Anforderungen an moderne Browser-basierende Web-Applikationen

Stefan Raubal

März 2001

Diplomarbeit in Telematik

durchgeführt am

Institut für Informationsverarbeitung und
Computergestützte Neue Medien
der Technischen Universität Graz

Begutachter: O.Univ.-Prof. Dr.phil. Dr.h.c. Hermann Maurer
Betreuer: Dipl.-Ing. Thomas Dietinger

Ich versichere, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Zusammenfassung

In den letzten Jahren wuchsen die Anforderungen an das erst seit kurzem bestehende Genre der Web-Applikationen rapide: Bereiche wie e-Commerce, e-Banking oder e-Learning verlangen nach sicheren Übertragungen, schnellen Antwortzeiten und immer mehr Funktionalität. Diese Anforderungen sind nur durch aufwendiges und bedächtiges Design sowie die Nutzung von unterstützenden Plattformen und Werkzeugen erfüllbar. Diese Diplomarbeit versteht sich als Einführung in den vielschichtigen Bereich der Techniken rund um Web-Applikationen, stellt einige davon genauer vor und versucht, Software-Entwicklung unterstützende Prinzipien wie Design Pattern näherzubringen. Der praktische Teil befasst sich mit der Realisierung eines Browser-unabhängigen Application Window Toolkits in JavaScript für die Firma Hyperwave Graz. Dabei wurde besonders darauf geachtet, einen qualitativ hochwertigen Planungs- und Entwicklungssprozess durchzuführen, sowie, möglichst viele Erkenntnisse aus dem theoretischen Teil einfließen zu lassen. Die detailliert beschriebene Entstehung des Endprodukts wird abgeschlossen von einem Ausblick, in welche konkreten Anwendungen das Ergebnis integriert werden kann.

Abstract

Over the last few years the requirements for the still very young genre of web applications have increased rapidly: fields like e-commerce, e-banking or e-learning call for secure transactions, fast response times and ever growing functionality. These demands can only be satisfied through elaborate and thoughtful design as well as the use of supporting platforms and tools. This thesis serves as an introduction to the different topics and techniques concerning web applications. It describes in some detail several of these methods and presents approaches like Design Patterns. The practical part deals with the realization of a browser independent Application Window Toolkit in JavaScript, carried out for Hyperwave Graz. Major emphasis was placed on keeping design and implementation at a high quality level and on including most of the insights of the theoretical part in the practical work. The detailed description of the product's development is concluded with a summary of possible perspectives of concrete applications for the developed toolkit.

Danksagung

Mein aufrichtiger Dank gilt allen, die mich ein Stück auf dem Weg zu dieser Diplomarbeit begleitet haben.

Meinem Betreuer Dipl.-Ing. Thomas Dietinger möchte ich für seine begeisternde und aufmunternde Art, mich zu unterstützen, danken. Neben den organisatorischen und fachlichen Hilfestellungen war sie es, die meine Schritte beschleunigte.

Danke, Thomas.

Keiner hat mich so intensiv ertragen müssen wie Dipl.-Ing. Simon Zwölfer, der mit mir über ein halbes Jahr sein Büro teilen musste. Stand er mir zwar offiziell nur in technischen Belangen zur Seite, so wurde er mir zu einem Freund und unterstützenden Wegbegleiter in vielerlei Dingen.

Danke, Simon.

Abgesehen von all jenen, die mir mit kleineren und größeren Ratschlägen oder Hinweisen den Weg geebnet haben, möchte ich allen danken, die mir in den letzten Monaten abseits der Arbeit zur Seite standen. Vor allem meiner Freundin gilt hier meine Dankbarkeit für ihr Vertrauen, ihr Verständnis und ihre Unterstützung.

Danke, Isabella.

Darüberhinaus danke ich der Firma Hyperwave sowie dem Institut für Informationsverarbeitung und Computergestützte Neue Medien für die Unterstützung, die maßgebend zum Zustandekommen der Arbeit in dieser Form beigetragen hat.

Stefan Raubal

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

"Lesen Sie schnell, denn nichts ist beständiger als der Wandel im Internet!"
Anita Berres, dt. Publizistin

1.1 Problemstellung

Viele Tätigkeiten des täglichen Lebens können heute schon über das Internet abgewickelt werden: Vom virtuellen Einkaufsbummel mit abschliessender Bestellung der gewünschten Artikel über Kontoverwaltung und Abwicklung von Versicherungsgeschäften bis hin zum Absolvieren von Fortbildungs-Kursen ist alles mit einem Web-Browser und ein paar Mausklicks zu erledigen. Kaum eine Sparte gibt es, der noch kein kleines "e" vorangesetzt wurde.

Voraussetzung für diese Entwicklung – ob sie nun gutgeheißen oder abgelehnt wird – sind all die Technologien, die sich rund um das World Wide Web (WWW) gebildet haben bzw. immer noch neu bilden. Vor allem die zweite Hälfte der 90er-Jahre zeichnete sich durch enorme Fortschritte in diesem Bereich aus, unterstützt durch die Tatsache, dass sich das Internet in atemberaubender Geschwindigkeit verbreitet hat[1].

So entstand in den letzten Jahren der Begriff der "Web-Applikation", da die Internet-Seiten immer mehr Funktionalität boten. Die Ära der Text-basierenden Homepages ging zu Ende und es begann die Zeit der Web-Designer und Web-Entwickler. Mit allen grafischen Tricks versuchen erstere, den Informations- oder Unterhaltungs-suchenden Surfer auf der Seite zu halten. Die Web-Entwickler widerum müssen den schnell wachsenden Besucherzahlen mit einer skalierbaren Architektur und entsprechender Hardware begegnen, außerdem muss für sichere Transaktionen gesorgt werden.

Beispiels-Web-Applikation dieser Arbeit ist ein Web-Based-Training-System – die Hyperwave eLearning Suite (eLS). Für ein solches Produkt ergeben sich Anforderungen, die über jene vieler anderer Web-Applikationen hinausgehen:

[1] Laut [22] hat sich die Zahl der Österreicher, die mehrmals monatlich das Internet nutzen, in den letzten drei Jahren verdreifacht. Damit sind 39% aller über 14-Jährigen regelmäßig "online".

- Kurse müssen einfach zu erstellen bzw. anzupassen sein,
- verschiedene Rollen (Leiter, Tutor, Lernender usw.) sind zu unterscheiden und
- das Programm sollte ohne viel Aufwand auch via CD-Rom nutzbar sein.

Mit den Problemen eines solchen Web-Based-Training-Systems beschäftigt sich diese Diplomarbeit – weniger mit Aspekten wie Benutzer-Verwaltung oder Datenbank-Anbindung, sondern mehr in Richtung Generierung einer skalierbaren, modularen Architektur zur Darstellung der Kurse im Browser.

1.2 Web-Applikationen

Was bedeutet nun eigentlich der Begriff "Web-Applikation"? Worin besteht der Unterschied zu einer Web-Seite? Jim Conallen beantwortet diese Fragen in [3] folgendermaßen: "The distinction between web sites and web applications is subtle, and relies on the ability of a user to effect the state of the business logic on the server. Certainly if no business logic exists on a server, the system should not be termed a web application. For those systems where the web server (or an application server that uses a web server for user input) allows business logic to be effected via web browsers, the system is considered a web application."

Das heißt, der Benutzer einer Web-Applikation muss über den Browser die Möglichkeit haben, mittels geeigneter Eingabehilfen (meist handelt es sich dabei um Formulare) den Zustand des Applikations-Systems zu verändern (in vielen Fällen bedeutet das einen Datenbank-Zugriff, u.U. mit weiteren Auswirkungen). Dabei sollte der Benutzer die für ihn notwendigen Rückmeldungen und Sichten auf das System bekommen. Ein uns allen vertrautes Beispiel für eine simple Web-Applikation stellt ein Gästebuch für eine Homepage dar. Eine solche Mini-Anwendung ist natürlich nicht mit einem Tele-Banking-System vergleichbar, aber vom prinzipiellen Ablauf her passiert nicht viel anderes.

1.2.1 Moderne Web-Applikationen

Wie uns das Gästebuch-Beispiel gezeigt hat, gibt es schon längere Zeit Anwendungen im Internet, die man als Web-Applikation bezeichnen kann. Doch seit der Einführung von CGI und HTML-Formularen hat sich viel getan:

- Der typische Internet-Benutzer ist nicht mehr der technische Experte, wie es vor weniger als 10 Jahren noch der Fall war.
- Die Anwendungen müssen mit dem Zigfachen an Zugriffen zurechtkommen.
- Die Systeme modellieren immer komplexere Abläufe, deren Zustände nicht mehr einfach abbildbar und für den menschlichen Geist durchschaubar sind.
- Die aufwendige grafische Gestaltung verlangt zusätzlichen Skript-Code und die Berücksichtigung von Browser-Unterschieden.

Trotz – oder gerade auf Grund – der Vielschichtigkeit und Komplexität von Web-Applikationen hat sich noch kaum eine Design-Kultur gebildet, wie es im Bereich der Funktionalen oder Objekt Orientierten Programmierung schon länger der Fall ist. Einen Grund dafür sieht Jim Conallen ([3]) in den zur Verfügung stehenden Hilfsmitteln: "Current development environments make it so easy to produce simple web applications that they have the unfortunate side effect of encouraging us to develop and evolve applications in the absence of serious analysis and design."

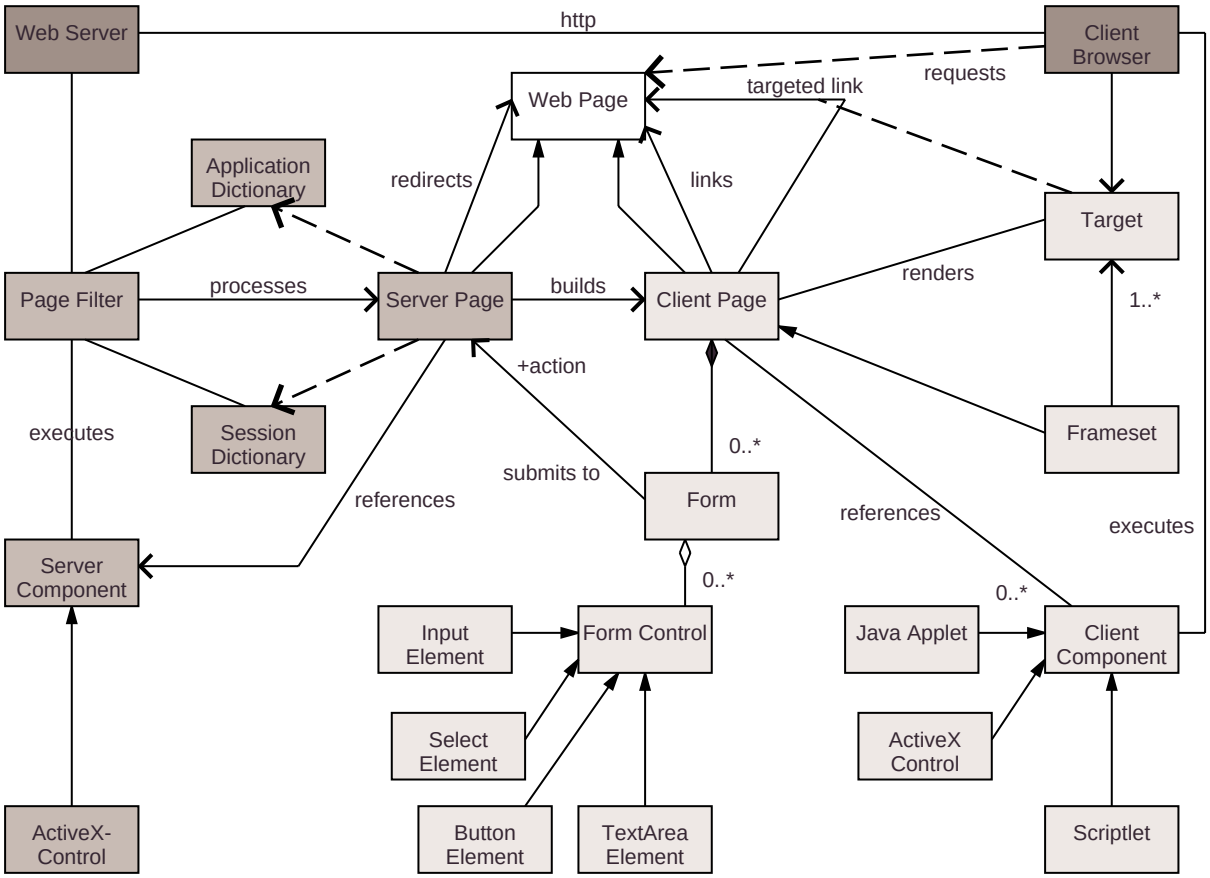

Abbildung 1.1: Allgemeines Modell einer Web-Applikations Architektur.

Abbildung 1.1 (entnommen aus [3] und basierend auf Microsofts Active Server Page Technologie) zeigt trotz vereinfachter Darstellung der Interaktionen, dass es sich bei einer Web-Applikation um ein extrem komplexes System handelt. Auch wenn im allgemeinen nicht alle Möglichkeiten, die in der Abbildung gezeigt werden, genutzt werden – allein die Menge der verschiedenen Techniken, die angewandt werden können, bedürfen der Auswahl eines erfahrenen Entwicklers.

Ein Punkt, der gerade bei Anwendungen für das Web sehr wichtig ist, ist Skalierbarkeit. Web-Applikationen haben damit zu rechnen, hunderte bis tausende (beinahe) gleichzeitige

Anfragen bearbeiten zu müssen. Für alle Benutzer müssen Session-Daten gehalten werden, Datenbank-Requests durchgeführt und Web-Seiten generiert und zurückgeschickt werden. Viele potentielle Flaschenhälse verbergen sich im System – und oft ist die Anzahl der zu erwartenden Benutzer nicht vorherzusehen.

Und nicht nur in Bezug auf Performance gilt es, skalierbar zu sein. Die Web-Applikation sollte auch gut wartbar und mit neuer Funktionalität bzw. Schnittstellen zu anderen Systemen erweiterbar sein. Fragen wie "...und eine französische Version ist ja sicher kein großer Aufwand, oder?" lassen so manchen Web-Entwickler erschaudern.

1.2.2 Hybride Systeme

Der biologische Fachausdruck "Hybride" wurde in den letzten Jahren mehrfach auch in den technischen Bereich übertragen (man denke an die Bezeichnung "Hybrid-Antrieb" für Motoren, die über verschiedene Quellen ihre Energie beziehen). Gemeint sind jeweils Mischformen, die zwei oder mehrere unterschiedliche Aspekte vereinen.

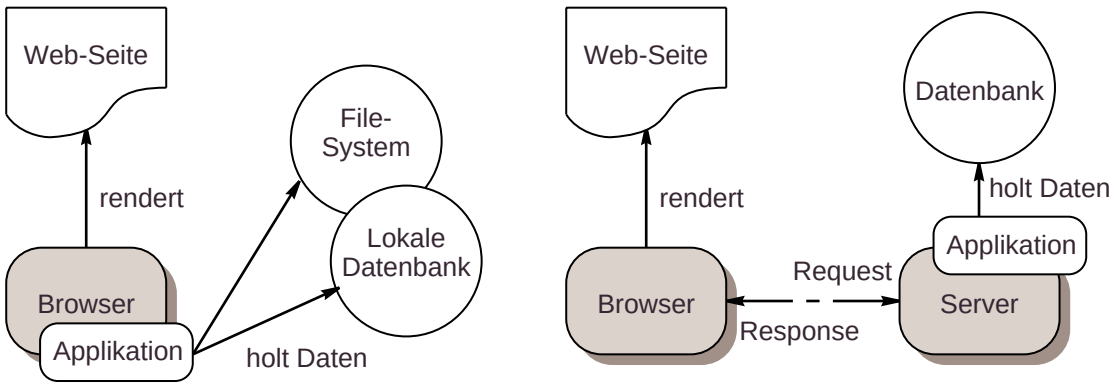

Abbildung 1.2: Web-Applikation im offline- (links) und online-Modus (rechts).

Die Beispiels-Applikation eLearningSuite will in zweifacher Hinsicht einem hybriden Ansatz entsprechen. Zum einen soll ein Kurs sowohl via Inter-/Intranet als auch via CD-Rom/DVD zu absolvieren sein. Damit im zweiten Fall nicht all die Vorteile von Web-basierendem Lernen verloren gehen, soll der Benutzer die Möglichkeit haben, Daten wie Anmerkungen, Forums-Nachrichten etc. zu einem gewünschten Zeitpunkt mit dem Online-System zu synchronisieren. Interessant ist das vor allem für all jene, die über keine breitbandige und dauerhafte Verbindung zum Internet verfügen. Außerdem können Videos in einer viel besseren Qualität auf einem austauschbaren Datenträger gespeichert werden, als es heutige Streaming-Techniken erlauben.

Der zweite Hybrid-Ansatz besteht darin, dass es keine eindeutige Trennung in Client-seitiges und Server-seitiges Denken geben soll. Einzelne Module sollen sowohl am Applikations-Server als auch im Browser integrierbar sein (dafür bieten sich als Sprache z.B. JavaScript

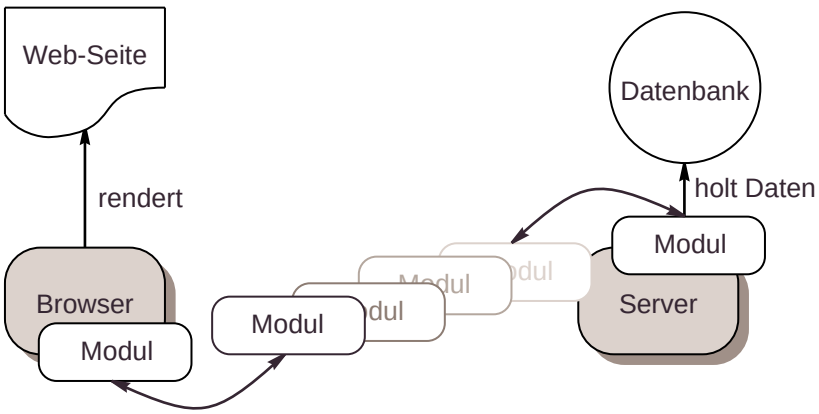

Abbildung 1.3: Austauschbarkeit von Modulen zwischen Server- und Client-Seite.

und Java an). Diese Anforderung verlangt eine präzise Definition der Schnittstellen und ein flexibles Framework.

1.3 Grobspezifikation

1.3.1 Problemstellung

Die Hyperwave eLearning Suite (ehemals GENTLE-WBT) leidet zur Zeit an dem Problem, ein reines Online- bzw. ein reines Offline-System (unter Verwendung des CD-Publishers) zu sein. Gesucht ist nun ein hybrides System, das auch offline seine Stärken (Annotationen, Diskussions-Forum, dynamische Erweiterbarkeit etc.) behalten kann. Dazu ist eine Synchronisation bzw. Replikation des System-Status notwendig, welche vom Benutzer, der sonst offline arbeitet, bei einer bestehenden Internet-Verbindung vorgenommen werden kann (bzw. automatisch geschieht). Abgesehen davon soll auch eine Server-Unabhängigkeit erreicht werden – konkret heißt das, eine Loslösung vom reinen Hyperwave-Konzept anzustreben. Schließlich gilt es, eine leicht customizierbare Plattform zu schaffen, die einfach an Wünsche neuer Kunden angepasst werden kann und Browser-unabhängig sein soll.

1.3.2 Schichtenmodell

Um Portabilität, Skalierbarkeit, Modul-Austauschbarkeit etc. zu gewährleisten, ist es für eine Applikation wichtig, ihre Datenflüsse über definierte Layer abzuwickeln. Um so mehr gilt dies für eine hybride Web-Applikation im oben definierten Sinne. Betrachtet man die herrschenden Browser-Inkompatibilitäten, so wird der Bedarf nach einer solchen Lösung noch deutlicher. Dabei gilt es, mehrere Schichten zu implementieren, die in der Abbildung 1.4 dargestellt sind.

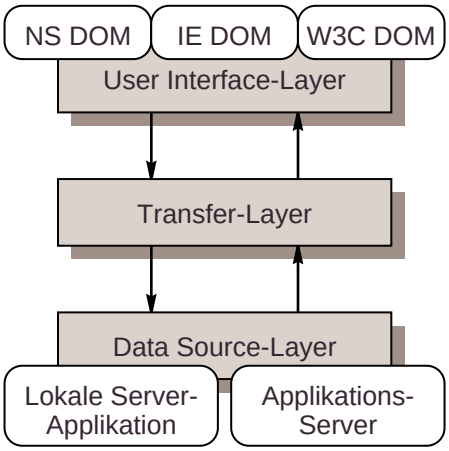

Abbildung 1.4: Grobe Schichten-Einteilung für eine Web-Applikation.

Die oberste Schicht, der User Interface-Layer, sorgt für die endgültige Darstellung der Daten auf einem spezifischen Browser, dem passenden Dcoument Object Model entsprechend (z.B. bei Netscape 6 nach dem W3C DOM Level 1 bzw. 2). Dieser Layer stellt eine Kompatibilitätsschicht nach unten – er gleicht also die unterschiedlichen Wege, wie Dynamic HTML von den führenden Browser-Herstellern implementiert wurde, aus.

Die mittlere Schicht kapselt Cache-Funktionalität und ermöglicht diverse Transformationen, die gegebenenfalls notwendig sind, um Roh-Daten für den User Interface-Layer vorzubereiten.

Von wo die jeweiligen Daten zu holen sind, entscheidet die dritte Schicht, der Data Source-Layer. In der Offline-Version greift er auf einen lokalen Server zu (z.B. über TCP/IP oder in einem Applet), online auf den Applikations-Server im Internet (z.B. Hyperwave-Server). Natürlich können mehrere Schichten zu einer zusammengefasst bzw. in weitere aufgeteilt werden – wie detailliert das Schichtenmodell implementiert wird, hängt von den Gegebenheiten ab.

1.3.3 Ansatz dieser Arbeit

Die hier betrachtete Arbeit wird sich mit den obersten Schichten eingehender beschäftigen, im speziellen mit dem User Interface-Layer sowie mit der Kommunikation zwischen diesem und dem Transfer-Layer. In erster Linie geht es sowohl um konzeptionelle Fragen als auch um konkrete Ansätze. Zur Lösung des Problems sollen schon implementierte Technologien bzw. Dokumente über solche gefunden und analysiert werden. Es ist immer im Auge zu behalten, dass das 5-Layer-Customization-Modell der Hyperwave eLearning Suite aufrechterhalten wird. Dazu sollte eine einfache Austauschbarkeit der Ressourcen gewährleistet sein.

Das Endprodukt sollen Dynamic HTML Foundation Classes sein, auf deren API die

Module weiterer Hyperwave-Applikationen aufbauen können. Das API soll sowohl auf low-level-Ebene (GUI-Primitives wie Buttons, Eingabefelder und ähnliches) als auch auf höherem Abstraktionsgrad (Tree-View, Wizard usw.) erweiterbar sein.

Kapitel 2

Historisches

"Wir können aus der Geschichte nichts lernen,
außer, dass wir nichts aus der Geschichte lernen."
Georg Wilhelm Friedrich Hegel, Philosoph

2.1 Entwicklung von Applikations-Technologien

2.1.1 Schwerpunkt Web-Applikationen

Die historische Entwicklung von Technologien zur Gestaltung von dynamischen Web-Seiten bzw. kompletten Applikationen ist ein langer Weg, der 1991 am CERN mit der Erfindung des WWW seinen Anfang nahm. Mittlerweile sind eine kaum überschaubare Anzahl von Techniken und deren Implementationen am Markt.

Ziel dieses Abschnitts ist es, einen Überblick über diese Entwicklung und die entstandenen Technologien zu bieten. Eine gute Einführung in dieses Thema bietet auch [27].

Statisches HTML

Hier kann bei einer Web-Seite kaum von einer Applikation die Rede sein, vielmehr von einer Ansammlung von Information. Die einzige Interaktionsmöglichkeit durch den Benutzer (das Wort "Besucher" ist hier noch passender) besteht in der Wahl eines Links, dem er folgen möchte.

HTML-Formulare, http-Post/Get, CGI

Durch die Kombination dieser Techniken (vor allem durch das Common Gateway Interface) kann erstmals von einer Applikation gesprochen werden. Mittels Formularen kann von Benutzerseite Input an das System erfolgen, durch CGI kann entsprechend darauf reagiert werden – Seiten können erstmals wirklich dynamisch erzeugt werden.

Was auf privaten Pages zur euphorischen Nutzung von Gästebüchern und Zugriffszählern führt, wird auf professioneller Seite genutzt, um Suchsysteme zu kreieren und zu optimieren.

Cookies

Dank Cookies kann auf einfache Weise Information am Client (sogar über eine Session hinaus) gespeichert werden. War es vorher nur möglich, durch Weitergabe des States via URL den Zustand einer Web Applikation zu speichern, geht das nun (sofern der Benutzer diese zulässt) mittels Cookies.

Wie bei fast jeder Technologie fürs Web folgte auf eine Welle der Begeisterung eine Skepsis bezüglich Datensicherheit. Mittlerweile zählen Cookies durchaus zu den akzeptierten Techniken[1].

Client Side Scripting (JavaScript, JScript, VBScript)

Diese in Netscape 2.0 erstmals eingeführte Technik bietet im Grunde wenig neue Funktionalität – vielmehr ermöglicht sie, Vorhandenes zu einem neuen Ganzen zusammenzufügen. Durch die Scriptsprachen hatte der Entwickler plötzlich Zugriff auf Teile der API des Browsers (Bookmarks, History, etc.).

Was auf der einen Seite für Animationen, blinkende Lauftexte und Roll-Over-Buttons benutzt wird, findet auch große Anwendung in Bereichen, die nicht direkt etwas mit dem GUI zu tun hatten. Form-Inhalte können nun auf der Client-Seite überprüft werden, Event-Dialoge und die genaue Kontrolle über Fenster ermöglichen bessere Kommunikation mit dem Benutzer. Der mangelhafte Zugriff auf das API eines Web-Dokuments führte später zur Entwicklung von Dynamic HTML (siehe dort). Client seitige Scriptsprachen revolutionierten den Stand der Web Applikationen nicht, das geschah vielmehr durch Server Side Scripting (siehe dort).

Plug-Ins

Das Plug-In Interface der Browser ermöglichte es, andere Dokumentsprachen als HTML zu integrieren, ohne die Browser immer größer und größer werden zu lassen. Der Nachteil dabei besteht im notwendigen Download und der Installation des Plug-Ins. Einmal besorgt, ermöglichen sie jedoch – gerade im Multimedia-Bereich (Shockwave, RealAudio, RealVideo, etc.) – die Realisierung von beeindruckenden Web-Seiten. In anderen Anwendungsbereichen haben sie sich aber nicht etablieren können.

Active-X

Die Active-X Technologie ermöglicht Browsern, die Ressourcen von Microsoft Windows zu nutzen. Somit kann praktisch jede Windows Ressource im Browser eingebettet werden.

[1] Eine Statistik aus dem Jahr 1998 bezüglich Cookie-Akzeptanz ist bei [12] zu finden.

Allerdings ist diese Technologie auf ein Windows System mit Internet Explorer angewiesen – deshalb wird Active-X nicht in dem Maße genutzt, wie es seiner Mächtigkeit entspräche.

Java Applets

Applets ermöglichen es, unterschiedlich eingeschränkte[2] Java Applikationen im Browser zu starten. Dank JDBC ist es mittels Java leicht möglich, zu verschiedensten Datenbanken eine Verbindung aufzubauen (für Applikationen extrem wichtig) und da Java fürs Netz entwickelt wurde, stehen meist schon alle notwendigen Klassen zur Verfügung.

All den Vorteilen stehen jedoch zwei Nachteile gegenüber: Zum einen dauert der Download einer aufwendigen Klassenstruktur recht lange und zum zweiten muss der Browser eine passende Java Virtual Machine (JVM) anbieten. Das Problem dabei ist, dass die bereits in den Browsern integrierten VM's generell einige Versionen älter als die aktuellen API's sind. D.h. der Benutzer kann u.U. genötigt sein, sich zusätzlich noch ein passendes Plug-In besorgen (was z.B. für JFC (z.B. Swing) notwendig ist. Außerdem mangelt es an der Möglichkeit, von Java aus auf JavaScript zuzugreifen (nur in den neuesten Browsern via Netscapes LiveConnect möglich). Damit fehlt einem Applet die Möglichkeit, auf das Document API einzuwirken (es kommt also noch nicht als Ersatz für JavaScript in DHTML in Frage).

