.) ; Toms, Elaine (Hrsg.) ; Sugimoto, Shigeo (Hrsg.): *JCDL '07: Proceedings of the 7th ACM/IEEE Computer Society joint conference on Digital libraries*. New York : ACM, 2007, S. 494

[Nichols u. a. 1995] Nichols, David A. ; Curtis, Pavel ; Dixon, Michael ; Lamping, John: High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System. In: *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*. New York : ACM, 1995, S. 111–120

[Ohshima u. a. 2007] Ohshima, Yoshiki ; Yamamiya, Takashi ; Wallace, Scott ; Raab, Andreas: TinLizzieWysiWiki and WikiPhone: Alternative approaches to

asynchronous and synchronous collaboration on the Web. In: *Fifth International Conference on Creating, Connecting and Collaborating through Computing (C^5 2007)*. Los Alamitos, Kalifornien : IEEE Computer Society, 2007, S. 36–46

[Rode u. a. 2006] Rode, Jochen ; Rosson, Mary B. ; Quiñones, Manuel A. P.: End User Development of Web Applications. In: Lieberman, Henry (Hrsg.) ; Paternò, Fabio (Hrsg.) ; Wulf, Volker (Hrsg.): *End User Development* Bd. 9, Springer Netherlands, November 2006 (Human-Computer Interaction Series), 161-182

[Smith u. a. 2003] Smith, David A. ; Kay, Alan C. ; Raab, Andreas ; Reed, David P.: Croquet - A Collaboration System Architecture. In: *C5 2003 Conference on Creating, Connecting and Collaborating through Computing (C^5 2003)*. Los Alamitos, Kalifornien : IEEE Computer Society, Januar 2003, 2

[Stanley 1992] Stanley, A. E.: *HyperTalk and HyperText: Programming the Graphic Interface in the Macintosh and Windows environment with HyperCard 2 Plus*. Oxford, Großbritannien : Butterworth-Heinemann, 1992

[Sun 2002] Sun, Chengzheng: Undo as concurrent inverse in group editors. In: *ACM Transactions on Computer-Human Interaction* 9 (2002), Nr. 4, S. 309–361

[Sun u. Chen 2002] Sun, Chengzheng ; Chen, David: Consistency maintenance in real-time collaborative graphics editing systems. In: *ACM Transactions on Computer-Human Interaction* 9 (2002), Nr. 1, S. 1–41

[Sutherland 1963] Sutherland, Ivan E.: *Sketchpad, A Man-Machine Graphical Communication System*. New York : Garland Publishing, 1963 (Outstanding Dissertations in the Computer Sciences)

[Taivalsaari 2008] Taivalsaari, Antero: *The Sun Labs Lively Kernel: A Technical Overview*. `http://research.sun.com/projects/lively/LivelyKernel-TechnicalOverview.pdf`. Version: Februar 2008. – nicht veröffentlicht

[Taivalsaari u. a. 2008] Taivalsaari, Antero ; Mikkonen, Tommi ; Ingalls, Dan ; Palacz, Krzysztof: Web Browser as an Application Platform: The Lively Kernel Experience. / Sun Microsystems Laboratories. 2008 (TR-2008-175). – Forschungsbericht

[Wheeler 2004] Wheeler, Kyle: *HyperTalk: The Language for the Rest of Us*. `http://www.memoryhole.net/~kyle/papers/hypertalk.pdf`. Version: Januar 2004. – nicht veröffentlicht

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe. Es wurden keine anderen als die angegebenen Quellen verwendet. Zitate sind entsprechend kenntlich gemacht.

Potsdam, den 13. November 2009