Server Side Includes, Server Side Scripting

Spezielle Tags direkt im HTML-Code werden in dieser Technologie direkt vom Web-Server interpretiert. Am Browser kommt nur mehr "reines" HTML an. Die ersten Ansätze führten direkt Shell-Scripts aus, was zu extremen Sicherheitslücken führte und nicht sehr flexibel bzw. Plattform-unabhängig war. Aber auf jeden Fall boten diese Ansätze eine großartige Möglichkeit für Webapplikationen.

Aus diesen Ansätzen entwickelten sich zwei Skriptsprachen, die heute weit verbreitet und in sichere Umgebungen eingebunden sind: PHP und Server Side JavaScript. (Über die enorme Verbreitung von PHP (als Apache-Modul) gibt u.a. [28] Auskunft.) Gerade die enge Verschränkung durch Code im HTML-File ist dabei eine Gefahrenquelle für die Skalierbarkeit und Wartbarkeit einer Webapplikation. Umfangreiches Wissen über HTML, Style Sheets und die jeweilige Skriptsprache sind unumgänglich.

Web Application Server

Aus dem Ansatz der Server Side Include's entstand eine neue Plattform für Webapplikation, die sg. Web Application Server. Sie bieten ein ausformuliertes API an, um schnell und effizient Server-seitig Dynamik in eine Site integrieren zu können. Sie ersetzen meist vollständig

[2]Grundsätzlich werden Applets in einer so genannten "Sandbox" ausgeführt, in der sie Restriktionen unterliegen (kein Zugriff auf lokale Dateien, keine Netzwerk-Verbindungen zu anderen Hosts, kein Starten von Applikationen am Client, Fenster sehen anders aus...). Indem Applets signiert werden, können diese Beschränkungen aufgehoben werden.

einen herkömmlichen Web-Server (oder bilden einen weiteren Applikations-Layer hinter diesem).

Unter die bekanntesten WAP's fallen BEA's WebLogic, Allaires ColdFusion und IBM's WebSphere[3]. Für reine Web-Applikationen erscheint eine solche Plattform optimal – allerdings ist man damit auf einem vorherbestimmten Weg (und abhängig vom Anbieter des Servers).

Servlets

Die Möglichkeit zu einem Java Applikations-Layer hinter (oder in) einem Web-Server bieten Servlets. Dies sind Java-Klassen, die auf einem API von Sun aufbauen und im Request/Response-Mode agieren. Was bei einem Request getan werden soll, obliegt vollständig dem Entwickler – er hat dabei sämtliche Funktionen (und Vorteile) von Java zur Verfügung. Dank JDBC, JNDI oder RMI stehen die Wege zu sämtlichen Datenbanken, Directory-Services oder zu anderen remote Klassen offen. Kein Wunder also, dass Servlets zu einer der führenden Plattformen für Webapplikationen geworden sind.

Servlets werden entweder in Servlet Engines oder in Java Web-Servern selbst exekutiert und sind – entgegen anderslautenden Gerüchten über Java – alles andere als langsam (siehe dazu [26]). (Anmerkung: Was an Java langsam ist, ist das GUI und das Eventhandling.) Was Skalierbarkeit, Internationalization etc. angeht, ist Java auf jeden Fall eine gute Wahl.

JavaServer Pages (JSP)

Diese auf Servlets basierende Skripting-Technologie soll hier gesondert betrachtet werden, da sie sich binnen kurzer Zeit zu einer häufig genutzten Technik entwickelte. JSP entstand aus dem Umstand, dass die Nutzung von Servlets zur Generierung von HTML Code sehr umständlich ist und bei jeder Content-Änderung der Code angegriffen werden muss. JSP erlaubt es, Java Code direkt in ein HTML File einzufügen, welches dann am Server ausgeführt und mit den Ergebnissen des Java-Codes vermischt wird.

Leider bringt das nicht die gewünschte Trennung von Code und Content, denn damit liegt die Programmlogik beim Inhalt (und nicht umgekehrt wie früher). Der Einsatz von JavaBeans vermindert den Störfaktor dieser Verschachtelung, jedoch nicht zur vollen Zufriedenheit. Zur Diskussions-Beitrag zu diesem Thema sind [16] und [17] lesenswert.

Dynamic HTML

Die Einführung des DOM (Document Object Models) und die Erweiterung der Fähigkeiten von Style Sheets ermöglichte es schließlich, auf Seite des Clients für volle GUI-Dynamik sorgen zu können. Dank DOM wird jedes Zeichen eines Dokuments ein GUI Widget, dank Style Sheets kann Dynamik integriert werden und gleichzeitig Content von Design getrennt werden. Damit ist dHTML das aktuelle Schlagwort für jeden Web-Designer, allerdings

[3] Eine Liste Java-basierender Applikations-Server bietet [33].

reduzieren sich diese Erweiterungen fast ausschließlich auf den grafischen Bereich (das GUI einer Web-Applikationen kann somit zumindest einfacher designed und gewartet werden).

Das große Problem auf diesem Bereich ist die Browser-Inkompatibilität zwischen Netscape und Internet Explorer (siehe dazu Kapitel 4).

XML/XSL/XSLT/XSP

Abgesehen davon, dass XML (Extensible Markup Language) und XSL (Extensible Stylesheet Language, anfangs Extensible Style Language genannt) für statische Seiten einsetzbar sind bzw. sein werden (als Ersatz/Ergänzung zu HTML und CSS), zeigen sie in Zusammenhang mit XSLT ihre Mächtigkeit. Die XSL Transformation definiert, wie XML Seiten umgestaltet werden sollen, um ein neues Dokument zu erhalten, welches in Struktur und Inhalt komplett verändert sein kann. Dieser result tree kann – wenn er nur ein Zwischenergebnis darstellt – via XSL in die gewünschte Endform (z.B. HTML, PDF, TXT) gerendert werden.

Bei XSP wird diese Technologie serverseitig eingesetzt und insofern erweitert, als dass Java Code in ein XML Dokument eingefügt werden kann. Dieser wird dann compiliert und im resultierenden Dokument durch seine Resultate ersetzt (vgl. JSP, ASP). Dank XML, XSLT und XSL kann eine ausgezeichnete Trennung zwischen Logik-Programmierung, Content-Generierung (Authoring) und Design ermöglicht werden.

2.1.2 Schwerpunkt "klassische" Applikationen

Stand-Alone Applikationen

Die klassischen Stand-Alone Applikationen bilden das Gegenstück zu statischem HTML auf Web-Applikations-Seite. Sie bilden die Basis, auf der Programm-Interaktionen zwischen mehreren Rechnern aufsetzen.

Client-Server Architekturen

Diese schon sehr früh entstandene Architektur erhielt mit TCP/IP und UDP 1981 zwei Protokolle, das sich schnell durchsetzten. (Darauf basiert auch jetzt noch praktisch alles, was mit dem Begriff Web-Applikation zu tun hat.) Die Entwicklungen auf diesem Gebiet bildeten den Nährboden für Protokolle, die im Internet verwendet werden: HTTP, FTP, HTTPS usw. Für erweiterte Techniken zum Handshake, zur Fehlerkontrolle, zur Verschlüsselung etc. bestand dadurch große Nachfrage.

Weiters entstanden Datenbank-Systeme, die als Server Daten anboten (SQL etablierte sich hier als Standard). Einige der Hersteller von SQL-Servern (Microsoft, Oracle, usw.) sprangen schnell auf den Internetzug auf und kamen mit Web-Servern auf den Markt, die eine einfache Anbindung an Datenbanken ermöglichen.

Interessant ist, dass manche Anbieter (z.B. Oracle) die Wichtigkeit von Web-Unterstützung erkannt haben und ihre Datenbank-Systeme in Web Application Server integriert haben.

CORBA

CORBA (Common Object Request Broker Architecture) entstand aus der Notwendigkeit, Objekte (im OOP-Sinn) remote zur Verfügung stellen zu können. Sowohl die Entwickler als auch die Applikationen verteilten sich in immer mehr Netzen (LAN's bzw. WAN's) – der CORBA Standard bot Abhilfe. Vorerst stark im CPP-Sektor benutzt, wurde er mit dem Aufkommen von Java auch dafür verwendet. Allerdings sollte für den Einsatz von remote Objekten eine genügend große, gesicherte Bandbreite zur Verfügung stehen, der Einsatz beschränkt sich damit eher auf den Intranet-Bereich.

Java (RMI, JNDI, Jini)

Die Einführung von Java (anfangs gedacht als Sprache für Haushaltsgeräte) war Indikator der Tendenz (und Erfüllung des Wunsches), Plattform- und Orts-unabhängig zu entwickeln und zu distributieren. Für das Web war vor allem die Möglichkeit interessant, mittels Applets in einer HTML-Seite eine Applikation ausführen zu können.

Mit RMI (Remote Method Invocation) bietet Java eine Alternative zu CORBA an, die dank HTTP-Tunneling auch Firewalls integrieren kann. Durch die Kombination Applet, Servlet und RMI bestehen theoretisch kaum mehr Restriktionen für eine vollständige Verteilung der Module einer Applikation.

JNDI (Java Naming and Directory Interface) bietet Java Applikationen eine Schnittstelle zu Naming- und Directory-Services – dies ist in der verteilten Welt immer wichtiger.

Jini ist eine Technologie, um Services in einem Netzwerk auf möglichst einfache Methode Clients zur Verfügung zu stellen. Diese Dienste (Applikationen, Server, Drucker, etc.) sollen Plug-and-Play mäßig beliebig in oder aus dem Netz genommen werden können. Ebenso können Clients jederzeit anfragen, welche Dienste gerade angeboten werden.

2.1.3 Ausblick

Verfolgt man die in den vorigen Abschnitten aufgezeigten Entwicklungen (Web-Applikationen und "klassische" Applikationen), so wird klar, dass beide Strömungen immer enger zueinanderwachsen. Web-Applikationen werden immer komplexer, verlangen immer mehr Sicherheits-Vorkehrungen, Performance-Boosts, Ressourcen-Verwaltung etc., andererseits sollen viele Applikationen auf Browsern oder anderen Thin-Clients lauffähig, konfigurierbar oder wartbar sein.

Sollte sich XML als Daten-Beschreibungs- und Strukturierungs-Sprache durchsetzen, was durchaus zu erwarten wäre, so würde dies noch weiter die beiden Welten miteinander verschmelzen lassen.

Allmählich stellt sich die Frage, wo das Ende der Möglichkeiten der Browser-Programmierung abzusehen ist. Durch XML als Datenformat, mächtige Möglichkeiten für die Generierung des grafischen Benutzer-Interfaces sowie objektorientierte Skript-Sprachen, die mit jeder Version ausgefeilter werden[4], stellen Browser eine Art "Betriebssystem im Betriebssystem"

[4] Man denke dabei z.B. an das Exception-Handling in JScript 5.

dar. Denkt man diesen Gedanken zu Ende, leuchtet die Notwendigkeit eines freien, von mehreren Firmen beherrschten Browser-Marktes deutlich ein.

2.2 Zwei Web-Applikationen im Detail

Auf dem Bereich der Web-Desktops, die in den letzten Jahren vermehrt aufgetaucht sind, finden sich recht aufwendig gestaltete Web-Applikationen. Als Web-Desktop bezeichnet man eine Web-Seite mit Funktionen zum Verwalten von Bookmarks und Mail-Accounts, zum Administrieren von Terminen (via Kalender) und meist auch zum Speichern von (einer beschränkten Menge von) Daten auf einem Server. Dabei sticht das Projekt Desktop.com hervor, welches ein offenes API zur Verfügung stellt, durch das es möglich ist, eigene Applikationen für Desktop.com zu schreiben.

2.2.1 Desktop.com

 http://www.desktop.com/

Dieses Projekt bietet neben den üblichen Funktionen wie Email- und Bookmark-Verwaltung, Virtuelles Fotoalbum, Terminplaner, etc. einen remote Speicherplatz (Desktop Drive), ein ausgefeiltes GUI und vor allem ein offenes API zur Erstellung eigener Applikationen.

In den Anfängen unterschieden sich die grafische Präsentation unter Netscape und Internet Explorer stark, da viel Client-seitige Funktionalität in ActiveX, JScript 5.0 und HTML Components integriert war. Mittlerweile wurde das klassisch Table-basierende Layout unter Netscape in ein Frame- und Layer-gestütztes umgewandelt. Gewisse Features wie die Fenster im Desktop-Fenster sind unter Netscape nicht realisierbar – das Look-and-Feel bleibt ein anderes als im Produkt von Microsoft.

Seit den Anfängen der Web-Präsenz von Desktop.com wird auf das offene API hingewiesen, doch Stand November 2000 fehlt immer noch die genaue Beschreibung desselben. Das Framework nennt sich “Luna” und basiert auf einem Vermischen von ML-Templates und Perl. Ergänzend dazu bietet das Projekt ein Event-Modell, das eigene Events und deren Handler zu definieren erlaubt. Dazu wurde ein “Data Namespace” eingeführt, der den Zugriff auf Templates, Query-Strings, Cookies, Applikationen usw. ermöglicht.

Leider fällt beim Nutzen des Systems bald die verhältnismäßig hohe Trägheit auf. Von einem Arbeiten wie am lokalen Desktop kann daher noch nicht gesprochen werden. Trotzdem steckt in diesem Projekt extrem viel Know-how, und die Idee, diese Funktionalität (momentan noch frei) zur Verfügung zu stellen, ist eine Bereicherung für das Web.

2.2.2 MyInternetDesktop.com

 http://www.myinternetdesktop.com/

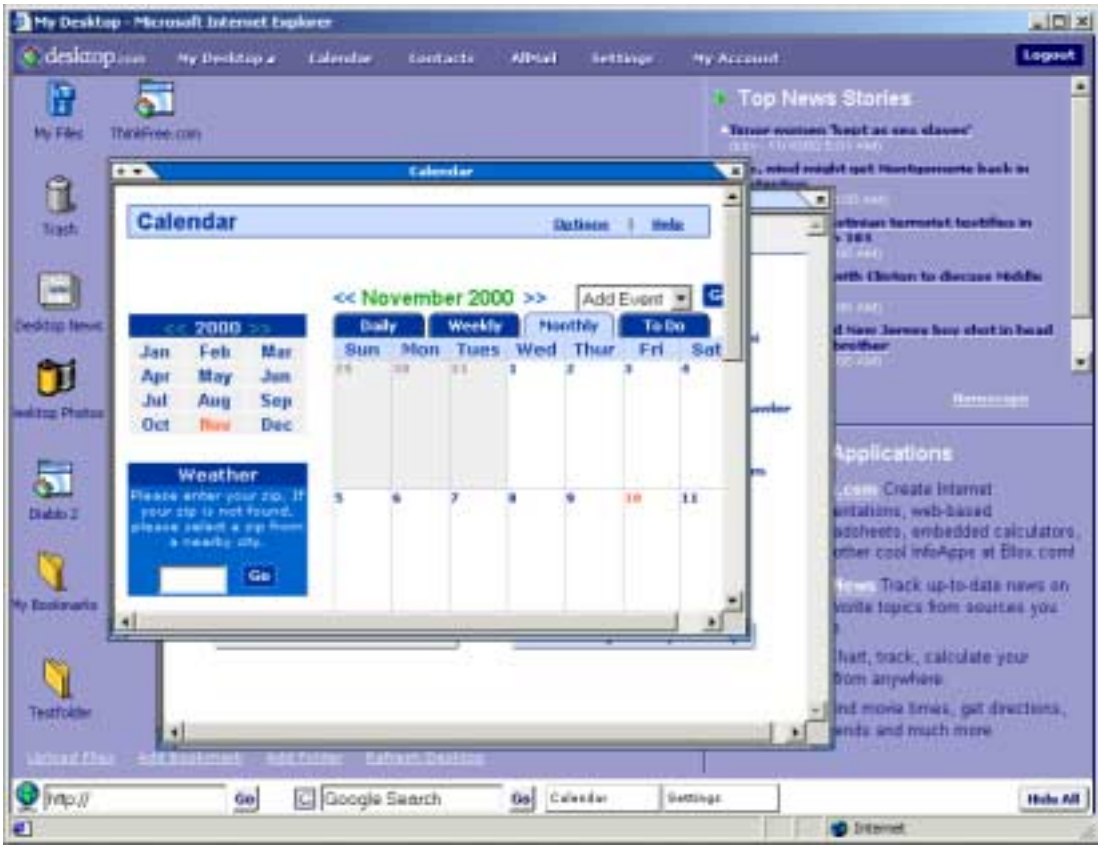

Abbildung 2.1: Screenshot von Desktop.com (Internet Explorer).

Im Gegensatz zu Desktop.com bietet MyInternetDesktop.com keine Möglichkeit, eigene Applikationen hinzuzufügen. MID ist ein klassischer Web-Desktop-Dienst, der ToDo-Listen, Email-Verwaltung, Fotoalbum und anderes mehr zur Verfügung stellt. Das anfangs versuchte Imitieren einer Windows-ähnlichen Oberfläche (die jedoch nur durch HTML-Tables aufgebaut worden ist), wurde später von einem eigenständigen Design abgelöst (ebenfalls Table-basierend).

Technisch gesehen ist MID eine komplexe, aber wenig aufregende Web-Applikation. Es wird mit Cookies und hidden Form-Elements gearbeitet, die Session-ID wird aus dem Datum erzeugt und Server-seitig eingeparst. Interessant sind die Möglichkeiten des virtuellen Speicherplatzes – er erlaubt den Download von Dokumenten via URL's auf den Speicher im MID sowie das Sharing von einzelnen Dokumenten innerhalb von Arbeitsgruppen.

Marcelo Lewin informierte am 7. November 2000 in einer Mail alle Benutzer des Systems, dass MyInternetDesktop.com am 16. November seine Tore schließen würde. Ab nun würden die Technologien des Internet Appliance Networks nicht mehr direkt dem Endnutzer zur Verfügung stehen.

Abbildung 2.2: Screenshot der alten Version von MyInternetDesktop.com.

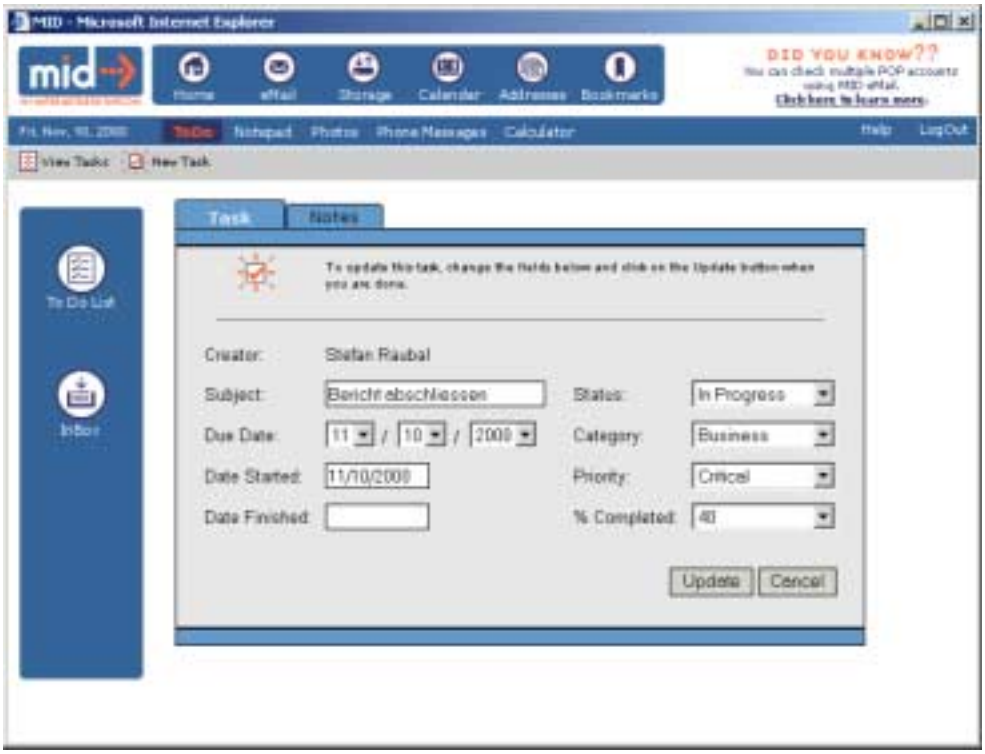

Abbildung 2.3: Screenshot der neuen Version von MyInternetDesktop.com.

Kapitel 3

Techniken

"Technology is like fish.
The longer it stays on the shelf,
the less desirable it becomes."
Andrew Heller, IBM

3.1 Projekte Java-basierender Frameworks

Eine Vielzahl von Projekten beschäftigt sich seit etlichen Jahren mit dem Bereich Web-Applikationen. Dabei geht es meist darum, ein Server-System aufzubauen, mit dem möglichst effizient eine Web-Oberfläche mit der darunterliegenden Funktionalität (meist eine Form von Datenbank-Integration) verknüpft werden kann.

Die meisten der im Folgenden aufgelisteten Frameworks für Web-Applikationen basieren auf der Kombination von Java und XML. Warum findet man diese Verbindung so oft in diesem Bereich?

Die Antwort lautet: "Java und XML bilden ein gut miteinander harmonisierendes Paar." So zitiert [31, S.1] David Brownell: "Java is cross-platform code and XML is cross-platform data". Doch nicht nur aus diesem Grund besteht eine solch große Affinität zwischen den beiden Techniken – weitere Zusammenhänge ergeben sich aus dem Zeitpunkt, an dem Java und XML entwickelt bzw. populär wurden und aus der Tatsache, dass Firmen wie Sun oder IBM sehr schnell stabile Parser-Klassen zur Verfügung stellten.

Das Parsen eines XML-Dokuments benötigt allerdings eine umfangreichere Klassenbibliothek und ist nicht zuletzt deshalb bisher rein am Server zu finden. In einiger Zeit ist zu erwarten, dass Web-Browser XML und XSL beherrschen und diese Fähigkeit auch anderen API's zur Verfügung stellen (Ansätze dazu sind im Internet Explorer 5 und Mozilla 6 schon zu finden). Bis dahin kann bei kleineren Projekten u.U. ein JavaScript XML-Parser behilflich sein [15, 20].

3.1.1 The Apache Cocoon Project

 http://xml.apache.org/cocoon/index.html

Beschreibung

Das Cocoon-Projekt hat das Ziel, das Publizieren von Daten in verschiedenen Formaten und auf unterschiedliche Plattformen zu erleichtern. Ausgerichtet ist das Projekt auf den Web-Bereich – so existieren "Konverter" (XSL-Transformationen), um XML-Daten als PDF-Dokument oder für WAP-Browser lesbar publizieren zu können. Wie bei Apache üblich wird versucht, den Standards des W3C zu entsprechen bzw. auf ihnen aufzubauen.

Das Konzept beruht auf dem Umstand, dass Inhalt, Aussehen und Logik (Funktionalität) eines Dokuments meist von verschiedenen Quellen stammen[1]. Cocoon schafft eine Basis, diese drei Schichten bestmöglich zu trennen und unabhängig voneinander zu editieren und zu verwalten.

Verwendete Technologien

Das Cocoom-Projekt basiert auf Servlets und nutzt XSLT, XSP, XSL:FO (PDF-Rendering) und WAP-Rendering-Funktionalität.

3.1.2 The Apache Xang Project

 http://xml.apache.org/xang/index.html

Beschreibung

Apache Xang stellt ebenfalls eine Architektur für Web-Applikationen zur Verfuegung, die eine saubere Trennung von Daten, Logik und Repräsentation vorsieht. Es basiert auf Standards wie HTTP, XML, XSL, DOM und ECMA-Script. Eine Xang-Applikation definiert sich über ein XAP-File, das verschiedene Datenquellen vereint,URL-adressierbar macht und zusätzliche Methoden definiert, wie die Daten angesprochen werden können. Letzteres kann durch HTTP-Requests, über einen Web-Browser oder ein einfaches Client-seitiges API geschehen.

Verwendete Technologien

Als Server-seitige Programmier-Plattform werden Servlets verwendet. Aufgesetzt wird auf Xerces und Xalan, sowie einem frei verfügbaren ECMA-Script Interpreter.

3.1.3 Turbine

 http://java.apache.org/turbine/

[1] Vergleichbar mit dem Model/View/Controler-Paradigma für Applikationen.

Beschreibung

Turbine wirbt damit, erfahrenen Java-Entwicklern das schnelle Erstellen sicherheitskritischer Web-Applicationen zu ermöglichen. Das Framework stellt Methoden (Servlets) zum Parameter-Parsen, Datenbank Connection Pools, Job Scheduling, Caching, Integration anderer Techniken (wie WebMacro, Velocity, Castor, ...), JavaMail-Schnittstellen und vieles mehr zur Verfügung, was für Web-Applikationen immer wieder neu erfunden wird.

Verwendete Technologien

Als 100%-iges Java-Projekt basiert Turbine auf der Servlet-Technologie. Das heißt, um Turbine benutzen zu können, braucht man lediglich einen Servlet-fähigen Web-Server bzw. eine Servlet-Engine wie Tomcat, JServ oder JRun.

3.1.4 Jetspeed

 http://java.apache.org/jetspeed/site/overview.html

Beschreibung

Jetspeed ist die Open Source Implementierung eines Informations-Portals. Dabei wird Information aus verschiedenen Quellen (von XML über SMTP bis zu neueren Protokollen wie iCalendar) gruppiert und visualisiert. Dabei kann der Benutzer selbständig Datenquellen hinzufügen und deren Darstellung beeinflussen.

Verwendete Technologien

Jetspeed setzt auf dem Portlet API und verschiedenen Basis-Frameworks wie Apache Turbine, Cocoon oder Castor (Marshalling von XML Daten in Java Objecte) auf.

3.1.5 Struts

 http://jakarta.apache.org/struts/

Beschreibung

Ein kleineres Framework für Web-Applikationen im Model-View-Controller Design-Pattern ist Struts, das Teil des Jakarta-Projekts ist.

Das Herz von Struts ist ein Kontroll-Servlet, welches Requests an die betreffenden Action-Klassen verteilt. Inkludiert ist weiters eine JSP-Tag Bibliothek und Utility-Klassen, um XML-Daten parsen zu können.

Verwendete Technologien

Da Struts im Rahmen des Jakarta-Projekts entwickelt wird, baut es wie die Servlet Engine auf dem Servlet 2.2 API und den JavaServer Pages 1.1 auf.

3.1.6 XMLC

 http://xmlc.enhydra.org/

Beschreibung

Als Werkzeug für den Enhydra-Applikationsserver von Lutris entstanden, ist XMLC mittlerweile zu einem eigenständigen – wenn auch immer noch von Lutris geleiteten – Open Source Projekt geworden. Hinter XMLC steckt die Idee, HTML- bzw. XML-Dokumente in Java Klassen (entsprechend dem DOM API) zu compilieren, welche dann via Servlet-Umgebung ihren Inhalt manipulieren lassen und in Client-verarbeitbarer Form als Antwort auf einen Request zurücksenden.

Die Trennung zwischen Designer und Programmierer ergibt sich dadurch automatisch. Ersterer entwirft die Web-Seiten wie bisher in seinem Editor – alles, worauf er zu achten hat ist, die dynamischen Objekte mit eindeutigen ID's zu versehen. Diese ID's dienen dem Programmierer als Einstiegspunkte in die Java Klassen und ermöglichen ihm DOM-Manipulationen.

Dies alles ist für den Server-seitigen Einsatz ausgelegt – ein kleiner Prototyp zeigte aber, dass der Einsatz am Client ebenso möglich wäre. Problematisch ist dabei v.a. die Größe der notwendigen Klassen und Interfaces sowie die Nutzung von LiveConnect.

Basierend auf XMLC bildete sich im Frühjahr 2000 eine Initiative namens "Rocks", die XMLC mit einem Model-View-Controler Framework und einer GUI-Komponenten Bibliothek versehen will. Nähere Informationen dazu sowie einen Vergleich mit Turbine und Struts finden sich ebenfalls auf der Homepage von XMLC.

Verwendete Technologien

XMLC ist in Java realisiert und bildet darin HTML/XML dem W3C DOM gemäß ab.

3.1.7 WebMacro

 http://www.webmacro.org/

Beschreibung

Im Gegensatz zu den vorigen Projekten wurde bei WebMacro eine eigene Skript-Sprache eingeführt und nicht auf JSP oder XML gesetzt. WebMacro bezeichnet sich als "HTML template engine and back end servlet development framework". Web-Designer arbeiten mit HTML und WebMacro-Skripts, Programmierer rein in Java. Dabei wurde vor allem darauf

geachtet, die Syntax der Template-Sprache intuitiv und simpel zu halten, da der typische Anwender keine Programmier- oder XML-Erfahrung besitzt.

Ein kleines Beispiel, der Homepage des Projekts entnommen, soll das verdeutlichen:

```
#set ContentType = "text/html"
<html>
  <head><title>$Customer.Name</title></head>
  <body bgcolor='white'>

    <h1>$Customer.Name: History<h1>
    Here's a list of your orders since $Customer.Orders.StartDate:

    <table width='70%'>
      <th>
        <td>Order Date</td>
        <td>Item Requested</td>
        <td>Number of Units</td>
      </th>

      #foreach $order in $Customer.Orders {
        <tr>
          <td>$order.Date</td>
          <td>$order.Item.Name</td>
          <td>$order.Number</td>
        </tr>
      }
    </table>
  </body>
</html>
```

Die Kommandos und Variablenreferenzen werden von Web-Editoren als Text interpretiert und rufen keine Stilüberprüfungs-Fehler hervor (sie sind nicht Tag-basierend wie bei JSP). Die foreach-Schleife bildet ein einfaches, aber mächtiges Konstrukt, um Datenmengen zu iterieren. Der Zugriff auf Java-Objekte erfolgt über einen Hashtable-Mechanismus, d.h. alles was getan werden muss, um ein Objekt in einer HTML-Seite ansprechen zu können, ist, dieses in eine Hastable einzufügen.

Ein sehr interessanter Artikel übere JSP contra Template-Sprachen wie WebMacro (samt Zusammenfassung der nachfolgenden Diskussion) ist bei [16, 17] zu finden.

Während des Zeitraums der Erstellung dieser Arbeit wurde WebMacro in das Apache Velocity Projekt integriert (nachdem zuerst der Versuch unternommen wurde, es dort zu reimplementieren). Damit wurde diese bemerkenswerte Initiative in ein größeres Konzept eingebettet und ihm die Chance gegeben, noch weitere Kreise zu ziehen.

Zwei weitere Projekte, die ein ähnliches Konzept verfolgen, seien hier nur namentlich genannt:

- FreeMarker http://freemarker.sourceforge.net/
- oTembo http://www.meangene.com/otembo/

Verwendete Technologien

WebMacro basiert auf Servlets und der eigenen Template Skript-Sprache.

3.2 Die Client-Seite

Der Vielzahl von Techniken, den Inhalt einer Web-Seite am Server zu generieren, steht eine eher geringe Anzahl von Möglichkeiten gegenüber, dynamisch Daten im Browser zu visualisieren. Drei grundlegend verschiedene Ansätze gibt es hierzu:

- Dynamisches HTML
- Java Applet
- Plug-In

Die Nutzung von Dynamic HTML hat den Flair einer handwerklichen Tätigkeit – bei genügendem Geschick ist es möglich, praktisch alles zu realisieren. Leider verleitet die Nutzung von Skriptsprachen leicht dazu, jedlichen guten Programmierstil früher oder später zu verlieren. Ein großes Projekt einer Web-Applikation muss daher sehr gut geplant und kontrolliert werden, damit das Endprodukt kein "Hack" (oder, um im oberen Bild zu sprechen, kein "Pfusch") wird.

Java Applets nötigen dagegen eher zum Einhalten strikter Codier-Standards. Weiters ist die Idee der Design Patterns in Objekt Orientierten Programmiersprachen mehr gereift (und besser umsetzbar) als in Sprachen wie JavaScript[2]. Problematisch wird es hier beim GUI (Graphical User Interface). Trotz der Einführung des Swing Window Toolkits ist Java auf diesem Bereich nicht so flexibel wie es DHTML sein kann. Dazu kommt, dass nicht jeder Netz-Benutzer Applets aktiviert hat – die immer wieder auftauchenden Warnungen bezüglich Sicherheitslücken tun das ihre zu dieser Skepsis. Und im Gegensatz zu JavaScript (bei dessen Implementation in den Browsern ebenso Security-Probleme bekannt sind) können bis jetzt Java Applets ohne große Einbußen ausgeschaltet werden.

Die dritte Variante – Plug-Ins wie Shockwave zu verwenden – ermöglicht das einfache Erstellen Multimedia-lastiger Seiten. Allerdings ist es auch hier wieder notwendig, auf Skripte zurückzugreifen. Zusätzlich ergibt sich der Nachteil, dass das Plug-In installiert sein muss. Diese sind teils recht groß und schrecken dadurch den Benutzer vor dem Download ab (wenn dieser nur eine Modem-Verbindung besitzt).

[2]Ein Ansatz, Design Patterns für Web-Programmierung zu finden und zu katalogisieren, ist bei [32] zu finden.

3.2.1 Beispiel: Button-Realisierung

Am konkreten Beispiel eines komplexeren Buttons (siehe Abbildung 3.1) seien die unterschiedlichen Ansätze miteinander verglichen.

Abbildung 3.1: Beispiel typischer eLS-Buttons in verschiedenen Zuständen.

Folgende Anforderungen sollen gelten:

1. Die Beschriftung sei Text-Ressource und nicht Teil einer Grafik – somit kann er leicht ausgetauscht werden.
2. Ein Mouse-Over Effekt sei realisiert.
3. Der Button sei ein- und ausschaltbar und habe dabei seine Darstellung zu ändern. (So ist im Beispiel der oberste Button deaktiviert – und zeigt zugleich an, dass man sich im Bereich "Courses" befindet.)

Die Realisierung eines solchen Buttons in JavaScript ist nichts Ungewöhnliches und wird als Referenz gegenüber einer (hypothetischen) Java-Implementierung herangezogen[3]. Beim Einsatz von Java Applets gibt es zwei mögliche Ansätze zu unterscheiden:

- Ein eigenes Applet für jeden einzelnen Button.
- Ein großes Navigations-Applet mit allen Buttons.

Die erste Variante scheint flexibler zu sein, bei einer Parameter-gesteuerten Implementierung der zweiten Variante könnte jedoch ebenfalls ein Eingreifen in den Java-Code vermieden werden, wenn Änderungen an den Texten, den URL's oder der Reihenfolge der Buttons vorgenommen werden müssen.

Betrachten wir nun die Vorteile von Java gegenüber Dynamic HTML:

- Strukturiertes, auf Klassen basierendes Objekt Orientiertes Programmieren möglich.
- Unabhängig vom DOM des Browsers.
- Nutzung schon vorhandener Button-Klassen möglich.

Dem gegenüber stehen die Nachteile:

[3] eLS realisiert seine Buttons in JavaScript.

- Java Applets müssen im Browser aktiviert sein. (Nicht zu unterschätzen ist auch die Zahl der Browser, die kaum oder gar nicht Applets unterstützen.)
- Abhängigkeit von der installierten Java Virtual Machine – so ist z.B. für Swing bei den meisten Browsern ein Java Plug-In notwendig.
- Für erweiterte Funktionalität ist u.U. LiveConnect notwendig.
- Darstellung der Hintergrundfarbe im Applet nicht immer gleich wie in HTML.
- Veränderung des Mauszeigers, wenn er über dem Button steht, kann nicht unter allen Browsern realisiert werden.

Das Abwiegen der Vor- und Nachteile möge unter unterschiedlichen Gesichtspunkten zu verschiedenen Entscheidungen führen – der Autor würde im allgemeinen zu einer Implementierung mittels dHTML raten.

3.2.2 Vom Page-Request zur fertigen Seite

Bis eine Web-Seite im Browser endgültig dargestellt wird, ist eine Reihe von Schritten zu bewältigen, die meist von der Browser-Software übernommen wird. Sollen jedoch eine Seite und deren Objekte (Ressourcen) eigenständig angefordert, geparst und dargestellt werden, müssen diese Mechanismen selbst implementiert werden. Fünf Abschnitte, die hier als Module bezeichnet seien, liegen auf dem Weg vom Request zur fertigen Seite:

1. Retrieval-Modul
2. Parsing-Modul
3. Merging-Modul
4. Rendering-Modul
5. Caching-Modul

Retrieval-Modul

Dieses Modul holt von der Datenquelle die notwendigen Daten. Diese Quelle ist ein Server im Online-, das Filesystem im Offline-Fall. D.h. jeder Link ruft eine Aktion in diesem Modul auf und fordert dabei ein Objekt mit einer gewissen ID an (Pfeil 1 in Abbildung 3.2). Das Retrieval-Modul besorgt sich das entsprechende Objekt (2,3) und gibt es an das Parsing-Modul weiter (4).

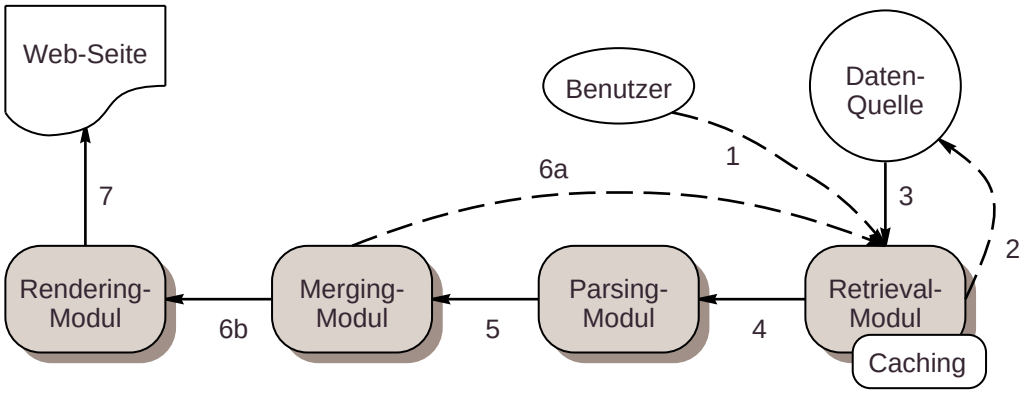

Abbildung 3.2: Der Weg vom Request zur fertigen Seite.

Parsing-Modul

Dieses Modul ist nur dann notwendig, wenn als Dokumentenbeschreibungssprache eine Sprache gewählt wird, die erst in HTML (oder eine andere, dem Rendering-Modul bekannte Grammatik) transformiert werden soll (hier bietet sich XML mit XSLT an). Das Problem hierbei: Das Parsen muss sehr performant sein und der Download der Parsing-Klassen bzw. Skripts darf nicht zu lange dauern. Der Output des Parsing-Moduls wird an das Merging-Modul weitergereicht (5).

Merging-Modul

Das Merging-Modul ist für das Einmischen weiterer Ressourcen zuständig. Bei der Hyperwave eLS z.B. gibt es die Möglichkeit, Annotationen direkt im Kursinhalt einzufügen. Aber auch andere, dynamisch auf dem Client erzeugte Daten können an dieser Stelle in das Dokument integriert werden. Dieses Modul wendet sich an das Retrieval-Modul, wenn für weitere Objekte Requests notwendig sind (6a) und bedient das Rendering-Modul mit den endgültigen Daten (6b).

Rendering-Modul

Das Rendering-Modul schließlich sorgt für die Darstellung des Dokuments im Browser (7). Hier wird entweder HTML-Content in das Document-Objekt eines Browser-Fensters geschrieben, ein Applet für die Darstellung der Objekte vorbereitet oder ein Plug-In angesteuert.

Caching-Modul

Die Position dieses Abschnitts in der Kette zwischen Request und Seitenaufbau ist beliebig wählbar. Es ist auch leicht vorstellbar, den Cache in einen Teil wie das Retrieval-Modul

zu integrieren (wie es in Abbildung 3.2 angenommen wurde).

3.2.3 Ressourcen-Verwaltung

Um Anforderungen wie Internationalization, Skalierbarkeit und Wartbarkeit gerecht zu werden, ist bei größeren Applikationen unbedingt ein gutes Ressourcen-Konzept notwendig. Dies trifft ganz besonders auf Web-Applikationen zu, da deren Ressourcen meist sehr vielfältig sind und über verschiedene Quellen verteilt gespeichert werden.

Folgendes ist in diesem Zusammenang u.a. zu beachten:

Hierarchische Struktur Eine Klassenhierarchie der Ressourcen vereinfacht das Design und erhöht die Wartbarkeit.

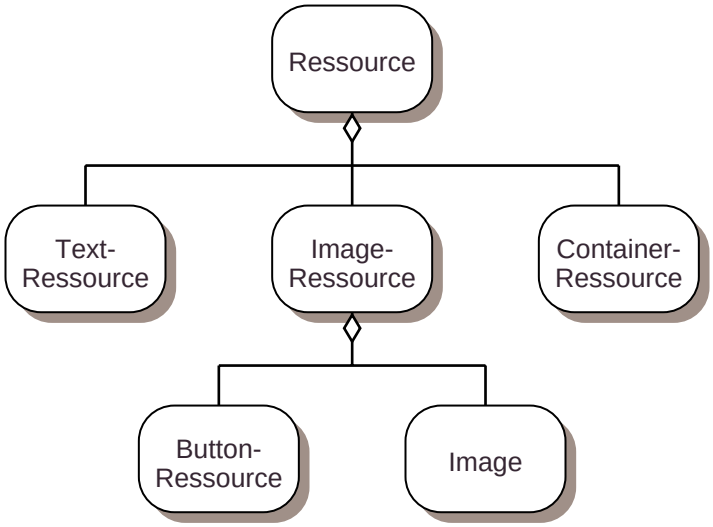

Abbildung 3.3: Beispiel für eine einfache Klassenhierarchie.

Ressourcen Repository Eine eindeutige und gut erweiterbare Lösung bezüglich der Orte, an denen Ressourcen gespeichert werden, ist für die Eindeutigkeit und Customizierbarkeit unumgänglich. Das Servlet API 2.2 hilft dabei schon ein wenig, denn es definiert selbst, wie die Verzeichnis-Struktur einer Web-Applikation aufzubauen ist.

Scoping-Konzept Die Darstellung einer Ressource wird vom sg. Scope bestimmt, in dem sie sich befindet – d.h. je nach eingestellter Sprache, Bandbreite der Verbindung, Rolle im System oder weiteren Modifikatoren mag ein Objekt anders darzustellen sein. Hierfür ist es wichtig, genau zu definieren, welche Attribute den Scope eines Dokuments bestimmen und welche Priorität sie haben (d.h. "welche Einstellung überschreibt welche?"). Diese Attribute müssen am Client bekannt sein und dort Einfluss auf die Erstellung des endgültigen Dokuments haben.

3.2.4 Vorhandene dHTML Crossbrowser-Ansätze

Mike Halls Bibliothek mit Crossbrowser-Funktionen

 http://www.brainjar.com/dhtml/dhtmllib.html

Diese Bibliothek bietet in einem js-File von ca 12 KB Basisfunktionalität zur Arbeit mit absolut positionierten Elementen in Netscape und Internet Explorer (jeweils Versionen 4.0+). Dies inkludiert Methoden zum Editieren von Position, Sichtbarkeit, Clipping, Hintergrund und Scrolling.

Erzeugt werden müssen die Elemente via <DIV>-Tag, mittels getLayer()-Funktion bekommt man einen Handle, der weitere Modifikationen erlaubt.

X-Objects von Shelley Powers

 http://www.yasd.com/dynatech/xobjsintro.htm

Die X-Objects Library geht einen ähnlichen Weg. Auch hier müssen die dynamischen Elemente als HTML DIV's existieren, bevor über eine create_objects()-Methode aus all diesen Objekten X-Objects erstellt werden (je nach Browser mittels new ns_object() oder new dom_object() oder new ie_object(). Wie man sieht wird dabei bereits zwischen Netscape 4, Internet Explorer und Netscape 6 unterschieden.

Dort, wo es möglich ist, werden Gemeinsamkeiten genutzt, so dass die Definiton der Methoden eines X-Objects folgendermaßen aussieht:

```
function ns_object(obj) {
   this.objMoveAbsolute = domMoveAbsolute;
   this.objMoveRelative = domMoveRelative;
   this.objReplaceHTML = nsParamReplaceHTML;
   this.objReplaceText = nsReplaceText;
   this.objGetVisibility = nsVisibility;
   //... weitere Methoden-Definitionen weggelassen
}

function dom_object(obj) {
   this.objMoveAbsolute = domMoveAbsolute;
   this.objMoveRelative = domMoveRelative;
   this.objReplaceHTML = domParamReplaceHTML;
   this.objReplaceText = domReplaceText;
   this.objGetVisibility = domGetVisibility;
   //... weitere Methoden-Definitionen weggelassen
}

function ie_object(obj) {
```

```
    this.objMoveAbsolute = domMoveAbsolute;
    this.objMoveRelative = domMoveRelative;
    this.objReplaceHTML = ieParamReplaceHTML;
    this.objReplaceText = domReplaceText;
    this.objGetVisibility = domGetVisibility;
    //... weitere Methoden-Definitionen weggelassen
}
```

Der Zugriff erfolgt über ein Array von X-Objetcs, z.B. beim Verschieben eines Elements namens "aDIV" über theobjs["aDIV"].objMoveRelative(10, 20).

Die gebotene Funktionalität beinhaltet alle notwendigen Positionierungs-, Skalierungs-, Sichtbarkeits- und Clipping-Einstellungen, die Größe der Bibliothek überschreitet auch kaum die 10 KB-Grenze.

Crossbrowser-Objekte von Dan Steinman bzw. DynAPI 2

 http://www.dansteinman.com/dynduo/

Das mittlerweile von 13 Entwicklern als Open Source Projekt unter der Gnu Public License vorangetriebene Produkt hat – wie der Name schon andeutet – einen anderen Ansatz. DynAPI 2 bietet 180 KB Dynamic HTML im großen Stil. Aufbauend auf Kern-Objekten, die als Wrapper für window.document und alle dynamischen Elemente dienen, und Erweiterungen wie Event-Objekte oder Drag-and-Drop Unterstützung stehen ausprogrammierte Widgets wie Scrollbars, Trees und Dialoge zur Verfügung.

Dabei wird der Mechanismus des Class-Loadings eingesetzt, um dem Entwickler die Möglichkeit zu bieten, aus dem reichen Sortiment die für ihn notwendigen Klassen zu importieren.

Ein einfaches Beispiel soll die Art, mit DynAPI zu arbeiten, verdeutlichen:

```
<html>
  <head><title>CoreLib example</title>
  </head>

  <script language="Javascript" src="../js/dynapi.js"></script>

  <script language="Javascript">
    DynAPI.setLibraryPath('../js/lib2.0/')
    DynAPI.include('core.api.*')

    DynAPI.onLoad=function() {
      myLayer = new DynLayer()

      myLayer.setSize(100,100)
```

```
      myLayer.setBgColor('#c0c0c0')
      myLayer.moveTo(100,100)

      this.document.addChild(myLayer)
    }
  </script>

  <body></body>
</html>
```

Diese Bibliothek erlaubt, dynamisch Objekte zu erstellen, ohne dass sich der Entwickler um DIV's oder ähnliches kümmern muss, ebenso wird ein Browser-unabhängiges Event-Handling Modell angeboten. Noch nicht unterstützt wurde jedoch zum Zeitpunkt des Schreibens dieser Zeilen das DOM des neuen Netscape 6.

Rund um dieses API gibt es ein Diskussionsforum, Mailinglisten sowie Tutorials und Dokumentationen.

Scott Issacs DHTMLLib 2.1

 http://www.insidedhtml.com/dhtmllib/page1.asp

Einen ganz anderen Weg geht DHTMLLib. Hier wird in einem 15 KB Skript versucht, die Browser-Unterschiede auszugleichen, um in Netscape in der vom Internet Explorer vertrauten Weise arbeiten zu können. So ergänzt eine setup()-Methode das DOM durch Objekt-Referenzen, damit via document.all und obj.style auf die positionierten Elemente bzw. deren Style-Objekt zugegriffen werden kann.

Zusätzlich werden ein onScroll-Event simuliert, Unterschiede wie visibility="visible" und visibility="show" ausgeglichen oder ein Event-Bubbling-Konzept unter Netscape nachgebildet.

Da das Projekt im Dezember 1999 das letzte Mal überarbeitet wurde, gibt es natürlich noch keine Unterstützung für Netscape 6.

Resümee

Je nach Anwendungsbereich und Programmiervorliebe bieten sich verschiedene Lösungen an, die jeweils ihre Vor- und Nachteile besitzen:

Komplexes dHTML-Projekt: Für aufwendiges Dynamic HTML scheint DynAPI wie geschaffen – es erlaubt die volle Kontrolle über das DOM und ein wirklich Browser-unabhängiges Erstellen dynamischer Elemente. Darüberhinaus werden für viele Anwendungsfälle bereits Widgets angeboten. Allerdings sollte eine solche Web-Seite bzw. Web-Applikation ausführlichen Gebrauch von den Funktionen der DynAPI machen, denn sonst wäre deren Einsatz ein klassischer "Overkill".

Vorhandene Erfahrung mit IE-DOM: In diesem Fall bietet sich Scott Issacs DHTML-Lib 2.1 an, denn damit ist kein Einarbeiten in andere DOM-Modelle notwendig. Allerdings kann diese "Bibliothek" nicht 100%-ig das Verhalten des Internet Explorers simulieren – in manchen Fällen ist dann ein fundiertes Wissen über das Netscape-DOM notwendig. Ein weiteres Manko ist das Fehlen der Unterstützung für Netscape 6.

Netscape 6 Unterstützung notwendig: Die einzige oben angeführte Bibliothek, die diese Anforderung erfüllt, ist jene von Shelley Powers. Abgesehen davon ist dieses Projekt ein gut überlegter, schlanker Ansatz, der ohne viel Aufwand auf weitere Browser bzw. DOM's modifizierbar ist (das zeigt auch die Tatsache, dass er bereits das W3C-DOM unterstützt).

Kapitel 4

Browser-Probleme

"Das Geheimnis des Erfolgs?
Anders sein als die anderen."
Woody Allen, eigtl. Allen Stewart Konigsberg,
amerik. Regisseur, Schauspieler, Gagschreiber und Schriftsteller

4.1 DOM-Inkompatibilitäten

In den letzten Jahren wandelte sich das Denken der Web-Entwickler von "cross-browser" hin zum Begriff "cross-DOM"-Entwicklung. Dies wurde verstärkt durch das Erscheinen der ersten Releases von Netscape 6 (Mozilla), welches erstmals das Document Object Model des W3C unterstützt, und zwar Level 1 und teilweise auch Level 2 (letztere Spezifikation ist selbst noch zum Zeitpunkt des Schreibens dieser Zeilen in Arbeit[1]). Zusätzlich dazu entschieden sich die Entwickler, Altlasten von Netscape 4 zu verwerfen, die weit abseits von jeglichem Standard lagen, wie z.B. das Layer-Objekt.

Dieser mutige Schritt macht viele Seiten, die dHTML verwenden, für den Mozilla-Browser untauglich – v.a. dann, wenn die Browser-Unterscheidung auf dem Namen "Netscape" und der Version 4 oder größer basiert. Durch sein Document Object Model ist der neue Browser dem Internet Explorer 5 von Microsoft viel ähnlicher geworden . Leider gibt es erst wenige Bibliotheken oder Referenzen, die beim Unterstützen des neuen Modells helfen – trotzdem ist die Entwicklergemeinde dem konsequenten Ansatz gegenüber positiv eingestellt.

Ein weiteres Problem birgt die Tatsache, dass Microsofts Rendering Engine unter Macintosh (Tasman) in der Version 5 ebenfalls den Versuch unternahm, näher dem W3C-DOM zu kommen, diese Spezifikation aber nicht komplett erfüllt[2]. Damit muss beim Entwickeln auch das Betriebssystem beachtet werden, nicht nur Typ und Version des Browsers.

Für den Entwickler bedeutet das – will er zu allen gängigen Browsern kompatibel bleiben, dass er immer mehr Fälle zu unterscheiden hat. Da dieser Zustand wohl noch einige

[1] Der aktuelle Status und genaue Details zum W3C DOM finden sich unter [14]
[2] Zum Thema "Tasman" finden sich im Netz extrem unterschiedliche Meinungen.

Zeit anhalten wird, ist zu empfehlen, bei allen DOM-Zugriffen (via Javascript bzw. Java) eine Zwischenschicht zu verwenden, die je nach Browser die richtige Funktion ausführt. An dieser zentralen Stelle können dann neue Browsern bzw. Browserversionen gehandhabt werden. Interessante Artikel dazu sind bei [21] und [23] zu finden.

4.1.1 Vergleich der Objektmodelle

Der folgende Abschnitt soll nun einen Überblick über die Gemeinsamkeiten und Unterschiede der Objektmodelle der gängigen Browser bieten. Das Wissen um diese Differenzen ist für jeden Entwickler von Client-seitigem JavaScript von großer Bedeutung. Denn obwohl sich der Teil der Sprache, der mit "Core JavaScript" betitelt ist, in den letzten Jahren kaum verändert hat, gab es, was das Objektmodell betrifft, unterschiedlichste Entwicklungen.

Das Objektmodell ist jener Teil, den die Laufzeitumgebung (d.h. in den meisten Fällen der Browser) beiträgt. Abbildung 4.1 zeigt eine einfaches Illustration eines solchen Objektmodells.

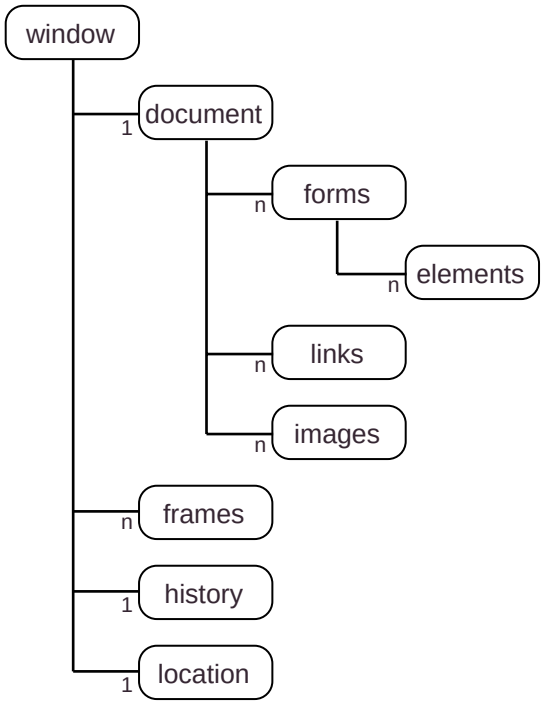

Abbildung 4.1: Vereinfachte Darstellung des Basis-HTML-DOM.

Diese Strukturen sind historisch gewachsen und ihre Entwicklung wurde stark durch die jeweiligen gängigen Browser vorangetrieben. So war es Netscape 3, das erstmals Manipulationen des Image-Objekts erlaubte. Schon damals begann das Dilemma der Programmierer, überprüfen zu müssen, ob das DOM des jeweiligen Browsers das gewünschte Objekt anbietet – Abfragen wie `if (document.images)` waren geboren.

Das Navigator 4 Objektmodell

Der Navigator 4 war es 1997, der durch die Einführung des Layer-Objekts erstmals wirkliches dHTML erlaubte. Layers ermöglichen es, Elemente im Fenster zu bewegen, sie überlappen zu lassen oder ihre Sichtbarkeit zu verändern.

Netscape verstand einen Layer als ein Objekt, das ein eigenes Dokument besitzt, welches über eine URL oder – häufiger – über document.writeln seinen Inhalt zugewiesen bekommt. Somit ergab sich ein DOM wie in Abbildung 4.2.

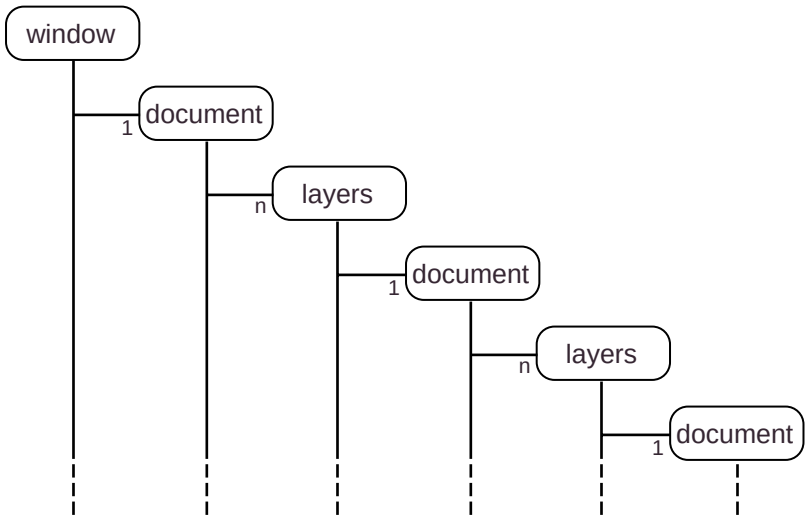

Abbildung 4.2: Vereinfachte Darstellung des Netscape 4 DOM.

Diese Sichtweise erlaubte auch – wie es aus Abbildung 4.2 hervorgeht – die Verschachtelung von Layern. Nehmen wir an, ein Image wird von einem DIV-Tag mit absoluter Positionierung via Cascading Style Sheets wie folgt umgeben[3]:

```
1 <body>
2   <div id='theDIV' style='position:absolute'>
3     <img name='theImage' src='nice_pic.gif'>
4   </div>
5 </body>
```

Dieses Image-Objekt ist nun nicht mehr als window.document.images['theImage'], sondern als window.document.layers['theDIV'].document.images['theImage'] ansprechbar – und den Unterschied macht allein position:absolute.

[3]Diese wohl am häufigsten verwendete Methode bewirkt die implizite Erzeugung eines Layer-Objekts.

Das Internet Explorer 4 Objektmodell

Ab der etwas nach Netscape 4 erschienen Version 4 des Internet Explorers unterstützt auch Microsofts Browser dHTML. Jedoch wurde hier nicht der Weg eines Layer-Objekts gegangen, sondern praktisch das gesamte DOM um Dynamic HTML Fähigkeiten erweitert. Dazu wurde das Style-Objekt als Property fast aller Objekte eingeführt. Dies ermöglicht neben der Positionierung und dem Ändern der Größe auch gleich das Setzen von Eigenschaften wie border-width oder background-color.

Der Inhalt eines dHTML-Elements ist im Internet Explorer sehr elegant über die innerHTML Property zu setzen – und auch auszulesen. Letzteres ist im Netscape Navigator generell nicht möglich.

Das Referenzieren der Objekte erfolgt nicht in der strengen Hierarchie von Netscapes DOM, sondern hier werden alle Objekte als gleichwertig nebeneinander betrachtet. Dazu wurde die Collection document.all eingeführt, die alle mit einer ID versehenen Elemente aufnimmt. Somit wird das Bild aus obigem Beispiel über window.document.all['theImage'] angesprochen. In Abbildung 4.3 ist das sich aus diesem konkreten Fall ergebende DOM visualisiert.

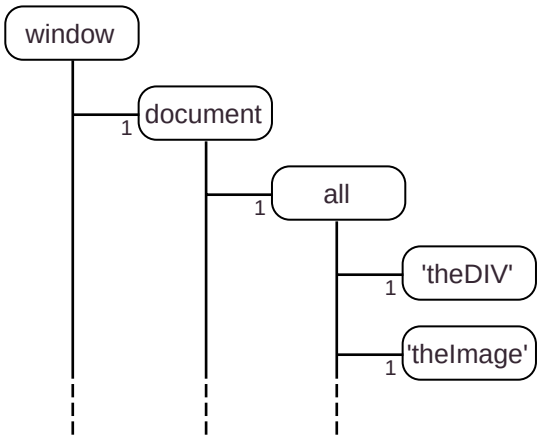

Abbildung 4.3: Vereinfachte Darstellung des Internet Explorer 4 DOM .

Das W3C Objektmodell

Das W3C Konsortium erarbeitete unter anderem in den letzten Jahren auch einen Vorschlag für den künftigen Standard eines Document Object Models. Den jeweiligen aktuellen Status findet man, wie schon oben erwähnt, unter [14].

Das entwickelte Objektmodell lehnt sich stark an jenes des Internet Explorers an und wurde auch von diesem in den Versionen 5 und 5.5 zum Teil umgesetzt – zusätzlich zu den beibehaltenen Mechanismen aus Version 4. Netscape 6 unterstützt dieses Modell von allen momentan gängigen Browsern am besten und beschränkt sich auch größtenteils darauf.

Das W3C-DOM, das zum Zeitpunkt des Erstellens dieser Arbeit als Empfehlung im Level 2 und als Entwurf im Level 3 vorlag, basiert beim Setzen der Größe, der Position oder anderer Eigenschaften ebenfalls auf der style Property der einzelnen Elemente. Anders sieht es jedoch aus, wenn man Elemente erzeugen, löschen oder deren Inhalt verändern will. Hier entpuppt sich das neue DOM als echte XML-Struktur. Und so muss man, um zum gewünschten Objekt zu gelangen, durch den DOM-Baum wandern. Dafür steht die Methode getElementById des Document-Objekts zur Verfügung.

Ist das Element vom Typ TextNode, so kann sein Inhalt über die nodeValue Property ausgelesen bzw. verändert werden. Ist dem jedoch nicht so (handelt es sich z.B. um ein Image-Objekt), so müssen die passenden Methoden bzw. Properties gesetzt werden. Allerdings wurde dieser restriktive und saubere (wenn auch manchmal umständliche) Ansatz schon unter Netscape aufgeweicht, denn dort wird wie im Internet Explorer innerHTML unterstützt.

4.2 Event Modell-Inkompatibilitäten

Nicht ganz so ausführlich beschrieben, aber doch nicht unerwähnt bleiben, sollen die Unterschiede der Browser in Bezug auf das Event Handling.

Dynamic HTML lebt von Events. Erst das Abfangen und richtige Reagieren auf Mausbewegungen oder Mausklicks, auf Tastatur-Eingaben oder Scroll-Events erlaubt lebendige Interaktion mit dem Benutzer. Das auch hier recht unterschiedliche Wege gegangen wurde, darf nicht verwundern. Gab es vor wenigen Jahren nur ganz wenige verschiedene Event-Typen, so "...erkennt beim Internet Explorer 5 allein das Image-Objekt satte 49 Event-Typen"[25, S.49].

Ein erster Unterschied fällt bei der Registrierung eines Event-Handlers auf. Reicht es im Internet Explorer, eine im Interface des Objekts definierte Methode zu definieren (siehe folgendes Code-Beispiel), muss unter Netscape 4 bei einigen Event-Typen auch die Methode captureEvents aufgerufen werden.

```
1 <script>
2   document.all['theDIV'].onclick = function myOnClick()
3   {
4     // ... Event Handling
5   }
6 </script>
```

Das W3C definierte die Methode addEventListener, die es erlaubt, zu einem Event-Typ mehrere Handlermethoden an ein Objekt zu binden. Allerdings wurde, was das Event-Handling betrifft, auf Abwärtskompatibilität geachtet, und somit ist der gleiche Ansatz wie im Internet Explorer ebenfalls erfolgreich.

Der wohl größte Unterschied im Event-Handling Modell zwischen Netscapes und Microsofts Browsern besteht in der Reihenfolge, in welcher der Event die DOM-Hierarchie durchläuft: Im Navigator "sickert" der Event vom Window-Objekt hinunter zum tiefsten

Element des DOM-Baums, den das Event noch betrifft. Genau umgekehrt ist es im Internet Explorer – dort "blubbert" der Event nach oben. Die folgende Abbildung stellt diese beiden Varianten gegenüber.

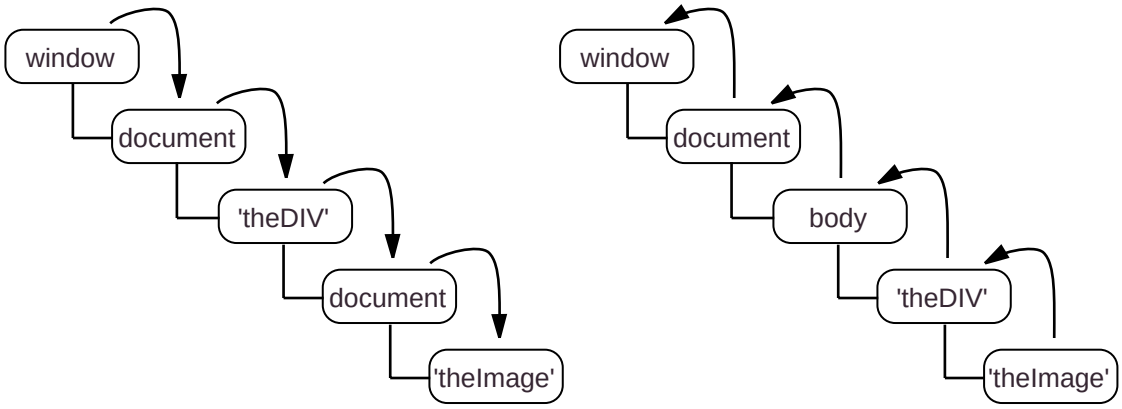

Abbildung 4.4: Der Weg eines Events unter Netscape Navigator (links) und Internet Explorer.

Glücklicherweise bietet Netscape die Methode routeEvent an, die ein Event an eventuelle weitere Handler weiterschickt. Der Aufruf dieser Methode am Anfang eines Handlers bewirkt, dass die Reihenfolge der des Internet Explorers angepasst werden kann.

Eine weitere Inkompatibilität besteht im unterschiedlichen Verankern des Event-Objekts. Unter Internet Explorer existiert dieses Objekt nur während ein Event auftritt, und zwar als Property des Window-Objekts. Unter Netscape wird das Event-Objekt als Parameter der Handler-Funktion übergeben.

Zusammenfassend kann ein einfaches Cross-browser Event-Handling wie im anschließenden Code-Fragment implementiert werden.

```
1  <script>
2    if ( document.captureEvents )
3      document.captureEvents(Event.CLICK);
4    document.onclick = function myOnClick(e)
5    {
6      if ( document.routeEvent )
7        document.routeEvent(e);
8      var evt = e | window.event;
9      // ... Event Handling
10   }
11 </script>
```

In Zeile 2 wird via Test auf Methoden-Existenz die Eigenheit von Netscape 4 berück-

sichtigt, dass Maus-Events speziell gecaptured werden müssen. Zeile 4 weist die im Anschluss definierte Funktion als Handler für Mausklicks dem Document-Objekt zu. Zeile 6 ermöglicht wie oben erwähnt, dass es auch unter Netscape 4 zu Event-Bubbling kommt. Schließlich wird in Zeile 8 der Variable evt das richtige Objekt zugewiesen – unter Netscape wird es ja als Paramtere der Handlerfunktion (hier e) übergeben, unter Internet Explorer existiert es als Feld des Window-Objekts (window.event).

Eine gute und einführende Zusammenfassung zu diesem Thema bietet [25].

4.3 Weitere Eigenheiten

In den folgenden beiden Abschnitten sollen Eigenheiten beschrieben werden, die den Entwicklern das Leben ein wenig schwerer machen. Den Anfang macht dabei der Internet Explorer, der trotzdem in fast jeder Hinsicht harmloser ist als das Konkurrenzprodukt von Netscape.

4.3.1 Internet Explorer-Spezialität

Durch gewisse Testfälle konnte eine interessante Eigenheit des Internet Explorers (verwendet wurden Version 5.0 und 5.5) festgestellt werden:

Die von Microsoft verwendete Architektur des Browsers behandelt das Ansprechen von window.opener wie eine virtuelle Client-Server-Verbindung (generell scheint dies bei vielen System-nahen Objekten so zu sein). Bemerkbar macht sich dies, wenn man versucht, Properties eines Objekts im "Vater-Fenster" vom neu erzeugten aus auszulesen, nachdem ersteres geschlossen wurde.[4] Diese Tatsache ist kein Problem, allerdings sollte man sich ihrer bewusst sein, wenn man auf Fehler obiger Art stößt.

4.3.2 Das Netscape Resize Problem

Ein ganz spezielles Thema beim Entwickeln von Dynamic HTML-Seiten ist das Netscape Resize Problem. Was beim Internet Explorer so gut wie keine Schwierigkeiten bereitet, führt zu großem Kopfzerbrechen bei Netscape-Browsern: das Verändern der Fenstergröße. Schlimm genug, dass es existiert – dazu kommt jedoch, dass Repaint-Verhalten und Kenntnis bezüglich Script-Elementen von vielen Faktoren abhängen und sich stark unterscheiden können.

Folgende Parameter bestimmen das Verhalten des Browsers[5]:

- Cache aktiviert oder nicht aktiviert
- Dokument über Webserver oder Filesystem geladen

[4] Man erhält dabei eine sehr interessante Fehlermeldung: "Der Aufgerufene (Server [nicht die Serveranwendung]) ist nicht verfügbar und verschwunden. Alle Verbindungen sind ungültig. Der Aufruf wurde nicht ausgeführt."

[5] Alle nachfolgenden Angaben beziehen sich auf Netscape 4.75 unter Windows 2000.

- Frameset oder framelose Seite
- Wenn Frameset, dann ob dynamisch oder statisch erzeugt

Für die folgende Tabelle wurde eine selbst geschriebene Testseite verwendet, als Webserver diente ein Apache 1.3. Vorweg eine kleine Erläuterung der verschiedenen Testseiten:

Framelose Seite Teile des Inhalts dieser Seite werden über document.write() erzeugt.

Statisches Frameset Darunter ist eine Seite mit mehreren Frames zu verstehen, deren jeweiliger Inhalt wie oben erzeugt wird. Die <frameset>-Definition ist statisches HTML.

Dynamisches Frameset Ähnliche dem statischen Frameset, jedoch wird auch die <frameset>-Definition mittels document.write()erzeugt.

Cache	Quelle	Framelose Seite	Statisches Frameset	Dynamisches Frameset
Ja	Webserver	Resize-Handler des Fensters wird aufgerufen.	Resize-Handler der Frames und des Framesets werden aufgerufen.	Resize-Handler der Frames und des Framesets werden aufgerufen.
Ja	Filesystem	Resize-Handler des Fensters wird aufgerufen.	Resize-Handler der Frames und des Framesets werden aufgerufen.	Resize-Handler der Frames und des Framesets werden aufgerufen.
Nein	Webserver	<script>-Blöcke im Body werden übersprungen. Resize-Handler wird aufgerufen, jedoch kann keine top-Methode aufgerufen werden.	Resize-Handler des Framesets wird aufgerufen. Bei dem Ansatz über src='javascript:...' sorgt Netscape selbst für Frame-Repaint.	Resize-Handler des Framesets wird aufgerufen. Jedoch verweigert Netscape sowohl den Ansatz über Redirecter-Pages als auch direkt via src='javascript:...' (top.frames enthält keine Objekte).

Tabelle 4.1: Das Resize-Verhalten Netscapes.

Cache	Quelle	Framelose Seite	Statisches Frameset	Dynamisches Frameset
Nein	File-system	Resize-Handler des Fensters wird aufgerufen.	Resize-Handler der Frames und des Framesets werden aufgerufen.	Resize-Handler des Framesets wird aufgerufen. Jedoch verweigert Netscape sowohl den Ansatz über Redirecter-Pages als auch direkt via src='javascript:...' (top.frames enthält keine Objekte).

Tabelle 4.1: Das Resize-Verhalten Netscapes.

Der Aspekt des Resize-Problems, dass Layer-Objekte – in jeglicher Konfiguration – nicht automatisch neu gezeichnet werden (was beim Internet Explorer immer stattfindet), wurde in Tabelle 4.1 nicht ausdrücklich festgehalten. Da dieser Effekt in jeder Konstitution auftritt, kommt man nicht umhin, entsprechende Handler zu schreiben.

Bei einem genaueren Blick auf die Tabelle fällt auf, dass die beschriebenen Probleme erst dann auftreten, wenn der Cache deaktiviert ist. Somit sollte es allen Benutzern der Web-Applikation, die Netscape benutzen, ans Herz gelegt werden, den Cache ja nicht auszuschalten.

Einen weiteren Wermutstropfen gibt es noch: Nach oftmaligem Resize verliert Netscape auch trotz Caches den gesamten Javscript-Scope – und somit unterbleibt das Aufrufen eines Resize-Handlers komplett.

Kapitel 5

Design Pattern

"Designing object-oriented software is hard,
and designing reusable object-oriented software
is even harder." [8, S.1]

5.1 Einführung

Mit dem Erscheinen des Buches "Design Patterns. Elements of Reusable Object-Oriented Software" von Gamma, Helm, Johnson und Vlissides im Jahr 1994 erlebte die OOP-Szene so etwas wie einen zweiten Frühling.

Die mit 400 Seiten für ein Entwickler-Handbuch eher schlanke Publikation übernahm einen Begriff aus der Architektur, um immer wiederkehrende Muster zu beschreiben – den Begriff der "Design Pattern". Christopher Alexander definiert diesen Begriff aus dem Bereich der Planung von Städten und Bauwerken folgendermaßen: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [7].

Die Autoren stellen in ihrem Werk fest, dass viel zu oft die Eigenheiten der Objekt-Orientierten Programmierung (z.B. Polymorphismus) wenig oder falsch genutzt werden. Doch das Buch ist nicht vorwiegend eine Anleitung zu gutem Objekt-Orientierten Design (obwohl auch das zutrifft), sondern bietet vor allem einen Katalog von bewährten und leicht erweiterbaren Klassen- und Interface-Strukturen, die auf viele Probleme anwendbar sind.

Diese 23 Design Pattern werden durch folgende Attribute beschrieben[1]:

Name and Classification Der Name soll das Muster gut beschreiben, außerdem wird es einer der Kategorien "Creational", "Structural" oder "Behavioral" zugeordnet.

Intent Eine Kurzzusammenfassung des Musters.

[1]Da dem Autor keine deutsche Übersetzung vorliegt und eine eigene zu Missverständnissen führen könnte, sind die Bezeichnungen der Attribute in Englisch belassen.

Also Known As Andere, sich etabliert habende Bezeichnungen.

Motivation Ein Beispiels-Szenario, das den Bedarf des Design Patterns aufzeigt.

Applicability In welchen Situation kann das Muster angewandt werden? Wie können diese erkannt werden?

Structure Eine auf OMT basierende grafische Notation, die Aufschluss über die Klassenstruktur gibt.

Participants Eine Beschreibung der benutzten Klassen bzw. Objekte.

Collaborations Wie wirken die Komponenten aufeinander?

Consequences Welche Vor- und Nachteile ergeben sich aus der Anwendung?

Implementaton Auf was ist bei der Implementierung besonders zu achten? Gibt es Sprach-spezifische Eigenheiten?

Sample Code Teile aus einem in C++ oder Smalltalk geschriebenen Anwendungs-Beispiel.

Known Uses In welchen bekannten Bibliotheken bzw. Applikation oder Frameworks wird das Pattern benutzt?

Related Patterns Hier findet man eine Liste mit verwandten und häufig in Kombination mit diesem Muster auftretenden anderen Design Patterns.

Das sehr zu empfehlende Buch, das sich als Referenz versteht und nicht einmal gelesen und dann weggelegt werden will, zeigt anhand einer Beispiels-Applikation (ein WYSIWYG Dokumenten-Editor) die Anwendung einiger Muster, der Hauptteil setzt sich jedoch aus einer genauen Beschreibung der Pattern zusammen.

Diese Sammlung ist natürlich nicht vollständig – und so entstanden sehr bald Ergänzungen, weitere Muster und rege Diskussionen rund um den ursprünglichen Katalog. Der interessierte Leser sei dazu auf [24] verwiesen.

5.2 Verwendete Design Pattern

Das Design der praktischen Arbeit wurde von einer Reihe von Entwurfs-Mustern inspiriert, deren Anwendung sich anbot. Im Folgenden werden diese genauer beschrieben.

5.2.1 Composite

Dieses Pattern bietet sich dann an, wenn eine komplexe Struktur (meist eine Baumstruktur), die sich aus beliebig vielen Teilen zusammensetzen kann, gleich anzusprechen sei wie die einzelnen Komponenten selbst. Ein Beispiel dafür sind Glyphen – eine abstrakte Repräsentation für darstellbare Elemente. Im Kontext einer Textverarbeitung umfasst

dies einfache grafische Komponenten ebenso wie Absätze oder ganze Tabellen. Selbst eine komplette Seite fällt in diese Definition.

Nun ist es wünschenswert, dass alle Glyphen gleich anzusprechen sind. Eine jede Komponente reagiert ihrer Natur entsprechend, doch sind sie alle über ein gemeinsames Interface steuerbar.

In einer Composite Struktur gibt es folgende Elemente (siehe auch Abbildung 5.1[2]):

Component Diese abstrakte Klasse definiert die Methoden der Subklassen, welche von diesen unterschiedlich (bzw. auch gar nicht) implementiert werden (z.B. paint(), add(Component), remove(Component) etc.).

Leaf Diese Klasse definiert Elemente, die keine weiteren Kinder besitzen ("primitive objects"). Sie implementieren eine Teilmenge der Methoden der Component Klasse (z.B. paint()).

Composite Ein Composite kann im Gegensatz zu Leafs weitere Kinder besitzen und definiert Kinder-abhängiges Verhalten (z.B. rufe paint() die paint()-Methode bei allen Kindern auf).

Client Der Client kennt nur das Interface des Component und spricht dementsprechend mit den Komponenten.

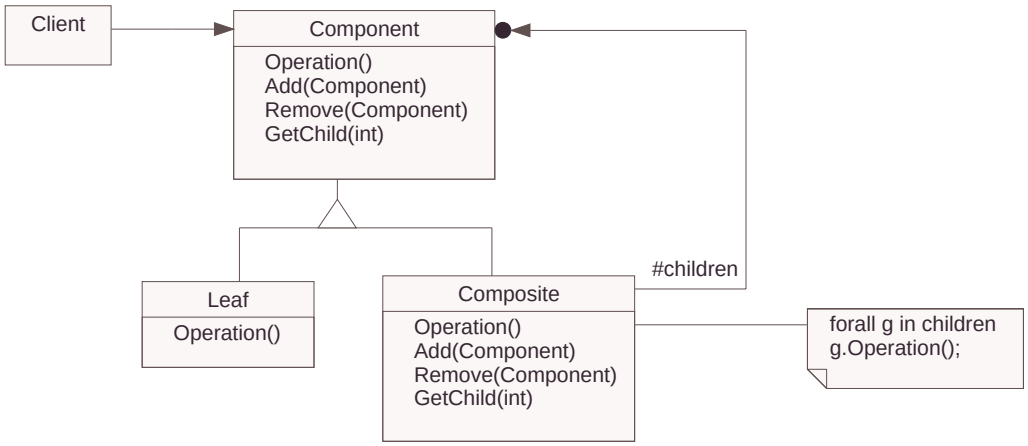

Abbildung 5.1: Struktur des Composite Design Pattern.

Dieses Design Pattern bietet sich optimal für die Implementation der Container-Struktur des JavaScript AWT's an. Auch hier sollen einem Container zusätzliche Komponenten (darunter auch weitere Container) übergeben werden können, die vom Layout Manager des Containers in Folge richtig ausgerichtet und platziert werden.

[2]Dieses UML-Klassendiagramm in OMT-Notation (und alle folgenden in diesem Abschnitt) wurden von [8] übernommen.

5.2.2 Strategy

Mit diesem Entwurfs-Muster wird es ermöglicht, Familien von Algorithmen zu definieren und zu kapseln – und ein gemeinsames Interface dafür zu definieren. Abhängig von entsprechenden Faktoren kann ein Client nun den einen oder den anderen Algorithmus verwenden. Umgekehrt kann ein Algorithmus von verschiedenen Clients genutzt werden (diese müssen aber das gleiche, dem Algorithmus bekannte Interface implementieren).

Folgende drei Klassen bzw. Interfaces sind dabei zu beachten:

Strategy Dieses Interface definiert die Schnittstelle eines Algorithmus in Richtung Context.

ConcreteStrategy Diese Klasse implementiert einen konkreten Algorithmus, dem Strategy Interface entsprechend.

Context Der Client hält eine Referenz auf ein Strategy Objekt und ruft bei Bedarf eine Methode dieses Elements auf. Für größere Modularität ist anzuraten, auch ein eigenes Interface in Richtung Strategy zu definieren.

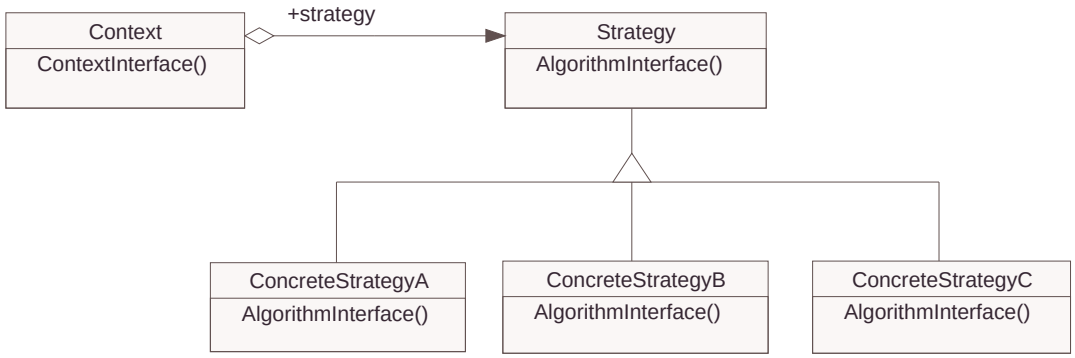

Abbildung 5.2: Struktur des Strategy Design Pattern.

Für den Umgang mit den Browser-Inkompatibilitäten bietet sich dieses Muster an – das betrifft den Bereich der Layer-Manipulation und den des internen Event-Handlings. Hier wird das Strategy Pattern auch implementiert, in Verbindung mit einer entsprechenden Abstract Factory (siehe nächstes Design Pattern).

5.2.3 Abstract Factory

Dieses Muster bietet ein Interface zur Generierung von ähnlichen Objekten, ohne die konkrete Klasse zu spezifizieren. Welches Objekt wirklich erzeugt wird, hängt dabei vom jeweiligen Kontext ab. So kann z.B. ein austauschbares Look-and-Fell auf diese Weise implementiert werden.

Die folgenden fünf Klassen bzw. Interfaces sind an diesem Pattern beteiligt:

AbstractFactory Dieses Interface definiert die Operationen zur Erzeugung von abstrakten Produkten.

ConcreteFactory Diese Klasse implementiert die Methoden zur Generierung von ConcreteProduct Objekten.

AbstractProduct Gemeinsames Interface der zu erzeugenden Objekte.

ConcreteProduct Definiert ein Objekt, das von einer ConcreteFactory erzeugt werden kann und implementiert das AbstractProduct Interface.

Client Benutzer dieses Patterns greifen nur über AbstractFactory- und AbstractProduct-Interfaces auf die agierenden Komponenten zu und verwenden niemals selbst new-Operationen (z.B. anAbstractFactory.createAbstractProduct()).

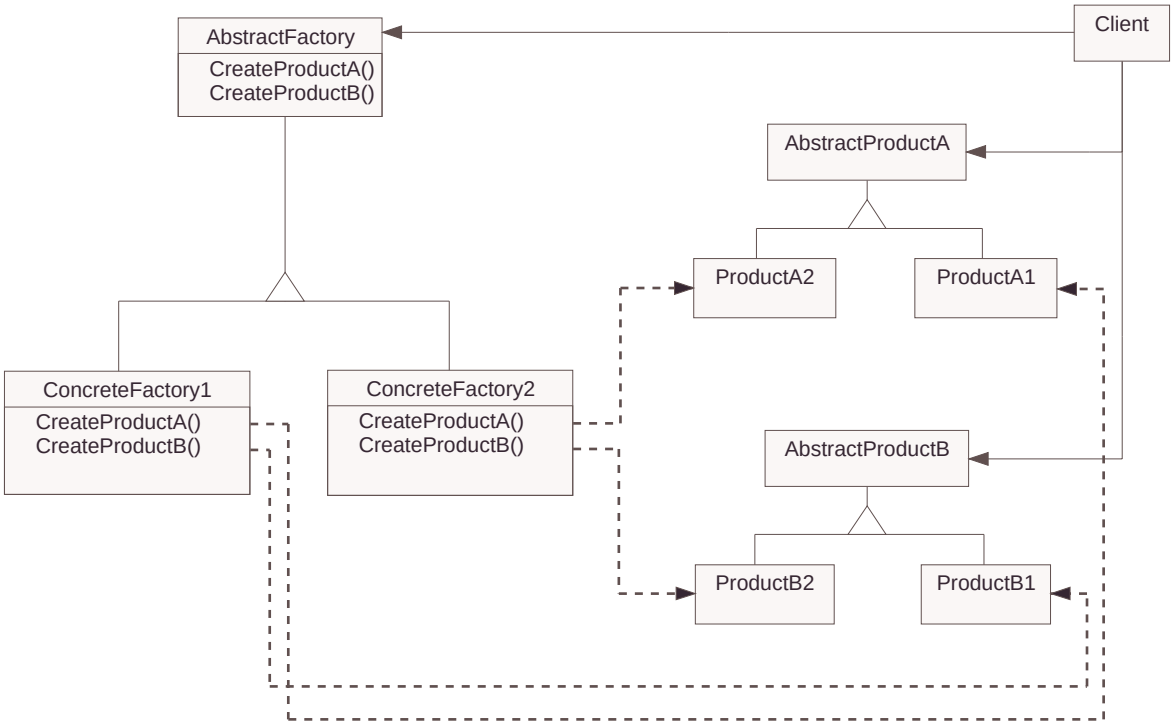

Abbildung 5.3: Struktur des Abstract Factory Design Pattern.

Im Zusammenhang mit den über das Strategy Pattern erzeugten Browser-Abstraktionen wird das Abstract Factory-Konzept im JavaScript AWT verwendet. So erfolgt die Generierung des Objekts, das sich um das interne Event Handling kümmert, über eine Factory, die abhängig vom Browsertyp ein entsprechendes ConcreteProduct instanziiert.

5.2.4 Observer

Für JavaScript- bzw. Java-Programmierer ist dieses Design Pattern recht vertraut – entspricht es doch (teils) dem Handler- bzw. Listener-Prinzip der beiden oben genannten Sprachen.

Das Observer Pattern definiert eine 1-zu-N-Beziehung, die es ermöglicht, beim Ändern des Zustands eines Objekts andere, davon abhängige Objekte zu informieren, damit sie entsprechend darauf reagieren können.

Eine Observer Struktur umfasst folgende Elemente:

Subject Die Subject Klasse verwaltet die bei ihr registrierten Observer, die sich in den meisten Implementationen auch wieder abmelden können.

Observer Dieses Interface definiert eine (oder mehrere) Metode(n), die von einem Subject aufgerufen werden, um den Observer über eine Änderung des Zustands zu informieren.

ConcreteSubject Diese Subklasse von Subject informiert ihre Observer, wenn sich ihr eigener interner Status geändert hat.

ConcreteObserver Implementiert das Observer Interface um von einem Subject benachrichtigt werden zu können. Je nach Implementation besitzt diese Klasse eine Referenz auf ein Subject bzw. bekommt alle notwendigen Daten mit der Benachrichtigung mitgeschickt.

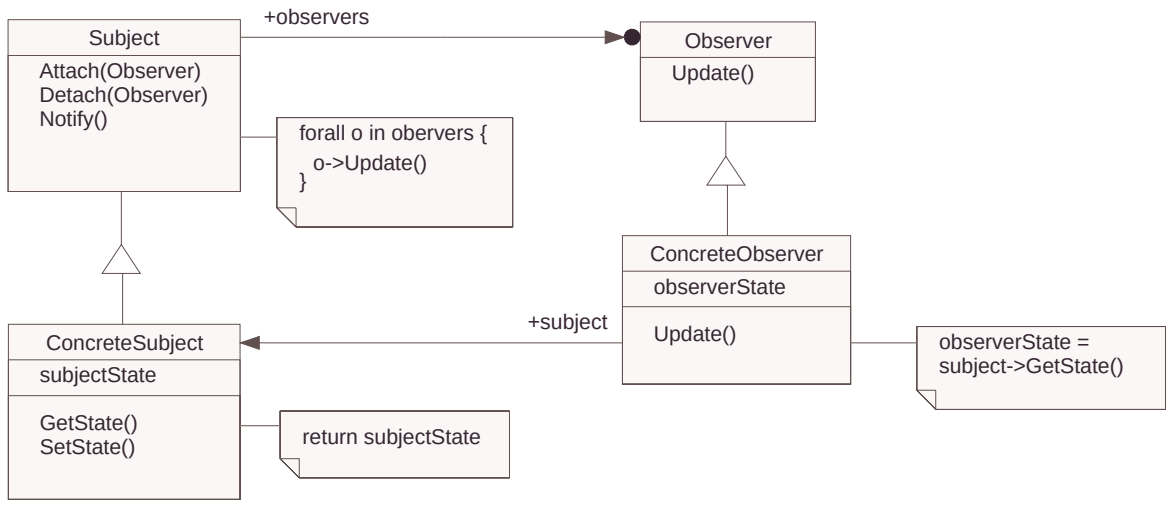

Abbildung 5.4: Struktur des Observer Design Pattern.

Das Event-Handling des JavaScript AWT's wurde nach dem Observer-Prinzip modelliert. Die Klasse CSEventSource entspricht dabei der Subject Klasse, alle ihre Subklassen sind ConcreteSubjects. Gemäß JavaScript-Konvention gibt es jedoch keine einheitliche

Update-Methode, sondern onEvent-Methoden, wobei "Event" für den jeweiligen Event-Typ steht (z.B. onMouseMove).

5.2.5 Singleton

Die Intention des Singleton Patterns ist, dafür zu sorgen, dass von einer Klasse nur eine einzige Instanz erzeugt wird sowie einen globalen Zugriff auf dieses Objekt zu schaffen.

Der Ansatz, einfach eine globale Variable zu benutzen, schützt vor mehrfacher Instantiierung nicht (und sprengt jegliches Objekt Orientiertes Denken). Die Idee des Singletons ist es nun, der Klasse selbst die Verwaltung ihrer einzigen Instanz zu überlassen.

Die Struktur ist simpel:

Singleton Die Singleton-Klasse bietet eine statische Methode, um auf die ebenfalls statische Instanz zuzugreifen. Sie muss darauf achten, dass mittels new-Operation keine Instanz der Klasse angelegt werden kann.

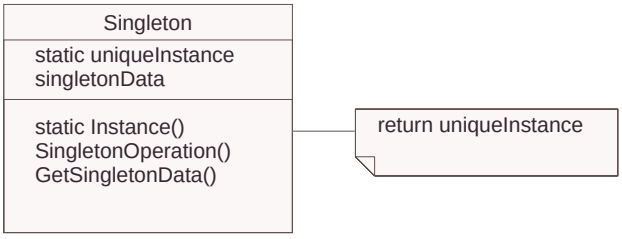

Abbildung 5.5: Struktur des Singleton Pattern.

Im JavaScript AWT wurde dieses Entwicklungsmuster in zwei verschiedenen Ansätzen integriert: Die Klassen CSWindow und CSBrowserAbstractionFactory werden direkt via getInstance-Methode angesprochen, die Klassen für das Browser-abhängige Layer-, Window- und Event-Handling werden über eine Factory erzeugt bzw. referenziert.

5.3 Pattern in der Web-Applikations-Entwicklung

Im Unterschied zur nunmehr schon langjährigen Erfahrung, die weltweit im Bereich der Objekt-Orientierten Programmierung besteht, ist die Entwicklung von Web-Applikationen eine sehr junge Disziplin. Somit verwundert es nicht, dass es noch wenige Pattern-Ansätze in diesem Bereich gibt – denn diese beruhen ja auf der Wiederverwendung länger erprobter Methoden.

Bei den gefundenen Ansätzen ist zwischen Design Pattern und Architectural Pattern zu unterscheiden. Während erstere den oben besprochenen ähneln und nur in einer anderen Umgebung angesiedelt sind, beschreiben zweitere Web-Applikationen im generellen auf einem höheren Abstraktionslevel. Obwohl es solche Architectural Pattern auch im Bereich

der Nicht-Web-Applikationen[3] gibt, wurde im Rahmen dieser Arbeit darauf verzichtet, diese näher vorzustellen.

5.3.1 Design Pattern

Einer der ersten Ansätze in diese Richtung findet sich bei [32]. Nathan Wallace zeigt darin die unterschiedlichen, in vielen Web-Applikationen auftretenden Problemstellungen[4] auf, deren Lösungsmöglichkeiten hin zu Design Patterns führen.

Wallace macht klar, dass Pattern im Bereich der Web-Applikations-Entwicklung alle betroffenen Schichten (von der Webseite über die Programmlogik bis zur Datenbank) beschreiben sollten, da gerade das komplexe Zusammenspiel dieser Ebenen eine gute Planung verlangt. Anhand des Beispiels eines Filters (Ausführen einer Funktion am Beginn oder Ende eines Requests) zeigt er schließlich, wie eine solche Beschreibung erfolgen könnte. Dabei unterscheiden sich die Attribute, die er dazu benutzt, nur kaum von denen aus der Objekt-Orientierten Programmierung, hinzugefügt wird lediglich der Punkt "Sequence of Events".

Abgesehen davon findet sich der Begriff der Design Pattern noch sehr selten im Zusammenhang mit Web-Entwicklung. Lediglich bei [29] und bei [13] wird von der Einführung eines neuen Patterns für bzw. vom Anwenden des Model-View-Controller-Patterns[5] auf Web-Applikationen gesprochen.

5.3.2 Architectural Pattern

Architectural Pattern im Bereich Web-Development beschreibt Jim Conallen in [2, S.99ff]. Damit bewegt er sich auf einer höheren Abstraktionsstufe, jedoch beschreibt auch er sehr konkret drei Modelle und ihre Struktur:

1. *Thin Web Client* – dieses Modell beschreibt eine Architektur, in der wenig Logik im Client liegt. Somit genügt ein Formular-unterstützender Browser, um die gewünschten Interaktionen durchzuführen.

2. *Thick Web Client* – eine solche Architektur zeichnet sich dadurch aus, dass ein großer Teil der Applikations-Logik am Client ausgeführt wird (via Java Applets oder ActiveX-Komponenten). Die Kommunikation mit dem Server erfolgt jedoch noch wie beim Thin Web Client über HTTP.

3. *Web Delivery* – in diesem Modell agiert die Web-Applikation als verteiltes Objekt-System. Die Kommunikation zum Server basiert auf anderen Protokollen denn HTTP (z.B. IIOP[6] oder DCOM[7]), welche die Nutzung von remote objects unterstützen.

[3] Man denke dabei an Ansätze wie das Layer-Modell oder das Model-View-Controler-Prinzip.
[4] Form Processing, Navigation, Database Operations, Authentification... (um nur einige zu nennen).
[5] Dieses Pattern ist eigentlich jedoch mehr den Architectural Pattern zuzurechnen.
[6] Internet Inter-ORB Protocol
[7] Distributed Component Object Model

Auf über zehn Seiten beschreibt Conallen – eine auf Web-Applikationen zugeschnittene UML-Symbolik verwendend – diese Architekturen und ihre Anwendungen, ihren Aufbau sowie die damit verbundenen Konsequenzen. Eingebettet ist dieser Bereich in einen auf UML basierenden Entwurfsprozess, der in diesem Buch gut beschrieben wird.

Kapitel 6

Praktischer Teil

"If you can't make it good,
at least make it look good."
Bill Gates, Microsoft

6.1 Einführung

Obiges Zitat des Chief Software Architect des wohl bekanntesten IT-Unternehmens der Welt kann durchaus als Motivation für die Qualität des angestrebten Entwicklungsprozesses herangezogen werden. Allerdings im abschreckenden Sinne – denn das Ziel war es, einen hochwertigen und nachvollziehbaren Weg einzuschlagen, an dessen Ende wirklich das gewünschte Produkt in gewünschter Qualität steht. Gerade für diese Arbeit war das sehr wichtig, denn eine Bibliothek wie diese muss leicht erweiterbar und stabil sein.

Zu diesem Zweck wurde ein Prozess gewählt, der zwischen Extreme Programming[1] und einem Ansatz wie jenem von Rational[2] liegt. Im Laufe der Planungsphase wurden einerseits viele Prototypen erstellt, um Performance- und Machbarkeitsstudien zu generieren, andererseits wurde beim Architectural und Detailed Design auf UML gesetzt.

6.2 Software Requirements

Die SRD's sollen hier vor aus zwei Gründen angeführt werden: Zum einen bieten sie einen guten Einblick in die gestellten Anforderungen und führen gleichzeitig in die verschiedenen Problematiken ein (einige Zeilen sollten allerdings, im Nachhinein betrachtet, eher in einem Architectural Design Document stehen, da sie Aussagen über den Lösungsweg tätigen).

[1] Der Begriff "Extreme Programming" wurde von Kent Beck eingeführt und beschreibt eine Entwicklungsmethode, die auf den Punkten Coding, Testing, Listening und Designing beruht – hier nach absteigender Gewichtung sortiert. Im Mittelpunkt stehen dabei kurze Zyklen eines inkrementellen Prozesses aus Codierung und Testen, wobei sehr viel Sorgfalt auf automatisierte, gute Testfälle gelegt wird. Nähere Informationen dazu finden sich bei [1], [9] und [10].

[2] Details zum Rational Unified Process finden sich unter [5].

Zum anderen können sie, wenn das fertige Produkt vorliegt, herangezogen werden, um ein prinzipielles Übereinstimmen zwischen Gewünschtem und Produziertem festzustellen. In Abschnitt 6.6.1 ist dies auch geschehen (allerdings erfolgte diese Überprüfung durch den Autor selbst, wäre jedoch Aufgabe einer Qualitäts-Abteilung gewesen).

Die Anforderungen verlagerten sich von den unter Abschnitt 1.3 aufgeführten Punkten zu Beginn der Design-Phase immer mehr zum reinen Entwickeln eines API's für Client-seitiges JavaScript. Dementsprechend finden sich keine Requirements mehr, die sich mit der Kommunikation mit einem darüberliegenden Transfer-Layer beschäftigen. Diese Anliegen wurden (und werden) im Rahmen des Hyperwave Application Frameworks (HAF) behandelt.

6.2.1 Anforderungen

Ziel dieses Dokuments ist es, zu einem Konzept zur Client-seitigen Generierung von Browser gestützten User Interfaces zu kommen. Aus einigen konkreten Anforderungen soll ein Design entwickelt werden, welches für einen Prototypen verwendet werden kann.

SR.1 JavaScript
: Das AWT ist in Client Side JavaScript zu realisieren, wobei Browser von Netscape und Microsoft ab den Versionen 4 (jeweils die aktuelleren Releases) zu unterstützen sind. Ebenso ist die Vorbereitung auf Browser der neuesten Generation, die (weitgehend) das W3C DOM implementieren, vorzusehen.

SR.2 Komponentensystem
: Ähnlich dem Composite Design Pattern soll das GUI aus Glyphen aufgebaut werden, welche verschachtelbar seien (vergleiche Interviews, Java AWT, Swing).

SR.3 Layout Management
: Ein und die selbe Glyphenmenge soll mittels verschiedener Layout Manager in unterschiedlichen Darstellungen (Anordnungen) renderbar sein. Das Layout Management soll austauschbar sein. Die Erfüllung der minimalen Anforderungen stellt eine BoxLayout-ähnliche Implementation dar (vergleiche Java Swing).

SR.4 Event Handling
: Das AWT soll ein offenes Event Handling Schema unterstützen, das auf der Grundlage von Listener Objekten (bzw. Interfaces) basiert.

SR.5 Drag-and-Drop
: Einzelne Glyphen seien mit der Eigenschaft auszustatten, mittels Drag-and-Drop verschiebbar zu sein. Das heißt, es müssen die Objekte mit Drag Source- bzw. Drop Target-Eigenschaften auszustatten sein.

SR.6 HTML-Rückwärtskompatibilität und Ressourcen
: Das Interface der Basiskomponenten muss so offen sein, dass nachträgliche Modifikationen (z.B. aufgrund von Cascading Style Sheets, Hintergrundbilder, etc.) noch

möglich sind. Dies ermöglicht eine Art Rückwärtskompatibilität zur HTML Welt in dem Sinne, dass wichtige Eigenschaften sowohl bei HTML Komponenten als auch bei den Basiskomponenten setzbar sind.

SR.7 Umgehung von Browserinkompatibilitäten
Auf Grund der unterschiedlich implementierten DOM's der einzelnen momentan gängigen Browser (Versionen 4 und 5/6 von IE und Netscape) ist es notwendig, eine Technik zu nutzen, um diese Unterschiede auszugleichen. Problematisch sind in dieser Hinsicht v.a. die Version 4 von Netscape und die Tatsache, dass es beim IE 5 zwischen den Windows- und Mac-Versionen Differenzen gibt.

Begriffsdefinitionen zum SRD

AWT Abkürzung für "Application Window Toolkit". Im Kontext dieses Dokuments jedoch als "Web Application Browser Window Toolkit" zu verstehen.

DOM Abkürzung für "Document Object Model". Beschreibt, wie die Hierarchie eines HTML-Dokuments Objekte verwaltet, wie auf solche zugegriffen werden kann und wie ihre Attribute geändert werden können.

Glyph Komponente, die renderbaren HTML-Code repräsentiert.

Ressource Komponente des User Interface, die getrennt von den Applikationen bzw. Modulen an einem genau definierten Ort des Applikationsservers wohnen soll.

6.3 Die Architectural Design Phase

Während der Planungsphase wurden etliche Testfiles erstellt. Ein Teil davon beschäftigte sich mit JavaScript Core Themen wie Lexical Scoping[3] oder der Realisierung von statischen Variablen, andere mit DOM- oder Event-spezifischen Problemen. Zu diesen gehören Tests bezüglich verschachtelten Layern, der Nutzung von Cascading Style Sheets, dem Löschen von Layern, Mouse- und Window-Events.

Größere Prototypen, die dabei entwickelt wurden, beschäftigten sich mit ersten Versuchen zum Layout-Management und zur Implementierung von Drag and Drop – auch über Frames hinweg. Letzterer brachte es auf über 1200 Zeilen Code und zeigte, dass Drag and Drop samt grafischer Visualisierung (in Form eines Icons neben dem Mauszeiger) auch über Frames hinweg möglich ist. Und das sowohl unter Internet Explorer 4 und neuer als auch unter Netscape 4. Obwohl das neue DOM des Navigators 6 leicht zu integrieren war, konnte hier ein Frame-übergreifendes Drag and Drop nicht realisiert werden (siehe auch Abschnitt 6.3.2).

Trotzdem zeigte dieser Prototyp, dass das angestrebte Ziel erreichbar ist und erlaubte ein genaueres Spezifizieren und Planen der Architektur. Ein solcher Ansatz entspricht

[3]Unter Lexical (oder Static) Scoping versteht man, dass Funktionen in jenem Scope exekutiert werden, in dem sie definiert wurden, nicht in jenem, der zum Zeitpunkt der Ausführung gilt.

stark dem des Extreme Programming, in dem sehr früh der Umfang bzw. die Funktionalität (Scope) des Endprodukts vom Kunden an Hand von ersten ausführbaren Teilen revidiert werden soll. Denn oft genug hat der Kunde keine genaue Vorstellung davon, was machbar ist und erkennt erst nach langer Design- und Implementierungsphase bei der Präsentation einer Alpha-Version, dass er sich alles ein wenig anders vorgestellt hat. Kent Beck schreibt in diesem Zusammenhang über eine Sichtweise, die oftmaligen Veränderungen von Requirements offen gegenübersteht: "This would have to be a process that tolerated change easily, because the project would change direction often. You wouldn't want to spend a lot on software that turned out not to be used. You wouldn't want to build a road you never drove on because you took another turn."[1, S.19]

6.3.1 Grundobjekte

Folgende Grundobjekte wurden im Rahmen der ersten Designphase identifiziert und sind hier mit ihren Spezifika, Methoden und Eigenschaften angeführt:

Frameset

- Wrapper für HTML Frameset
- Notwendig für Frame-übergreifendes Drag-and-Drop
- Schreibt HTML Frameset
- Resize-Kontrolle
- Event-mässig selbe Struktur wie Composite (Listeners möglich)
- calculateFrameDimensions()-Methode (NS, Drag-and-Drop)
- getFrameAt(x,y)-Methode (oder Methodik)

Frame

- HTML Frame/Document-Wrapper
- Kontrolle über Resize (zumindest wenn kein Frameset da ist), Scrolling
- Besitzt Liste von Composites
- Event-mässig selbe Struktur wie Composite (Listeners möglich)
- Methode(n) zum "Reparieren" der Layer-Objekte nach Resize (NS)
- GetCompositeAt(x,y)-Methode (oder Methodik)

Composite

- Keine Unterscheidung in Container (Node) und Component (Leaf)
- Kann als Container weitere Composites hinzugefügt bekommen [Composite.add(Composite)]
- Besitzt Event-Handling (auf Listener basierend – siehe unten)
- Hat Standard-Layout-Management (absolute Positions-Angaben oder FlowLayout?)
- Layout Management kann "on the fly" geändert werden [Composite.setLayout(LayoutManager)]
- Methoden-Namen stark an Java AWT/Swing angelehnt
- Standard-Methoden: add, remove, getName, setName, getLayout, setLayout, getX, getY, getWidth, getHeight, getZIndex, getVisible, getBackground, getForeground, setX, setY, setWidth, setHeight, setZIndex, setVisible, setBackground, setForeground, getStyle, setStyle, reshape
- Weitere Methoden: getPreferredSize, getMinimumSize, getMaximumSize, setPreferredSize, setMinimumSize, setMaximumSize, (addNotify, removeNotify,) invalidate, isEnabled, setEnabled
- Markierung eines Composite: isSelectable, setSelectable, isSelected, setSelected (Auswirkung: Änderung des Styles oder andere Render-Funktion (setSelectedStyle/setSelectedHTML)?)
- Drag-and-Drop: isDragSource, setDragSource, isDropTarget, setDropTarget
- Setzen und Auslesen des Inhalts (eines Leafs): getHTML, setHTML oder getNodeValue, setNodeValue (letztere beiden Methoden entsprächen W3C-DOM)
- add und remove können auch einen Index zu/statt einem Composite übergeben bekommen
- Event-Handling: addListener-Methoden, removeListener-Methoden (diesen Methoden wird jeweils das Listener-Onjekt sowie der Eventtyp, z.B. "MouseMove", übergeben)
- Private Methoden: createPhysicalLayer, deletePhysicalLayer

LayoutManager

- Arten: BoxLayout, FlowLayout, GridLayout(?)
- Kennen ihren Composite, für den sie zuständig sind
- Methoden: layoutContainer oder layoutComposite? (Frage der Nomenklatur)

Event

- Wichtige Informationen: Source, JavaScript-Event
- Frage: Soll es möglich sein, eigene Events zu erzeugen (Problem: man kann keine JavaScript-Events erzeugen)? Wie sehen diese Events dann aus?
- Typen: alle möglichen dHTML-Events sowie eigene
- Methoden der Listener: onEvent-Methode ("Event" steht für speziellen Eventtyp, z.B. "MouseMove"), isListenerFor(?)

Anmerkung: Generell sollte nicht alle Funktionalität auf einmal in die Composite-Klasse gesteckt werden, sondern Erweiterungen (z.B. für Drag-and-Drop, Events, LayoutManager etc.) nach Belieben dazugeladen werden können (über import-Statements, vgl. DynAPI 2).

6.3.2 Browser-Grundanforderungen bzw. Einschränkungen

In diesem Zusammenhang waren vor allem die in Kapitel 4 zusammengefassten Beschränkungen und Inkompatibilitäten der verschiedenen Browser zu beachten. Eine zum frühen Zeitpunkt der Planung aufgestellte Liste sah folgendermaßen aus:

Layer Objekt-Unterstützung (NS 4)

- Schreiben mit layer.document.write
- Auslesen nicht möglich (d.h. man muss sich selbst den Content merken)

DOM á la IE 4+ (DIV-Tags, innerHTML)

- Schreiben mit document.all["layer"].innerhtml
- Auslesen über selbe Technik

W3C-DOM (NS 6)

- Schreiben und Auslesen mit getElementById("layer").innerhtml (Variante NS 6, nicht W3C-DOM konform)
- Schreiben mit getElementById("parent").appendChild(layer) (Variante W3C)

Machbarkeitsstudie

Als eine Art Machbarkeitsstudie für eine Client-seitige JavaScript Bibliothek, die Dynamic HTML-Basisfunktionalität Browser-unabhängig zur Verfügung stellt, wurde der schon oben erwähnte Prototyp entwickelt und unter den verschiedenen Versionen der beiden meist verbreiteten Browsern, Microsofts Internet Explorer und Netscapes Navigator, getestet.

Eine Hauptanforderung an diesen Prototyp war die Frage des Visualisierens und Kontrollierens von Drag and Drop über Framegrenzen hinweg. In Hinblick auf diese Aufgabe ergaben sich die Punkte der Spalte "Anforderungen" von Tabelle 6.1 und Tabelle 6.2.

Anforderung	IE 4 Win32	IE 5.0 Win32	IE 5.0 Mac	IE 5.5 Win32
MouseMove-Koordinaten bei Drag	Auf richtigen Frame bezogen (wenn Focus weggesetzt wird)	Auf Ursprungsframe bezogen, d.h. man muss Maus über Frames bewegt haben, um dorthin einen Drag machen zu können	Keine Events in anderem Frame	Auf richtigen Frame bezogen
MouseUp-Event bei Drag	Wird nicht generiert (Workaround: bei nächstem mouseMove oder MouseDown wird Drag erkannt und beendet)	Kommt an	Kommt im Ausgangs-Frame an	Kommt an
Frame-Dimensionen vorberechnen	Keine Möglichkeit gefunden, spielt aber wegen der richtigen MouseMove-Koordinaten keine Rolle	Keine Möglichkeit gefunden, wäre aber gerade hier sehr nützlich (siehe oben)	Keine Möglichkeit gefunden, spielt aber wegen der fehlenden Events keine Rolle	Keine Möglichkeit gefunden, spielt aber wegen der richtigen MouseMove-Koordinaten keine Rolle

Tabelle 6.1: Anforderungen des Prototyps (Internet Explorer).

Anforderung	IE 4 Win32	IE 5.0 Win32	IE 5.0 Mac	IE 5.5 Win32
Window Move-Event	Existiert nicht, spielt aber wegen der richtigen MouseMove-Koordinaten keine Rolle	Existiert nicht, d.h. eine setInterval-Funktion, die eine Verschiebung des Fensters mitbekommt, ist notwendig, um dann coordsUpdated auf false zu setzen	Existiert nicht	Existiert nicht, spielt aber wegen der richtigen MouseMove-Koordinaten keine Rolle
Key-Events (im Prototyp irrelevant)	Ja (pro Frame ein Handler notwendig); nicht immer KeyUp erkannt	Ja (pro Frame ein Handler notwendig)	Nicht getestet	Ja (pro Frame ein Handler notwendig)
Status der CTRL-Taste bei MouseEvent abgefragt	Ja	Ja	CTRL-Taste ist in MaxOS fix vorbelegt, aber z.B. Shift-Taste kann abgefragt werden	Ja
Sich daraus ergebende Handicaps	Workaround für fehlendes MouseUp	Maus muss vor Drag-and-Drop über Framegrenzen hinweg schon über betroffenen Frames gewesen sein	Drag-and-Drop über Frames hinweg geht scheinbar nicht	Keine

Tabelle 6.1: Anforderungen des Prototyps (Internet Explorer).

Anforderung	NS 4.75 Win32	NS 4.61 Unix	NS 6 PR3 Win32
MouseMove-Koordinaten bei Drag	Auf Ursprungsframe bezogen, d.h. der richtige Frame wird aus den Screen-Koordinaten ermittelt	Auf Ursprungsframe bezogen, d.h. der richtige Frame wird aus den Screen-Koordinaten ermittelt	Keine Events in anderem Frame (ausser bei Markierung)
MouseUp-Event bei Drag	Kommt an	Kommt an	Nein (siehe oben)
Frame-Dimensionen vorberechnen	Ja – realisiert in calculateFrameDimension	Ja – realisiert in calculateFrameDimension	Keine Möglichkeit gefunden, spielt aber wegen der fehlenden Events keine Rolle
Window Move-Event	Existiert, d.h. man kann einen onMove-Handler installieren, der calculateFrameDimension aufruft	Nicht unterstützt unter Unix, d.h. eine setInterval-Funktion, die eine Verschiebung des Fensters mitbekommt, ist notwendig, um dann calculateFrameDimension aufzurufen	Existiert nicht
Key-Events (im Prototypen irrelevant)	Ja (ein Handler auf Frameset-Ebene reicht)	Key-Events kommen nur an, wenn einem Form-Element (Input-Text, Textarea) der Focus verpasst wird	Nein (wenn keine Texteingabefelder vorhanden, bleibt Fokus auf URL-Eingabefeld des Browsers)
Status der CTRL-Taste bei MouseEvent abgefragt	Ja	Ja	Ja

Tabelle 6.2: Anforderungen des Prototyps (Netscape).

Anforderung	NS 4.75 Win32	NS 4.61 Unix	NS 6 PR3 Win32
Sich daraus ergebende Handicaps	Methode calculateFrameDimension bietet keine 100%-ige Pixelgenauigkeit	Methode calculateFrameDimension bietet keine 100%-ige Pixelgenauigkeit	Drag-and-Drop über Frames hinweg geht scheinbar nicht

Tabelle 6.2: Anforderungen des Prototyps (Netscape).

Anmerkung: Bei den Handicaps ist das generelle Netscape-Resizeproblem (siehe Abschnitt 4.3.2) nicht berücksichtigt.

Gerade bezüglich Maus-Events während einer Drag-Operation ergaben sich gewaltige Unterschiede: Im Internet Explorer 5 am Mac und im Netscape Navigator 6 wurden in anderen Frames keinerlei Events generiert, im IE 5.0 unter Windows und im Navigator 4 bezogen sich die Koordinaten nicht auf den aktuellen, sondern auf den ursprünglichen Frame. Beim IE 4 schließlich gelang es nur durch Setzen des Focus auf den gerade überquerten Frames, richtige Koordinaten zu bekommen.

Die Probleme von IE 5 und Navigator 4 konnten durch Ermitteln der Frame- und Maus-Positionen, jeweils auf den Bildschirm bezogen, gelöst werden. Abbildung 6.1 zeigt das Prinzip dieses Verfahrens. Das bedingt beim Internet Explorer, dessen Window-Objekt die Properties screenX und screenY nicht kennt, dass man die Maus schon einmal über den Frame bewegt hat, bevor man eine Drag-Operation dorthin beginnt.

6.3.3 Zu testende Plattformen

Schon in der Spezifikation wurden die zu unterstützenden Browser angeführt – hier noch einmal in einer Tabelle dargestellt:

Browser	Windows 95/98/NT/2000	Unix-Derivat	MacOS
Internet Explorer	4, 5.0, 5.5	–	5.0
Netscape Navigator	4.7x, 6	4.7x, 6	4.7x, 6

Tabelle 6.3: Zu testende Plattformen.

Zu revidieren ist die Unterstützung für den Internet Explorer am Mac. Da dieser Browser von anderen Web-Applikationen der Firma Hyperwave nicht unterstützt wird (und sich schon bei den Prototypen bzw. bei den laufenden Tests der Bibliothek zeigte, dass diese Version ihre speziellen Beschränkungen hat), wurde er aus der Liste der zu bedienenden Plattformen gestrichen.

Am Schluss der Entwicklungsphase kam kurz die Frage auf, wie weit eine Unterstützung

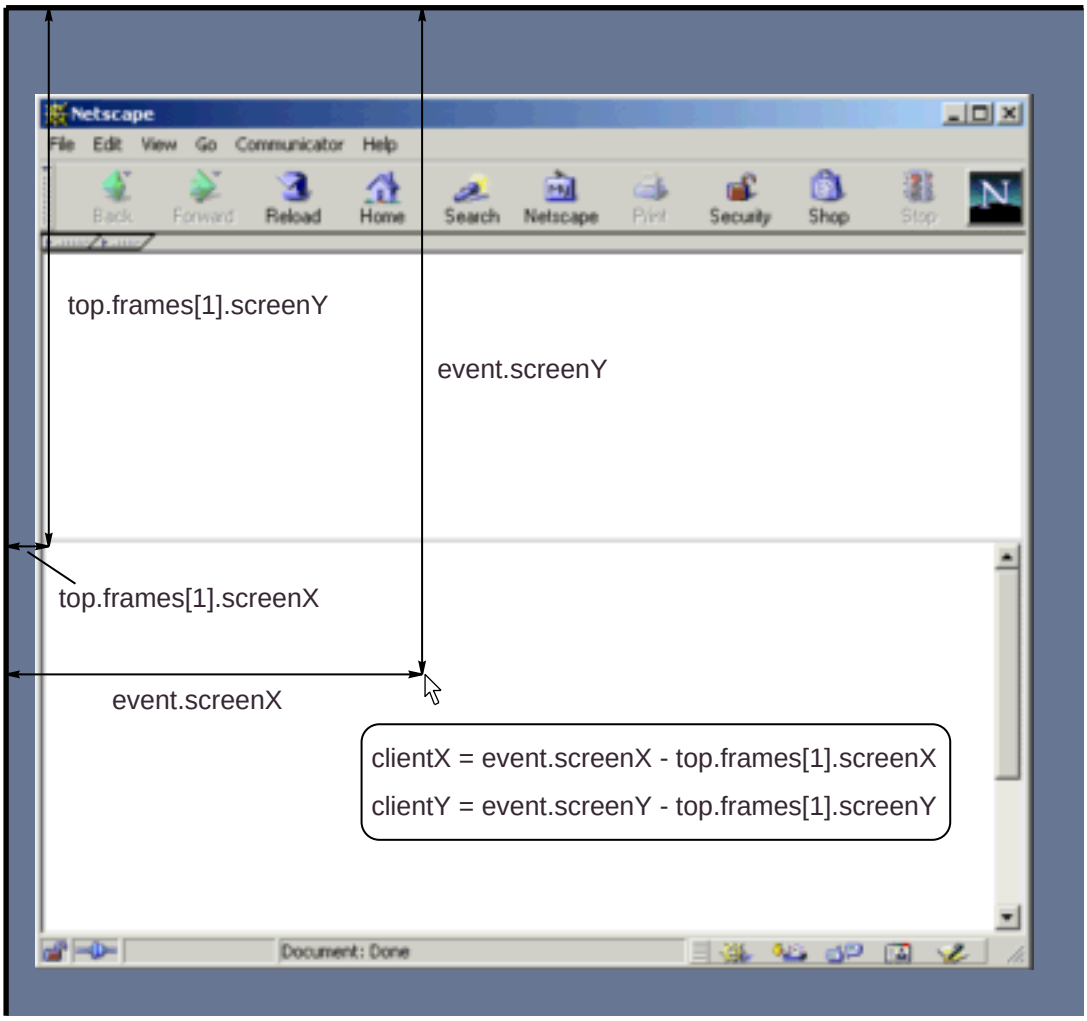

Abbildung 6.1: Work-Around eines Drag and Drop Event-Problems.

von Operas 5er-Browser möglich ist. Allerdings deuteten Tests darauf hin, dass dieser Browser immer noch – auch wenn er schon Teile des DOM Level 1 anbietet – Probleme mit JavaScript hat und deshalb wurde er im Rahmen dieser Arbeit nicht weiter betrachtet.

6.3.4 Kritische Punkte

Die als besonders kritisch (und demnach sehr zeitaufwendig) identifizierten Bereiche wurden am Beginn der Architectural Design Phase wie folgt aufgelistet:

- Event-Handling über Frames hinaus (v.a. bei Drag-and-Drop)
- Resize in Netscape

- Struktur (Design Patterns implementieren)
- Keyboard-Events (v.a. unter Unix)

Dazu ist im Nachhinein anzumerken, dass der vierte Punkt nicht mehr in die Zuständigkeit des JavaScript AWT's fällt und somit nicht behandelt wurde (dies fällt eher in die Zuständigkeit einer Formular-Abstraktion).

Als besonders kritisch entpuppte sich – wie auch nicht anders erwartet – das Problem des Resizes im Netscape Navigator Version 4: Das Problem kann zwar einfach mittels onresize = reload gelöst werden, diese Lösung ist jedoch unbefriedigend, da sie den Zustand der Applikation nicht wieder herstellt (und außerdem u.U. einen vermeidbaren Request absetzt). In einem Frameset kann trotz Reloads der Status wieder aufgebaut werden, da Objekte im top-Scope erhalten bleiben, in einer Frame-losen Seite funktioniert dies jedoch nicht.

Anmerkung. Zum Zeitpunkt der Fertigstellung der schriftlichen Arbeit stand noch nicht fest, ob die gewählte Lösung – die versucht, auch in Frame-losen Seiten ohne Reload auszukommen – 100%-ig stabil laufen kann. Im momentanen Status kam es bei der Nutzung von Event-Handling auf Layer-Basis nach wenigen Resizes zu Abstürzen von Netscape. Dies wäre eine nicht tragbare Situation, die immer den Einsatz von Frames als Alternative erfordern würde.[4]

6.3.5 Rückblick zu den Anfängen

Während einer langen Planungsphase kann es leicht passieren, dass anfängliche Ziele, Motivationen oder Ideen aus dem Blick geraten und man sich in falsche Richtungen bewegen beginnt. Um dem vorzubeugen, sollte man sich immer wieder der Wurzeln erinnern. Der nachfolgende Abschnitt, betitelt mit "Warum?", stellt einen solchen Rückblick im Laufe des Designs der JavaScript-Bibliothek dar.

Warum?

Eine wichtige Frage, die man sich während der Entwicklung immer wieder stellen sollte, ist: "Weshalb wurde dieses Projekt eigentlich vor Monaten ins Leben gerufen?" Bei einem solchen Rückblick werden dann wieder Schwerpunkte und Anwendungsfälle klar, die u.U. aus dem Auge verloren wurden.

Nun, warum wurde die Idee eines solchen JavaScript AWT's geboren? Verschiedene Motivationen führten dazu:

- Bedarf nach dHTML mit einheitlichem DOM-Interface
- Mangelhafte DOM-Manipulations-Möglichkeiten bei Netscape 4

[4] Die bei der Firma Hyperwave genutzte Technik der Kommunikationskanäle via JavaScript-Protokoll verlangt ohnehin die ständige Verwendung von Frames. Hier käme es zu keinen Einschränkungen.

- Resize-Problem bei Netscape 4
- Layout-Problem bei Netscape 4

Als sich die Gedanken zu diesem AWT konkretisierten, wurde klar, dass einige weitere interessante Dinge damit realisierbar sein würden:

- Layout Management
- Treeview bzw. Listview (oder andere Komponenten mit komplexer innerer Struktur)
- Drag-and-Drop (sogar über Frames hinweg)

Blickt man in die weitere Zukunft, so könnten aufwendigere Widgets ebenfalls auf dieser Technik aufbauend entwickelt werden. (Beispiel: Eine selbst programmierte Listbox mit Scrollbars und mehrfachem Selektieren.) Oder: Das Layout Management kann benutzt werden, um DIV's einer HTML-Seite, die sonst klassisch in einem HTML-Editor geschrieben ist, auszurichten.

Bei einem solchen Ausblick stellt sich die Frage, wie viele der Architektur- und Design-Entscheidungen hinfällig werden, wenn die Generation 4 der Netscape Browser kaum mehr benutzt wird? In erster Linie führten ja gerade die Mängel der dHTML-Fähigkeiten dieser Browser zur Entwicklung der besprochenen praktischen Arbeit.

6.3.6 Use Cases

Einige zentrale (und sehr rudimentäre) Use Cases wurden im Laufe der Architectural Design Phase herausgearbeitet:

Treeview

Das Beispiel einer Baumansicht ist eine gute Übung für das Umsetzen des angestrebten Model View Controller-Prinzips. Gerade hier kommt es schnell zur Verwebung von Daten, Darstellung und Applikations-Logik.

Der Entwickler, der eine Treeview verwendet, will auf ein Default-Rendering sowie Default-Verhalten zurückgreifen können. Wenn er besondere Wünsche hat, so soll er einfach die Darstellung beeinflussen können (View). (Beispiel: Je nach Dokument-Typ sollen verschiedene Icons vor einem Blatt im Baum gezeichnet werden.) Die Verknüpfung mit der Datenquelle muss standardisiert und möglichst unproblematisch vonstatten gehen (Model). Und Hooks – z.B. selbst definierte Events (onNodeOpened, onNodeClosed) – müssen das Eingreifen in die Logik des Systems erlauben (Controller).

Drag-and-Drop

Verschiedene Komponenten sollen durch `setDragSource(true)` mit der Maus aufnehm- und verschiebbar werden. Wird ein solches Objekt über einem anderen losgelassen, das mit der

Eigenschaft isDropTarget() versehen ist, so wird ein Drop-Event ausgelöst, auf den DragSource und Drop-Target reagieren müssen. Wenn das DragSource-Element ein DropTarget erreicht, sich darüber befindet bzw. es verlässt, sollen beide Komponenten die Events "DragEnter", "DragOver", "DragExit" bekommen.

Bei der Implementierung ist darauf zu achten, dass nicht jede Komponente neu die entsprechenden Listener-Methoden instanziert. Die View-Komponente (Icon neben Mauszeiger, in verschiedenen Situationen) muss auf jeden Fall ein Standard-Rendering besitzen und customizierbar sein.

"Fertiges HTML" mit Dynamik versehen

Will ein Designer HTML-Elemente nicht scripten, sondern z.B. in seinem HTML-Editor entwerfen, und trotzdem Funktionalität des JavaScript AWT nutzen, so muss es einfach möglich sein, Komponenten des AWT aus bestehenden Objekten des DOM's zu erstellen (via Konstruktor oder via Funktion).

Das folgende Beispiel soll zeigen, wie so etwas aussehen kann:

```
1  <html>
2    <head><title>Beispiel</title>
3    </head>
4     // ... Einbinden von Bibliothek
5
6    <script>
7      function createObjects() {
8        aComponent = new JSComposite(document.all["aDIV"]);
9        anotherComponent = new JSComposite(document.all["anotherDIV"]);
10       aComponent.moveTo(100, 100);
11        // ... weitere AWT-Funktionen
12     }
13   </script>
14
15   <div id="aDIV" style="position:absolute">Test DIV 1</div>
16   <div id="anotherDIV" style="position:absolute">Test DIV 2</div>
17
18   <body onload="createObjects()">
19     <h1>Standard HTML</h1>
20     <p>...</p>
21   </body>
22 </html>
```

Anmerkung: Dieses Beispiel zeigt trotz seiner Einfachheit schon das übliche Problem der Browser-Inkompatiblität, denn es wäre in Netscape nicht lauffähig – beim Verweis auf die DIV's wäre document.layers["aDIV"] statt des IE-spezifischen document.all zu benutzen.

6.3.7 Design-Entscheidungen

Von der Reihe von Design-Entscheidungen, die zu treffen waren, seien die anschließend angeführten in dieser Arbeit festgehalten:

Netscape mit ausgeschaltetem Cache – in diesem Fall ist generell ein Reload notwendig (auch wenn das Verhalten von Frameset und Frame-loser Seite unterschiedlich ist).

Drag-and-Drop mit nicht selbst kontrollierten Frames – hier ist keine Implementation möglich, bei der das Drag-Icon auch über dem fremden Frame sichtbar ist (Security-Einschränkungen erlauben es nicht, Event-Handler in Frames von einem anderen Server zu installieren).

Markierung von Elementen – Frame soll "Markierungs-Scope" bilden. D.h. wird innerhalb eines Frames nach einer Markierung ohne gedrückte CTRL-Taste ein weiterer Klick getätigt, so wird das Element, über dem dieser erfolgte (falls isSelectable() == true gilt) markiert. Alle anderen Elemente in diesem Frame, die vorher markiert waren, verlieren ihre Markierung. Ein Maus-Klick in einem anderen Frame soll bewirken, dass die Markierung im alten Frame gespeichert bleibt aber ausgeblendet wird, und beim nächsten Klick in diesen Frame wieder angezeigt wird. (Dieses Verhalten ist dem Windows-Explorer entnommen und sollte dem Benutzer vertraut sein.)

"Fertiges HTML" mit Dynamik versehen – um das Netscape Resize-Problem zu handhaben, muss in so einem Fall immer auch der Frame bzw. das Frameset mit dem passenden Handler versehen werden (entspricht einer Teilfunktionalität des Frame-Objekts).

Kein <script> innerhalb des HTML-Contents von Glyphen – dank Listener-Prinzip und schachtelbarer Komponenten sollte das auch nicht notwendig sein, denn damit kann jegliche dHTML-Funktionalität abgebildet werden (vgl. Beispiel!?). Ein solches Eingreifen in versteckte Strukturen wäre auch alles andere als sauber.

Event-Modell – generell soll dabei konsequent auf einem Observer-Prinzip aufgebaut werden. Das bedeutet, das keine native onEvent-Handler mehr verwendet werden sollen, sondern man sich bei Notwendigkeit als Listener beim entsprechenden Objekt registriert. Damit gibt es auch kein automatisches "Event-Blubbern" oder "Event-Sickern" mehr.

6.3.8 Rational Rose

Nach anfänglichem Arbeiten mit einem Texteditor und LaTeX-Dokumenten entstand bald der Wunsch, ein Werkzeug zur Verfügung zu haben, das für Konsistenz und bessere Übersicht im Design sorgen kann. Die Wahl fiel auf Rational Rose[5], dessen Professional J Edition 2000 benutzt wurde.

[5] Siehe [4].

Einige Tage wurde auch Together J von TogetherSoft[6] getestet – ein Entwicklungswerkzeug, das unter Java Swing läuft und allseits viel gelobt wird. Allerdings bot es einen äußerst sturen Auto-Layout-Modus, der die Klassendiagramme über mehrere Bildschirmseiten verteilte, auch wenn sie nur wenige Objekte enthielten.

Aus zwei Gründen viel die Wahl auf die Java-Code erzeugende Version von Rational Rose und nicht auf jene für C++.

1. Die Syntax von Java entspricht viel eher jener von JavaScript als es C++ tut.
2. Die Erzeugung von Java-Code ermöglichte es, automatisch ein JavaScript-Gerüst erstellen zu lassen.

Auf den zweiten Punkt wird im nächsten Abschnitt näher eingegangen.

Die Bedienung von Rational Rose Professional ist recht intuitiv. Die vielen verschiedenen Diagramm-Typen erlauben vielfältiges Designen. Doch gerade dadurch muss der Benutzer genau wissen, was und auf welchem Abstraktionslevel er entwerfen will. Martin Fowler und Kendall Scott beschreiben diesbezüglich in [30, S.44f] drei Sichtweisen für Klassendiagramme:

Konzeptionell In diesem Abstraktionslevel werden nur die wesentlichen Beziehungen der Problembereiche behandelt, was nicht den Beziehungen der implementierenden Klassen entsprechen muss. Diese Darstellung sollte Software-unabhängig sein.

Spezifizierend In dieser Sicht werden nur Schnittstellen betrachtet, keine Implementierungen. Dieses stark dem OO-Ansatz entsprechende Modell wird leider oft nicht im Auge behalten, und viel zu oft wird nicht gemäß Interfaces sondern gemäß Klassen programmiert.

Implementierend Auf diesem Level wird mit Klassen gearbeitet. Diese Sichtweise wird häufig eingenommen (vielleicht allein schon wegen der Bezeichnung "Klassendiagramm"), allerdings wäre meist die spezifizierende Sicht die bessere.

Fowler und Scott schreiben: "Das Verstehen der Sichtweise ist sowohl beim Zeichnen als auch beim Lesen von Klassendiagrammen wichtig. Leider ist die Abgrenzung der Sichtweisen unscharf, und die meisten Modellierer achten nicht auf die unterschiedlichen Sichtweisen." Besonders die Unterscheidung von Spezifikations- und Implementierungssichtweise sei sehr wichtig.

Auch die hier betrachtete Arbeit gelangte sehr rasch in einen hohen Detaillevel, und so wurden viele Klassen schon mit Methoden versehen, bevor die grundlegenden Beziehungen fest standen. Eine allgemein spezifizierende Sichtweise wäre für den Anfang sicher angebracht gewesen.

Als wirkliches Manko von Rational Rose muss die leicht auftretende Inkonsistenz zwischen Darstellung und Modell (das aber eigentlich nur auf den Diagrammen basiert)

[6] Produktinformationen und Downloads unter [6].

erwähnt werden. So passiert es schnell, dass Relationen wie "implements" oder "extends" nicht im Diagramm zu erkennen sind, diese aber bei Inspektion der Klasseneigenschaften zu sehen sind – und somit im logischen Modell bestehen.

Abgesehen davon war die Benutzung eines solchen Tools eine große Hilfe und die automatische Code-Generierung eine wirkliche Beschleunigung.

6.3.9 Doclet API

Rational Rose bietet wie viele Design-Tools die Möglichkeit, ein Code-Gerüst generieren zu lassen. Dies beinhaltet die Interface-, Klassen-, Methoden- und Felddefinitionen samt JavaDoc-konformen Kommentaren. Dieser erste Schritt in Abbildung 6.2 erzeugt jedoch keine wirklich hundertprozentig sauberen Kommentare, sodass eine selbst geschriebene Applikation kleine Korrekturen vornehmen musste.

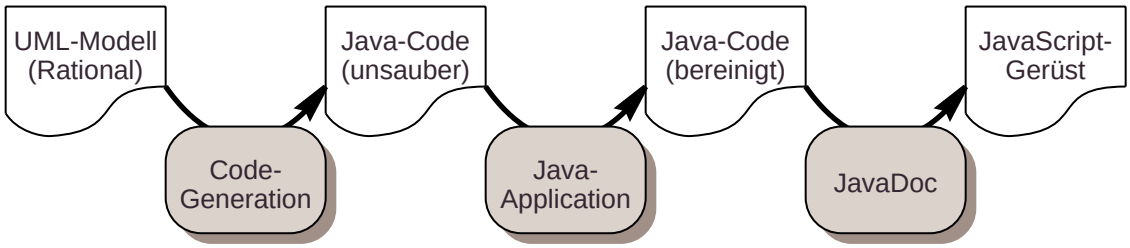

Abbildung 6.2: Weg vom UML-Modell zum JavaScript-Code.

Aus diesem nun sauberen Java-Sourcecode kann mittels JavaDoc[7] die gewohnte HTML-Dokumentation erzeugt werden. Doch nicht nur diese – das Doclet API ermöglicht das Erstellen von beliebigen Dokumenten, aufbauend auf der von JavaDoc geparsten Information[8].

Zwei eigene Doclets wurden geschrieben: Eines, um LaTeX-Dokumentation in Form von Tabellen zu erzeugen (das Ergebnis ist im Architectural Design Document zu sehen); ein zweites, um das JavaScript-Klassengerüst zu erhalten. Dieses musste den im Rahmen des Hyperwave Application Frameworks definierten Style-Guidelines entsprechen, was recht gut gelang.

Um jedoch das Design-Modell nach Änderungen des Codes während der Implementierung mit dem neuen Zustand konsistent zu halten, wäre ein Backward-Engineering Mechanismus notwendig. Für diese Aufgabe kann jedoch nicht mehr das Doclet API genutzt werden, sodass die JavaScript-Dateien selbst durchgeparst werden müssten, um daraus Java-Code zu erzeugen. Dies wurde im Rahmen dieser Arbeit nicht mehr getan. Allerdings existiert in der Firma Hyperwave ein Script, um für HAF-Style-Guidelines entsprechende JavaScript-Klassen eine HTML-Dokumentation zu erzeugen.

[7] Siehe [19].
[8] Siehe [18].

6.4 Architectural Design Document

Da das Architectural Design Document in Anhang A beigefügt ist, soll hier nur noch in kurzen Zusammenfassungen auf die Ergebnisse eingegangen werden. Begonnen wird dabei mit dem Klassendesign, welches den größten Teil des Dokuments ausmacht.

6.4.1 Klassendesign

Kompakt und anschaulich (weil gerade noch gut zu überblicken) ist in diesem Zusammenhang das Klassendiagramm, wie es von Rational Rose erstellt wird. In Abbildung 6.3 ist es wiedergegeben.

Einzelne Gruppen von Klassen sollen zum besseren Verständnis kurz erläutert werden:

Components Kern des Modells sind die von CSComponent abgeleiteten Klassen, die ein Composite-Muster bilden (siehe Abschnitt 5.2.1). Eine spezialisierte Erweiterung der CSComposite-Klasse bildet CSContentPane – dieses Objekt repräsentiert window.document aus dem herkömmlichen dHTML-Modell.

Events Eine abgeschlossene Gruppe bilden die Event-Objekte, welche von CSEventSource-Objekten erzeugt und an ihre Observer geschickt werden. Vor allem die Maus-Events beinhalten eine große Menge an getter- und setter-Methoden für all ihre Properties (Mauskoordinaten, Status der Maustasten bzw. gewisser Modifier-Keys).

Layout Manager Jeder CSComposite besitzt genau einen Layout Manager, der je nach Anforderungen zu wählen ist. Als Default Manager wurde ein Null-Layout Manager gewählt, der die Objekte an den von ihnen fixierten Stellen mit der von ihnen gewünschten Größe darstellt.

Browser Abstraction Factory Den Kern der Browser-Abstraktion bildet die Klasse CSBrowserAbstractionFactory, welche als Abstract Factory je nach Browser-Typ die passende Low-Level-Klasse für Layer- oder Event-Handling instanziiert (siehe dazu auch secrefAbstrFactDP.

Fremdklassen Die beiden grau hinterlegten Klassen wurden nicht neu spezifiziert sondern bestehen bereits seit längerem im Rahmen der Hyperwave eLearning Suite.

6.4.2 Erläuterung der Zustände

Dieser Abschnitt des Architectural Design Documents befasst sich mit dem Problem der Synchronisation zwischen logischem und physikalischem Komponenten-Modell.

Sobald eine Operation z.B. die Position, Größe oder Sichtbarkeit einer Komponente verändert, geschieht dies zuerst im logischen Modell. Die betroffene Komponente sowie seine Eltern-Hierarchie wird in diesem Fall "invalid" gesetzt. Erst der Aufruf von show() sorgt für eine entsprechende Änderung des physikalischen Modells (dies geschieht in der paint-Methode).

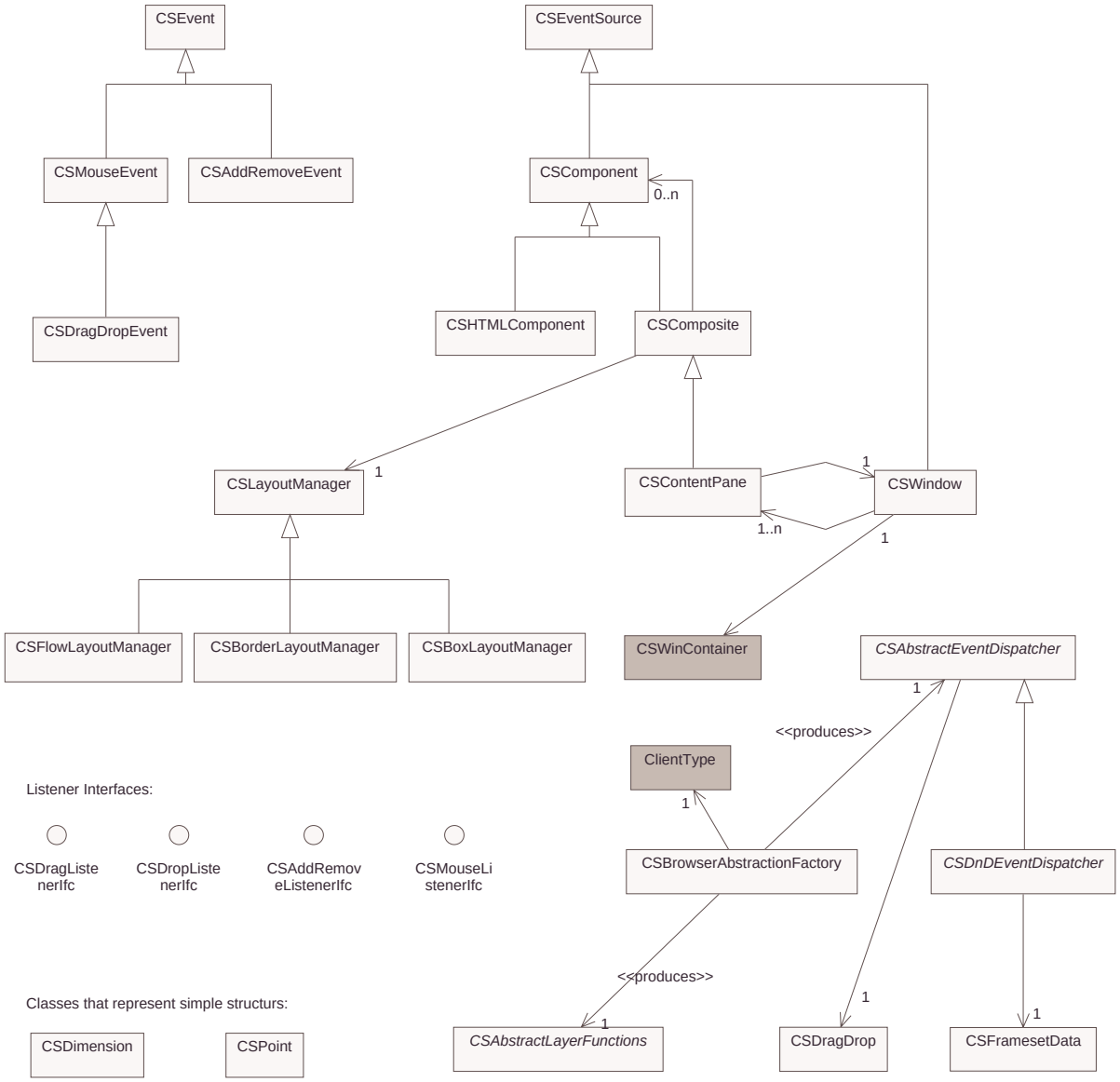

Abbildung 6.3: Klassenhierarchie laut Architectural Design Document.

Um unnötige Operationen zu vermeiden, werden nur jene Objekte, die “invalid” gesetzt sind, neu gezeichnet. Eine weitere Ersparnis bietet das Speichern der errechneten Größe eines Containers. Dieser Wert ändert sich ebenfalls, wenn an einem der Kinder eine invalidierende Änderung vorgenommen wird. Allerdings reicht es aus, den neuen Wert der gewünschten Größe einmal pro Layout-Vorgang zu berechnen – um dies sicherzustellen gibt es das Flag knowSize.

6.4.3 Event Sources

Alle von CSEventSource abgeleiteten Klassen offerieren die Möglichkeit, sich als Listener bei ihnen zu registrieren. Die dazu notwendigen Methoden wurden im Rahmen des Klassendesigns definiert – was offen bleibt ist, welche Events nun von welcher Quelle erzeugt werden. Dies beschreibt Abschnitt A.4 des ADD.

Allerdings wurde diese Zuordnung im Laufe der Implementierung abgeändert, so generiert die Klasse CSWindow nur noch Resize-Events, CSContentPane jedoch auch Load- und Unload-Events.

6.4.4 Drag and Drop

Abschnitt A.5 erklärt an einem einfachen Beispiel, wie Drag and Drop prinzipiell implementiert werden soll. Dies geschieht jedoch schon auf einer sehr detaillierten Ebene und würde eher in ein Detailed Design (das es explizit nicht mehr gegeben hat) passen.

Von der generellen Initialisierung über den Beginn einer Drag-Operation bis hin zum Drop wird anhand von Code-Fragmenten gezeigt, welche Operationen und Events auftreten. Einen sehr schönen Überblick bietet dazu das Activity Diagramm in Abbildung A.5.

6.4.5 Erzeugung eines Framesets

Abschließend geht das ADD noch darauf ein, wie die konkrete Erzeugung eines Framesets aussehen soll. Der dabei gezeigte Ansatz über eine Redirecter-Page wurde bei der Implementierung beibehalten, jedoch ein wenig modifiziert.

6.5 Implementierungsphase

6.5.1 Erkenntnisse während der Implementierung

Einige interessante Dinge sind erst bei der Codierung und dem Klassen-Testen aufgefallen. Nicht alles wurde dokumentiert – so bleibt der folgende Auszug (der sich auf den Navigator beschränkt) unvollständig.

Reload in Resize-Handlern

Wie schon weiter oben angeführt, muss unter Netscape nach einem Resize des Fensters bei der Verwendung von Framesets jeder einzelne Frame neu geladen werden. Allerdings zeigte der Aufruf der `reload` Methode den Seiteneffekt, dass dadurch die anderen Frames keinen Resize-Event mehr bekamen.

Als Lösung sorgt jeder Resize-Handler dafür, dass alle Frames neu geladen werden. Da jeweils nur ein solch ein Handler wirklich auch den Resize-Event bekommt, werden mehrfache Reloads von selbst vermieden.

document.close-Seiteneffekt

Das Beschreiben der Layer erfolgt in Netscape 4 über die Methoden document.write bzw. document.writeln, gefolgt von einem document.close. Letzteres löst – was vielleicht nicht gleich offensichtlich ist – einen Load-Event aus. Dies macht durchaus Sinn und kann auch für das Installieren von Event-Handlern genutzt werden.

Allerdings fingen in den ersten Implementation die für die Erstellung der Layer notwendigen Load-Handler des window.document Objekts diesen Event ab (auf Grund des Sickerns der Events unter Netscape konnte dies gar nicht verhindert werden) und reagierten darauf. Somit musste dieser Spezialfall gefiltert und abgefangen werden.

6.5.2 Known Problems

Zum Zeitpunkt der Fertigstellung dieser Arbeit befand sich das JavaScript AWT im Alpha-Stadium. Es wurde noch niemals von jemandem anderen als dem Autor benutzt und wurde somit auch nicht in allen Bereichen und Situationen getestet. Einige bis dahin bekannte Probleme und Verbesserungsmöglichkeiten seien im folgenden aufgeführt:

- Resizes in Frame-losen Seiten unter Netscape 4 führen bei aktiviertem Maus-Event-Handling zu baldigen Abstürzen (siehe auch Abschnitt 6.3.4). Es gilt noch, die Ursache dieses Problems aufzuspüren. Mögliche Quellen sind das Verwerfen von Layern mit Event-Handlern oder ein Speicher-Problem von Netscape, da bei Maus-Events der Speicher stark ansteigt und der Garbage Collector von Netscape recht unsauber arbeitet.

- Unter Netscape 4 führen Verkleinerungen der Größe der Content Pane bei darin enthaltenen Layern mit Formularen als HTML-Inhalt zu Darstellungsproblemen bei den Formularen. Und zwar dann, wenn sich zum Zeitpunkt der Änderung ein Formular im nicht sichtbaren Scrollbereich befand. Dies ist ein Repaint-Problem Netscapes, das vermutlich durch erneutes Beschreiben des Layers gelöst werden kann.

- Ein weiteres Problem mit dem Verkleinern von document.height bzw. document.width besteht unter Netscape 4 auf MacOS[9]. Auf dieser Plattform reagiert die Rendering Engine des Navigator nicht auf das Verkleinern der Werte, auch wenn diese wirklich verändert werden. Dieses Problem könnte entweder hingenommen werden (wichtig ist vor allem, dass das Vergrößern funktioniert) oder durch ein Reload beim Verändern der Größe behoben werden.

- Etwas problematisch ist auch die neueste Version von Netscape. Mozilla 0.8 erlaubt nicht, wie nach CSS2-Definitionen erwartet, das Scroll-Verhalten des Dokuments

[9] Zu diesem Thema muss angemerkt werden, dass das Verändern dieser beiden Properties des Document-Objekts nirgends erwähnt wird. [11] z.B. beschreibt diese Felder nicht explizit als read-only, spricht aber nur vom Auslesen der Werte.

zu regeln. Es kommt zu großen Abhängigkeiten vom via HTML gesetzten scrolling-Attribut des FRAME-Tags. Außerdem gibt es – noch zu wenig recherchierte – Unterschiede zwischen dem s.g. Quirks- und dem Standard-Modus[10].

- Die Realisierung von Drag and Drop beschränkt sich momentan noch auf Frame-interne Aktionen und ist unter Netscape 4 dort auch nicht 100%-ig stabil. Die Ursache davon muss noch eruiert werden.
- Vorsicht ist bei der Benutzung der pack-Methode der Klasse CSWindow geboten. Diese Methode ist grundsätzlich nur erfolgreich, wenn in einem Frame-losen Fenster aufgerufen. Doch darüber hinaus ist sie bei Content Panes mit Scrollbars nicht empfehlenswert und erzeugt beim Internet Explorer generell nicht exakte Ergebnisse[11].
- Die Methode, mit der die Layout-Manager überprüfen, ob sich eines ihrer Kinder an einer gewissen Stelle befindet, ist nicht optimiert, sondern wird übernommen von der Version des Flow-Layouts. Bei vielen Maus-Events (z.B. im Fall von Drag and Drop) und vielen Kindern kann eine Optimierung der Algorithmen eine Performance-Steigerung bewirken.

An der Behebung dieser Missstände wird weiter gearbeitet – zusätzlich dazu entstehen eine Beschreibung (Tutorial etc.) sowie eine verbesserte Dokumentation der Klassen.

6.6 Das Ergebnis

Nach ca. 190 Stunden Arbeit an den Klassen (exklusive Erstellung von Beispielen und Tests) wurde das Projekt als reif genug für eine Alpha-Phase befunden und in diesem Status dem Abschluss der Diplomarbeit zugrunde gelegt.

In Zusammenhang mit dieser Zahl ist interessant, dass am Ende der Design-Phase eine Aufwandsschätzung erfolgte, die eine Summe von 110 Mannstunden ergab. Ein Vergleich mit den geschätzten Zeiten zeigt vor allem bei folgenden Klassen große Diskrepanzen: CSContentPane (36 statt 4h), CSWindow (20 statt 6h), CSHTMLComponent (20 statt 1h) und CSEventDispatcherIE (16 statt 3h). Hier wurde der Aufwand offensichtlich enorm unterschätzt. Bei den ersten drei Klassen lag die zusätzliche Arbeit in den dort ebenfalls vorhandenen Browser-Unterschieden, die beim Design nicht bedacht wurden (die ersten Überlegungen beinhalteten Bowser-Unterschiede nur in den Bereichen Layern und Events).

6.6.1 Erfüllung der Software Requirements

Im folgenden soll überprüft werden, ob die gestellten Anforderungen (siehe Abschnitt 6.2.1) erfüllt wurden:

[10] Der Quirks-Modus, der bei nicht definiertem DTD zum Einsatz kommt, hat ein anderes Rendering-Verhalten.

[11] Dies liegt an der Tatsache, dass die Paramter des Aufrufs resizeTo die Größe des gesamten Fensters (inklusive Adressleiste oder Menüs) beeinflusst.

SR.1 JavaScript
Dieser Punkt kann als vollständig erfüllt betrachtet werden. Sowohl die geforderte Unterstützung gängiger Browser als auch die Vorbereitung auf neue Versionen wurde implementiert.

SR.2 Komponentensystem
CSComposite und ihre Subklassen entsprechen dem Glyphenprinzip und sind schachtelbar.

SR.3 Layout Management
Die Austauschbarkeit der wie gefordert funktionierenden Layout Managern ist gewährleistet. Zusätzlich zum geforderten Box-Layout wurden das Null-, Flow- und Border-Layout implementiert.

SR.4 Event Handling
Das Event-Handling wurde einfach erweiterbar gehalten und basiert auf dem Observer-Mechanismus.

SR.5 Drag-and-Drop
Zumindest innerhalb eines Frames funktioniert dies wie gefordert.

SR.6 HTML-Rückwärtskompatibilität und Ressourcen
Ein entsprechendes Setzen von Style Sheets ist möglich, diese sollten jedoch auf Formatierungen verzichten, die Position oder Größe des Objekts beeinflussen, wenn nicht gleichzeitig auch das logische Modell verändert wird. Im allgemeinen ist im entstandenen AWT das Beeinflussen der Komponenten über die gebotenen Methoden anzuraten.

SR.7 Umgehung von Browserinkompatibilitäten
Eine in diesem Punkt geforderte Handhabung wurde mit dem Strategy-Pattern gelöst. Allerdings wurde bis zum jetzigen Zeitpunkt keine saubere Lösung für den Internet Explorer unter MacOS implementiert.

6.6.2 Einfaches Anwendungsbeispiel

Ein einfaches Beispiel soll den Einsatz von Komponenten, Layout-Managern und Event-Handlern zeigen. Es handelt sich dabei um ein Frame-loses Fenster mit drei Bereichen, die jeweils durch ein CSComposite-Objekt gebildet werden. Auch wenn dieses Beispiel bei weitem nicht die volle mögliche Funktionalität des JavaScript AWT's zeigt, so kann es doch einen Einblick in seine prinzipielle Nutzung geben. Abbildung 6.4 zeigt, wie das Endprodukt aussehen wird.

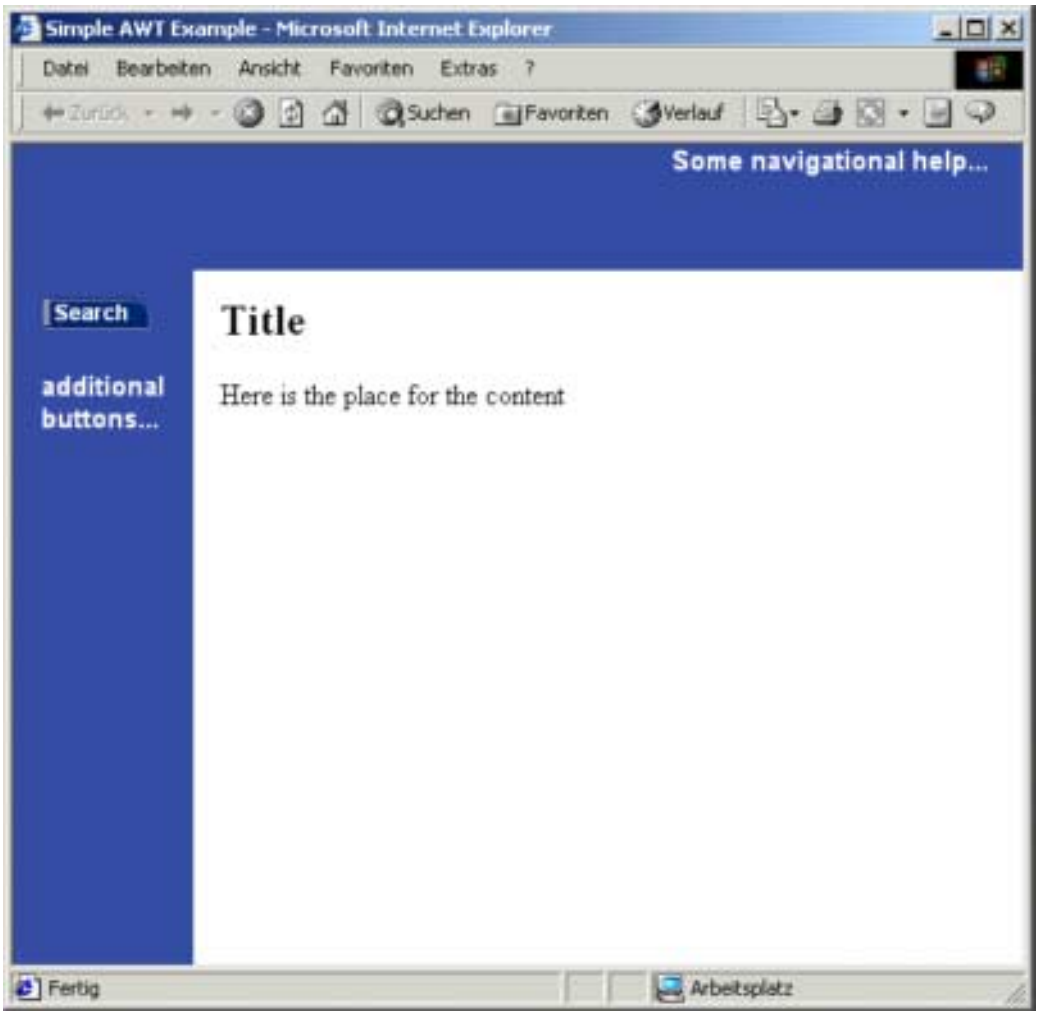

Abbildung 6.4: Einfaches Beispiel für die Nutzung des AWT.

Initialisierung

Im Rahmen des HAF wird auch ein Classloading-Mechanismus eingeführt, der jedoch zum behandelten Zeitpunkt noch nicht integriert wurde. Somit müssen die verwendeten Klassen manuell eingebunden werden. Das folgende Listing zeigt, wie dies aussieht (im Beispiel umfasst die Liste der eingebundenen Klassenfiles mehr als 20 Dateien):

```
1 <html>
2   <head><title>Simple AWT Example</title>
3   <script src="awt/CSAbstractLayerFunctions.js"></script>
4   <script src="awt/CSBrowserAbstractionFactory.js"></script>
5   <script src="awt/CSLayerFunctionsIE.js"></script>
```

Angeregt wird der Aufbau der Layer durch einen Load-Handler im BODY-Tag des Dokuments. Dieser ruft die Methode loadHandler auf, welche wir uns wieder genauer ansehen:

```
1  function loadHandler() {
2    var cs_event = new top.CSEvent("Load", window);
3    if ( !top.contentPane.checkListener(top.contentPane, "Load") )
4      top.contentPane.addListener(top.contentPane, "Load");
5    top.contentPane.fireEvent(cs_event);
6  }
```

Zeile 2 erzeugt ein CSEvent-Objekt, das den Abschluss des Ladens des Dokuments signalisieren wird. In Zeile 4 wird, wenn nicht schon passiert, die Content Pane selbst als eigener Listener registriert. Abschließend wird oben erzeugter Event abgeschickt – und damit die onLoad-Methode von contentPane aufgerufen.

Dieses Objekt und alle weiteren müssen jedoch erst erzeugt werden. Wenden wir uns also wieder dem <script>-Block zu.

Erzeugung der Komponenten

Zunächst soll das Content Pane-Objekt angelegt werden. Dies geschieht (am Beginn des <script>-Blocks im Kopfteil des Dokuments) in den folgenden Zeilen:

```
1  var contentPane = new CSContentPane(new CSBorderLayoutManager(0, 0));
2  contentPane.setScrollable(false);
3  var theWin = CSWindow.static_.getInstance();
4  theWin.addContentPane(contentPane);
```

In Zeile 1 wird eine CSContentPane erzeugt, wobei der Konstruktor als Parameter den zu verwendenden Layout Manager gesetzt bekommt. In diesem Fall wollen wir ein Border-Layout benutzen. Zeile 2 definiert, dass dieser Container keine Scrollbars besitzen soll (was für das Border-Layout wichtig ist).

In Zeile 3 wird eine Referenz auf das CSWindow-Singleton erzeugt und anschließend wird die Content Pane dem Fenster-Wrapper hinzugefügt.

Diese Zeilen würden das HTML-File schon ausführbar machen. Allerdings würden wir nur ein leeres Fenster zu sehen bekommen. Wie weitere Komponenten hinzugefügt werden, zeigt das nächste Code-Snippet:

```
1  var nav_status = new CSComposite(null, new CSDimension(500, 70), true,
2    new CSFlowLayoutManager(CSFlowLayoutManager.static_.RIGHT_));
3  nav_status.setBackgroundColor("#334CA3");
4  contentPane.add(nav_status, CSBorderLayoutManager.static_.NORTH_);
5
6  var nav_status_content = new CSHTMLComponent(
7    null, new CSDimension(200, 70), true);
8  nav_status_content.setContent(
```

```
9     "<span class='white_text'>Some navigational help...</span>");
10  nav_status.add(nav_status_content);
```

Diese zehn Zeilen definieren den oberen Bereich des Fensters. Damit die Komponente rechtsbündig ausgerichtet wird, verwenden wir ein entsprechendes Flow-Layout (Zeile 2). Nach dem Festlegen der Hintergrundfarbe wird in Zeile 4 das Composite-Objekt der Content Pane hinzugefügt – und zwar an die Stelle mit der Bezeichnung "Norden".

Für die Darstellung von HTML-Code (oder auch einfachem Plaintext) benötigt man ein Objekt der Klasse CSHTMLComponent. Ein solches wird in den Zeilen 6 und 7 erzeugt, wobei auf die Angabe einer Position verzichtet werden kann (erster Parameter des Konstruktors), da das Objekt ja von einem Layout Manager ausgerichtet wird.

Daraufhin wird der Inhalt der Komponente festgelegt. Dieser besteht aus einem einfachen Text, der via Style Sheets formatiert wird (auf die Style-Definition wird hier nicht näher eingegangen). Nun muss die Komponente nur noch ihrem Container hinzugefügt werden (Zeile 10).

Die weiteren beiden Komponenten werden in diesem Beispiel direkt in die Content Pane eingefügt.

Installieren von Handlern für Objekte in HTML-Komponenten

Der links zu sehende Button "Search" wurde mit der Möglichkeit versehen, auf sein Klicken zu reagieren. Dies ist zwar generell über eine JavaScript-URL möglich, der Ansatz via Click-Handler ist jedoch vorzuziehen. Und dies ist nicht selbstverständlich möglich: Zum einen ist der Layer nicht bekannt, der die Schaltfläche enthält, und zum anderen würde diese Referenz bei einem Wegwerfen des Layers (z.B. nach einem Resize) verloren gehen.

Die Lösung bietet ein spezieller Event, der von CSHTMLComposite's erzeugt wird, wenn ihr Content neu geschrieben (und damit ins DOM eingefügt wird) – der CSEvent mit dem Typ "ContentNew". Im folgenden ist die Erzeugung der linken Komponente angeführt:

```
1   var nav_buttons = new CSHTMLComponent(
2     null, new CSDimension(100, 500), true);
3   nav_buttons.setBackgroundColor("#334CA3");
4   nav_buttons.setContent(
5     "<div class='margin_box'>\
6       <a href='javascript:void 0;'\
7         onmouseover='document.images[0].src=\"images/button_over.gif\"'\
8         onmouseout='document.images[0].src=\"images/button.gif\"'\
9       ><img src='images/button.gif' border=0></a>\
10      <p class='white_text'>additional buttons...</p>\
11    </div>");
12  contentPane.add(nav_buttons, CSBorderLayoutManager.static_.WEST_);
13  nav_buttons.addListener(my_button, "ContentNew");
```

Die ersten Zeilen bieten schon Bekanntes. Aus den Zeilen 7 und 8 geht hervor, dass

dieser Button ein zusätzliche Darstellung für den Fall, dass sich der Mauszeiger darüber befindet, besitzt. Interessant ist die letzte Zeile. Hier wird, wie schon oben einmal beschrieben, ein Listener an einer Event-Quelle registriert. Das Observer-Objekt my_button besitzt eine Methode onContentNew, die jedesmal aufgerufen wird, wenn der Inhalt der HTML-Komponente neu geschrieben wird:

```
1  my_button.onContentNew = function (evt) {
2    evt.getScope().document.links[0].onclick =
3    clickFunction;
4  }
```

Die gewünschte Reaktion wird von der Funktion clickFunction festgelegt[12].

Frameset

Besteht der Wunsch nach einem Frameset, so kann dem leicht entsprochen werden. Zwei Modifikation sind dazu notwendig. Die erste betrifft den Load-Handler: Dieser wird samt dem BODY-Tag entfernt. Stattdessen ist folgende Frameset-Definition hinzuzufügen:

```
1  <frameset rows="*,100" border=0 frameborder=0 framespacing=0>
2    <frame name="frame1" src="redirect.htm?0" scrolling="no">
3    <frame name="frame2" src="footer.htm" scrolling="no">
4  </frameset>
```

Die zweite Änderung besteht darin, das Objekt contentPane in ein Array einzufügen – dieses Array wird im folgende mit contentPane angesprochen.

Die Redirecter-Page ähnelt vom Prinzip jener aus dem ADD (siehe Abschnitt A.6). Allerdings wird hier der selbe Mechanismus wie im Load-Handler im Beispiel ohne Frames angewandt:

```
1  <html>
2    <head>
3      <title>redirect file </title>
4    <script>
5      function loadFunction() {
6        var index = location.search.slice(1);
7        self.setTimeout('\
8          var cs_event = new top.CSEvent("Load", top.frames['+index+']);\
9          top.contentPane['+index+'].addListener(top.contentPane['+index+'], "Load");\
10         top.contentPane['+index+'].fireEvent(cs_event );', 1);
11     }
12   </script>
13 </head>
```

[12] Eine etwas komplexer Version, als die im Beispiel angewandte, basiert auf einer eigene Klasse und der geschickten Nutzung von inner functions, um den passenden Scope zu erhalten.

```
14   <body onload="loadFunction()"></body>
15 </html>
```

Hier erfolgt die Zuordnung des CSContentPane-Objekts zum entsprechenden Frame über den Load-Event. Für eine stabilere Ausführung sorgt die Timeout-Funktion.

6.6.3 Weitere Einsatzmöglichkeiten

Das Beispiel aus Abschnitt Abschnitt 6.6.2 könnte mit Frames oder Tables ebenso realisiert werden. Interessantere Anwendungsfälle sind sich dynamisch verändernde Darstellungsformen wie Listen, Baumstrukturen oder Tabbed Dialogs. In diesen können die unterschiedlichen Layout Manager effektiv zur Anwendung kommen und die Browser-Abstraktion das sonst notwendige unterschiedliche Handhaben des dHTML ersetzen.

Und gerade in solchen komplexen Widgets oder Dialogen ist es wichtig und hilfreich, dem Model View Controller-Prinzip zu folgen. Abbildung 6.5 symbolisiert diesen Ansatz für Tabbed Dialogs.

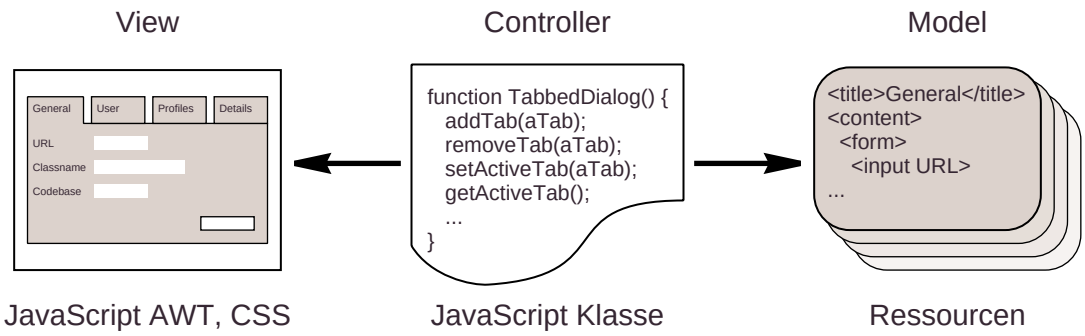

Abbildung 6.5: Model View Controller-Prinzip bei Tabbed Dialogs.

Im Rahmen des HAF-Ressourcenkonzepts entwickelte Beschreibungsmöglichkeiten für Daten wie Texte, Bilder oder ganze HTML-Strings bilden das "Model". Hiermit sind die Inhalte sauber von ihrer Visualisierung getrennt und leicht veränder- bzw. austauschbar. Der "Controller" sei in JavaScript realisiert und sollte unabhängig von den Daten und deren Visualisierung sein. Das Rendering kann durch das JavaScript AWT erfolgen, wobei sich dieses um Events und das Layout-Management kümmert. Schriften und Farben werden via Style Sheets festgelegt.

Die wohl erste konkrete Anwendung des AWT's wird die Implementierung eines Trees sein. Hier wird versucht werden, den oben skizzierten Ansatz umzusetzen. Außerdem wird sich an diesem komplexen Beispiel die "Alltags-Tauglichkeit" des entwickelten Produkts zeigen.

Anhang A

Architectural Design Document

A.1 Klassen-Design

Anmerkung 1: Obwohl es in JavaScript (in den aktuellen Versionen bis 1.5) keine solche Definition gibt, sind im Anschluss an die Klassen auch Interfaces angeführt. Implementiert eine Klasse ein solches, so werden die notwendigen Methoden auch in der Klassenbeschreibung angeführt und dokumentiert. Im Gegensatz dazu werden die von Superklassen geerbten Methoden nicht nochmals aufgelistet, sofern sie nicht überschrieben werden.

Anmerkung 2: Abbildung A.1 beschränkt sich in der Darstellung darauf, jene Klassen, die konkretes Event Dispatching und konkrete Layer Funktionalität zur Verfügung stellen, nicht abzubilden. Selbiges trifft auf die nachfolgende Liste zu – auch dort werden diese Klassen, obwohl schon in UML modelliert, nicht angeführt.

Anmerkung 3: Die beiden Klassen ClientType und CSWinContainer werden in Abbildung A.1 dunkel hinterlegt dargestellt. Damit soll symbolisiert werden, dass diese Klassen nicht Teil der Spezifikation sind, sondern schon im Gebrauch sind.

Anmerkung 4: Die folgende Liste von Klassen-Schnittstellen gibt nicht die Modifikatoren wie static oder abstract wieder, die aber im zugrunde liegenden UML-Modell vorhanden sind. Da diese Einschränkungen jedoch in den Bereich des Detailed Design fallen, wurde darauf verzichtet. (Ob eine Methode protected ist, kann am vorgestellten Underline (z.B. _onEvent()) entsprechend der Style-Guidlines erkannt werden.)

A.1.1 Class CSAbstractEventDispatcher extends CSBase

This abstract classes subclasses handle all the native JavaScript events that occur in a window and its document. They create appropriate CSEvent objects and dispatch them to the objects (component or window) that are listeners for this type of event. For each differing browser (NS4, NS6, IE), a different handler object is instantiated (see the subclasses of this class).

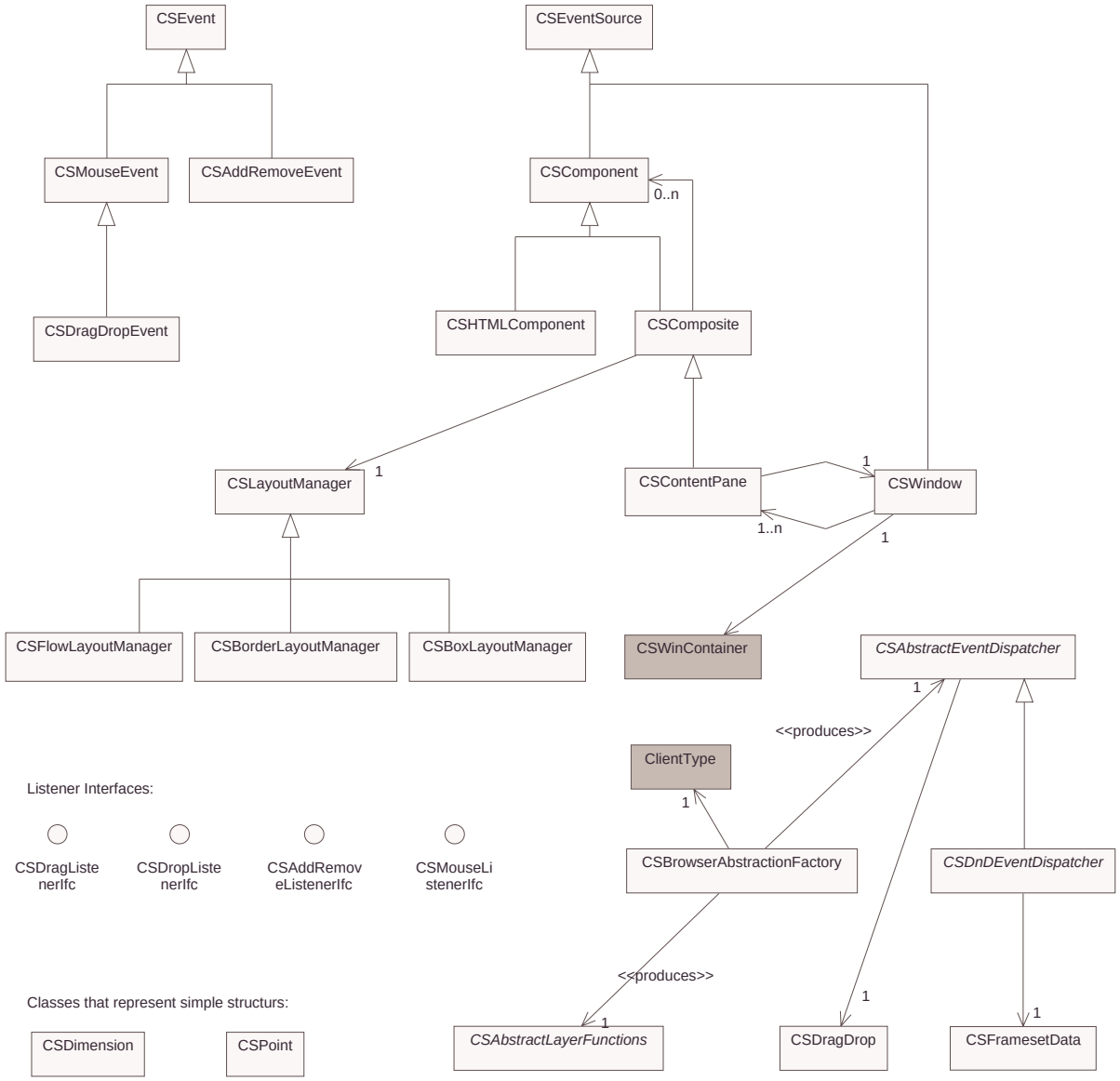

Abbildung A.1: Klassenhierarchie auf Architectural Design Level.

Return value	Method name, parameters and description
void	CSAbstractEventDispatcher() Creates a new CSAbstractEventDispatcher object (this class is abstract and should not be instantiated).

Tabelle A.1: Class CSAbstractEventDispatcher.

Return value	Method name, parameters and description
void	_captureEvent(String anEventType) Captures the specified native JavaScript event and handles it using the "onEvent" method. Called by "captureAllEvents" at the initialisation phase. This method's implementation depends on the used strategy ("abstract").
void	_captureAllEvents() Registers this object as handler for all mouse events at the window object(s).
void	_onEvent(String anEventType) Handles all native JavaScript mouse events. Creates a CSMouseEvent object and calls "fireEvent" on the object that is event source for this event. If CSDragDrop exists, it calls the onMouseEvent-method of its instance with a CSDragDropEvent as a parameter. This method's implementation depends on the used strategy ("abstract").
void	startDragDrop() Tells the dispatcher to create a new CSDragDrop object and to start informing it about occuring drag and drop operations.
void	stopDragDrop() Tells the dispatcher to stop informing the CSDragDrop object about occuring drag and drop operations.

Tabelle A.1: Class CSAbstractEventDispatcher.

A.1.2 Class CSAbstractLayerFunctions extends CSBase

This abstract class defines the methods that allow the manipulation and creation of browser dependent HTML base elements as Layer objects in Netscape or DIV's in Internet Explorer 4. The CSBrowserAbstractionFactory class creates an instance of a subclass depending from the client type.

Return value	Method name, parameters and description
void	CSAbstractLayerFunctions() Creates a new CSAbstractLayerFunctions object (this class is abstract and should not be instantiated).
void	setPosition(CSPoint aValue) Sets the position of the DHTML object (relative to the parent).

Tabelle A.2: Class CSAbstractLayerFunctions.

Return value	Method name, parameters and description
CSPoint	getPosition() Returns the position of the DHTML object (relative to the parent).
void	setSize(CSDimension aValue) Sets the size of the DHTML object.
CSDimension	getSize() Returns the size of the DHTML object.
void	setBackgroundColor(String aValue) Sets the background color of the DHTML object.
String	getBackgroundColor() Returns the background color of the DHTML object.
void	setVisibility(boolean isVisible) Sets the visibility of the DHTML object.
boolean	getVisibility() Returns the visibility of the DHTML object.
Object	createLayer() Creates a DHTML object (depending from the client type) and returns a reference to it.
void	deleteLayer(Object aLayer) Deletes the DHTML object.

Tabelle A.2: Class CSAbstractLayerFunctions.

A.1.3 Class CSAddRemoveEvent extends CSEvent

These events are generated by composites when a component is added or removed to notify objects that are interested in such operations.

Return value	Method name, parameters and description
void	CSAddRemoveEvent(String aType, CSBase aSource, boolean mayPropagate) This constructor creates a new event object.
void	setComponent(CSComponent aValue) Sets the component that was added or removed.
CSComponent	getComponent() Returns the component that was added or removed.

Tabelle A.3: Class CSAddRemoveEvent.

A.1.4 Class CSBase

This class is superclass for all further classes. It provides additional methods (which have to be implemented by the HAF project group) like "instanceof".

Return value	Method name, parameters and description
void	CSBase() This constructor creates a new CSBase object.

Tabelle A.4: Class CSBase.

A.1.5 Class CSBorderLayoutManager extends CSLayoutManager

This class provides BorderLayout. It interprets the layout constraints in a way that it only accepts values for the 5 positions (north, south, east, west, center) and ignores all components that posses other constraints values. If any constraint is used twice, all components after the first one are ignored. The first occurance of a null constraint is interpreted as should be centered. It is proposed to use each constraint region only once.

Return value	Method name, parameters and description
void	CSBorderLayoutManager(int anHgap, int aVgap) This Constructor creates a new layout manager with BorderLayout. If there are no gaps defined, they are both set to zero.
CSDimension	getPreferredSize(CSComposite aComposite) Returns the preferred size of the container. The width is calculated through the maximum of (north, south, east+west+center). The height is calculated by summing up north, south and maxumim of (east, west, center). In addition, the values of hgap and vgap are added the required number of times.
void	layout(CSComposite aComposite) Positions the child elements according to BorderLayout.
void	setHgap(int aValue) Sets the horizontal gap between children.
int	getHgap() Returns the horizontal gap between children.
void	setVgap(int aValue) Sets the vertical gap between children.
int	getVgap() Returns the vertical gap between children.

Tabelle A.5: Class CSBorderLayoutManager.

A.1.6 Class CSBoxLayoutManager extends CSLayoutManager

This layout manager offers the possibility to lay out components in row or column mode. If used in row mode it behaves similar to FlowLayout but without wrapping. If used in column mode it positions the first element on the top and all others beneath the first one. without any wrapping. Like FlowLayout it interprets the layout constraints as indices and it behaves the same in cases of ambiguity.

Return value	Method name, parameters and description
void	CSBoxLayoutManager(int anAxis, int anAlignment, int anHgap, int aVgap) This Constructor creates a new layout manager with BoxLayout. The second argument (after name) defines if column or row mode should be chosen (default is column mode). If there is no alignment as second argument set, it uses centered alignment as default. If there are no gaps defined, they are both set to zero.
int	getVgap() Returns the vertical gap between children.
void	setVgap(int aValue) Sets the vertical gap between children.
int	getHgap() Returns the horizontal gap between children.
void	setHgap(int aValue) Sets the horizontal gap between children.
int	getAlignment() Returns the alignment of the children.
void	setAlignment(int aConst) Sets the horizontal alignment of the children if column mode is chosen and the vertical alignment if row mode is chosen.
void	layout(CSComposite aComposite) Positions and resizes the child elements to achieve FlowLayout.
CSDimension	getPreferredSize(CSComposite aComposite) Returns the preferred size of the container. In row mode the width is calculated by summing up all elements widths plus the horizontal gaps between them and the height by calculating the largest height of all elements plus two vertical gaps. In column mode the height is calculated by summing up all elements heights plus the vertical gaps between them and the width by calculating the largest width of all elements plus two horizontal gaps.

Tabelle A.6: Class CSBoxLayoutManager.

A.1.7 Class CSBrowserAbstractionFactory extends CSBase

This class acts as a factory for generating browser dependend objects ("strategies", as you would call it in the design pattern world).

Return value	Method name, parameters and description
void	CSBrowserAbstractionFactory() Creates a new CSBrowserAbstractionFactory object.
CSAbstract-EventDispatcher	getEventDispatcher() Returns an object for mouse event dispatching, depending from the browser type.
CSDnD-EventDispatcher	getDnDEventDispatcher() Returns an object for mouse event dispatching in case of drag and drop across frames, depending from the browser type.
CSAbstract-LayerFunctions	getLayerFuntions() Returns an object for creation and manipulation of dynamic elements ("layers"), depending from the browser type.

Tabelle A.7: Class CSBrowserAbstractionFactory.

A.1.8 Class CSComponent extends CSEventSource

CSComponent describes the properties of the component class in the context of the composite design pattern. It is not an abstract class but provides all the necessary operations for leaf objects.

Return value	Method name, parameters and description
void	CSComponent(CSPoint aPosition, CSDimension aSize, boolean isVisible) This constructor creates a new component object. As an argument it is possible to pass the name, parent, position, size and visibility of the component.
void	setX(int aValue) Sets the X-coordinate for the object (relative to parent).
int	getX() Returns the X-coordinate of the component (relative to parent).
void	setY(int aValue) Sets the Y-coordinate for the object (relative to parent).
int	getY() Returns the Y-coordinate of the component (relative to parent).

Tabelle A.8: Class CSComponent.

Return value	Method name, parameters and description
void	setWidth(int aValue) Sets the object's width.
int	getWidth() Returns the object's width.
void	setHeight(int aValue) Sets the object's height.
int	getHeight() Returns the object's height.
void	setSize(CSDimension aValue) Sets the object's size.
CSDimension	getSize() Returns the object's size.
void	setPosition(CSPoint aValue) Sets the object's position.
CSPoint	getPosition() Returns the object's position.
void	setPreferredSize(CSDimension aValue) Sets the object's preferred size.
CSDimension	getPreferredSize() Returns the object's preferred size.
void	setVisible(boolean aValue) Sets if the object should be displayed (that means if it should be layouted and painted).
boolean	isVisible() Checks if the object should be displayed (that means if it should be layouted and painted).
void	setBackgroundColor(String aValue) Sets the object's background color.
String	getBackgroundColor() Returns the object's background color.
void	setLayoutConstraints(Object aValue) Sets the object's layout constraints - they fully depend on the used layout manager. Changing the layout manager will make it necessary to set them again in most cases.
Object	getLayoutConstraints() Returns the object's layout constraints.

Tabelle A.8: Class CSComponent.

Return value	Method name, parameters and description
CSBase	getParent() Returns the object's parent. In most cases this will be a CS-Composite object or any derived class. For CSContentPane this will null because they can't be added to another container.
void	setDragSource(boolean aValue) Defines if the component is a drag source or not (default: no).
boolean	isDragSource() Returns if the component is a drag source or not.
void	setDropTarget(boolean aValue) Defines if the component is a drop target or not (default: no).
boolean	isDropTarget() Returns if the component is a drop target or not.
void	show() Causes the object to display its physical layer in the right status. First it searches the uppermost parent and from there it calls layout() and paint() if the corresponding components are invalid.
void	paint() Handles the synchronization between the object's properties and its physical representation.
void	setValid(boolean aValue) Defines if the internal and external size-/position-data is in sync or not. If it is set true, this happens only for this object. If it is set false, this happens recursively for its parent as well.
boolean	isValid() Returns if the internal and external size-/position-data is in sync or not.
void	setSizeKnowledge(boolean aValue) Sets the size-knowledge flag. If it is set true, this happens only for this object. If it is set false, this is done recursive for the parent as well.
boolean	hasSizeKnowledge() Returns the size-knowledge flag.
void	_updatePhysicalLayer() Sets all the pyhsical layer data so that it is in sync with the internal data.
Object	setPhysicalLayer(Object aValue) Sets the object's physical layer.

Tabelle A.8: Class CSComponent.

Return value	Method name, parameters and description
Object	getPhysicalLayer() Returns the object's physical layer.

Tabelle A.8: Class CSComponent.

A.1.9 Class CSComposite extends CSComponent

CSComposite provides a container class for CSComponents. It offers layout management.

Return value	Method name, parameters and description
void	CSComposite(CSPoint aPosition, CSDimension aSize, boolean isVisible, CSLayoutManager aLayoutManager) This constructor creates a new composite object.
void	add(CSComponent aComponent, int aPosition) Inserts a component in the container with the specified layout constraints. If no constraints are given, the component might not be displayed (depending on the layout mechanism). While the component is added to the container, a physical layer object (DIV, node) is created and the reference is given to the component. This method generates Add-Events for listeners of that type.
void	remove(CSComponent aComponent) Deletes a component from the container. While the component is removed from the container, the physical layer object (DIV, node) is deleted as well. This method generates Remove-Events for listeners of that type.
void	setLayoutManager(CSLayoutManager aLayoutManager) Returns the layout manager for the container.
CSLayoutManager	getLayoutManager() Returns the layout manager for the container.
void	setPreferredSize(CSDimension aValue) Sets the preferred size of all its children - should only be called by the layout manager!
CSDimension	getPreferredSize() Returns the preferred size of all its children depending on the layout manager.
CSComponent[]	getChildren() Returns an array of the container's children.

Tabelle A.9: Class CSComposite.

Return value	Method name, parameters and description
CSComponent[]	getChildrenAt(int x, int y) Returns an array of all the child components at the relative position (x/y), in order of appearance in the hirarchy. If the value lies outside of the container, it returns null. If the point hits a gap it returns the container object itself. To perform this task, it calls the method with the same name at the layout manager.
void	layout() Sets the position and size of the container's children according to its layout policy.
void	paint() Synchronizes the physical layer and the internal properties of the container and its children.
void	_createPhysicalLayer() Creates the pyhsical layer that represents a component. Special attention should be paid to reusage of deleted layer objects in Netscape.
void	_deletePhysicalLayer() Deletes the pyhsical layer that represents a component. Special attention should be paid to reusage of deleted layer objects in Netscape.

Tabelle A.9: Class CSComposite.

A.1.10 Class CSContentPane extends CSComposite

This class acts as the top container inside a CSWindow object or an HTML frame(set). It must not be added to another composite.

Return value	Method name, parameters and description
void	CSContentPane(CSPoint aPosition, CSDimension aSize, boolean isVisible, CSLayoutManager aLayoutManager) This constructor creates a new frame object.
String	getHTML() Returns the HTML content that has to be written into the window.document object when the frame is loaded.

Tabelle A.10: Class CSContentPane.

Return value	Method name, parameters and description
void	init(String aFrameName) Called by redirector mechanism to set the reference to the frame window (the physical representation). This method writes its content (from method getHTML()) into the document of this specified window.
void	scrollTo(CSPoint aValue) This method causes the frame to scroll to the specified position. This has no effect if the frame is defined as not scrolling.
void	setX(int aValue) Does nothing (not useful for frame).
int	getX() Returns always 0.
void	setY(int aValue) Does nothing (not useful for frame).
int	getY() Returns always 0.
void	setPosition(CSPoint aValue) Does nothing (not useful for frame).
CSPoint	getPosition() Returns always 0, 0.
void	setWidth(int aValue) Does nothing (not useful for frame).
void	setHeight(int aValue) Does nothing (not useful for frame).
void	setSize(CSDimension aValue) Does nothing (not useful for frame).
CSDimension	getPreferredSize() Returns the object's size.
void	setVisible(boolean aValue) Does nothing (not useful for frame) - always stays visible.
Object	getLayoutConstraints() Returns always null.
void	setScrollable(boolean aValue) Defines whether the pane should have a scrollbar or not. In Netscape, this requires the UniversalBrowserWrite privilege.
boolean	isScrollable() Returns whether the pane should have a scrollbar when necessary or not.
void	setWindow(CSWindow aValue) Sets the CSWindow object that contains this content pane.

Tabelle A.10: Class CSContentPane.

Return value	Method name, parameters and description
CSWindow	getWindow() Returns the CSWindow object that contains this content pane.

Tabelle A.10: Class CSContentPane.

A.1.11 Class CSDimension

This class provides a data structur for modeling a two dimensional measurement with width and height.

Return value	Method name, parameters and description
void	CSDimension(int aWidth, int aHeight) This constructor creates a new dimension object.

Tabelle A.11: Class CSDimension.

A.1.12 Class CSDnDEventDispatcher extends CSAbstractEventDispatcher

Extends its superclass by adding support for mouse events during drag and drop between several frames. It is necessary to set information about the frames and their size and position. In Netscape 4+ this class has to handle mouse events for the frameset and in NS 4+ and MS IE 5.0 it has to transform screen mouse coordinates in relation to the component structure.

Return value	Method name, parameters and description
void	CSDnDEventDispatcher()
void	CSAbstractDnDEventDispatcher() Creates a new event dispatcher object for drag and drop across frames (this class is abstract and should not be instantiated).
void	setHTMLFrameset(Object aValue) Sets the physical frameset (in generall "top").
void	setFramesetData(CSFramesetData aValue) Sets the CSFramesetData object that allows the calculation of frame-size and -position.

Tabelle A.12: Class CSDnDEventDispatcher.

A.1.13 Class CSDragDrop extends CSBase

This class provides the functionality for drag and drop operations. In case of a drag operation ("MouseDown" over a drag source) it must recieve all mouse events ("MouseMove" and "MouseUp") from each frame in a window. This class should be treated as a Singleton (i.e. there should exist only one instance of this class). It provides default icons for all drag situations.

Return value	Method name, parameters and description
void	CSDragDrop(CSContentPane aValue) Creates a new drag and drop object and registers it as a mouse event and add/remove event listener at the specified CSContentPane and all its children.
void	onMouseMove(CSDragDropEvent anEvent) Handler method for mouse move events from an event source.
void	onMouseDown(CSDragDropEvent anEvent) Handler method for mouse down events from an event source.
void	onMouseUp(CSDragDropEvent anEvent) Handler method for mouse up events from an event source.
void	_addCurrentDragSource(CSComponent aValue) Adds a component to the list of current drag sources (without multiple selection, this list contains just one or no element).
CSComponent[]	_getCurrentDragSources() Returns an array of the current drag sources.
void	setDraggedIcon(String aURL) Defines the URL for the icon that should be displayed next to the mouse pointer during a drag operation above a legal drop target, if the drag source does not define an icon of its own.
String	getDraggedIcon() Returns the URL for the icon that should be displayed next to the mouse pointer during a drag operation above a legal drop target, if the drag source does not define an icon of its own.
void	setNoDropAllowedIcon(String aURL) Defines the URL for the icon that should be displayed next to the mouse pointer during a drag, if a drop operation is not allowed above this drop target.
String	getNoDropAllowedIcon() Returns the URL for the icon that should be displayed next to the mouse pointer during a drag, if a drop operation is not allowed above this drop target.

Tabelle A.13: Class CSDragDrop.

Return value	Method name, parameters and description
void	setCopyIconExtension(String aURL) Defines the URL for the icon that should be displayed next to the mouse pointer during a drag (in addition to the "draggedIcon" or the "noDropAllowedIcon").
String	getCopyIconExtension(String aURL) Returns the URL for the icon that should be displayed next to the mouse pointer during a drag (in addition to the "draggedIcon" or the "noDropAllowedIcon").
void	setNoInformationIcon(String aURL) Defines the URL for the icon that should be displayed next to the mouse pointer during a drag, if the client does not know whether a drop operation is allowed above this drop target or not.
String	getNoInformationIcon(String aURL) Returns the URL for the icon that should be displayed next to the mouse pointer during a drag, if the client does not know whether a drop operation is allowed above this drop target or not.

Tabelle A.13: Class CSDragDrop.

A.1.14 Class CSDragDropEvent extends CSMouseEvent

This class adds drag and drop event specific features to the CSEvent class.

Return value	Method name, parameters and description
void	CSDragDropEvent(String aType, CSBase aSource, boolean mayPropagate) This constructor creates a new drag and drop event object.
void	addDragSource(CSComponent aValue) Adds an object that is marked as one of the sources of the drag and drop operation (preparation for multiple selection).
CSComponent[]	getDragSources() Returns the object that is source of the drag and drop operation.
void	setDropTarget(CSComponent aValue) Sets the object that would be the target of a drop if it occured at this position. This is the highest object in the hierarchy at the current mouse pointer position that fullfills "isDropTarget==true".

Tabelle A.14: Class CSDragDropEvent.

Return value	Method name, parameters and description
CSComponent	getDropTarget() Returns the object that would be the target of a drop if it occured at this position.

Tabelle A.14: Class CSDragDropEvent.

A.1.15 Class CSEvent extends CSBase

This class provides the basic features of an event object. For special purpose usage like mouse events, key events or others it is recomended to extend this class.

Return value	Method name, parameters and description
void	CSEvent(String aType, CSBase aSource, boolean mayPropagate) This constructor creates a new event object.
void	setSource(CSEventSource anObject) Sets the origin of the event.
CSEventSource	getSource() Returns the origin of the event.
void	setType(String aValue) Sets the type of the event.
String	getType() Returns the type of the event.
void	consume() Disallowes the propagation of this event to other listeners.
boolean	mayPropagate() Returns if the propagation of this event to other listeners is allowed or not.

Tabelle A.15: Class CSEvent.

A.1.16 Class CSEventSource extends CSBase

This class provides the capabilities for organising event listeners and propagating events to them.

Return value	Method name, parameters and description
void	CSEventSource() This constructor creates a new event source object.

Tabelle A.16: Class CSEventSource.

Return value	Method name, parameters and description
void	addListener(CSBase aListener, String anEventType) Adds a listener object for a specific event type to the event source. If the source isn't already registered as a listener of this type at the event handler of the CSWindow object, it calls "addListener" on this event handler.
void	removeListener(CSBase aListener, String anEventType) Removes a listener object for a specific event type from the event source (if it has been a listener, otherwise it does nothing). If this was the last listener of this type, it calls "removeListener" on the event handler of the CSWindow object.
void	fireEvent(CSEvent anEvent) Fires an event to all registered listeners for this type of event. It does this by calling the method "onEventType" on all listeners, where 'EventType' is substituted by the event's type.
boolean	checkListener(CSBase anObject, String anEventType) Returns whether the specified object is a listener for the specified event type at this event souce or not.

Tabelle A.16: Class CSEventSource.

A.1.17 Class CSFlowLayoutManager extends CSLayoutManager

Layout manager that implements a FlowLayout. All the children are positioned one after (right from) the other and a new line is started if the available space makes it necessary. This layout mechanism interprets the layout constraints as integer values that specify the position in the list of children. If a component has no such constraints, it is placed at the lowest position that isn't used by any other component yet. If a component's contraints point to an index that is already used, it is placed at the lowest available position as well. If you use FlowLayout, try to avoid this indices at all or keep them all in the right order. Anything in between may lead to unwanted results.

Return value	Method name, parameters and description
void	CSFlowLayoutManager(int anAlignment, int anHgap, int aVgap) This Constructor creates a new layout manager with FlowLayout. If there is no alignment set, it uses centered alignment as default. If there are no gaps defined, they are both set to zero.

Tabelle A.17: Class CSFlowLayoutManager.

Return value	Method name, parameters and description
CSDimension	getPreferredSize(CSComposite aComposite) Returns the preferred size of the container. The width is calculated by summing up all elements widths plus the horizontal gaps between them, the height by calculated the largest height of all elements plus two vertical gaps.
void	layout(CSComposite aComposite) Positions and resizes the child elements to achieve FlowLayout.
void	setAlignment(int aConst) Sets the horizontal alignment of the children.
int	getAlignment() Returns the horizontal alignment of the children.
void	setHgap(int aValue) Sets the horizontal gap between children.
int	getHgap() Returns the horizontal gap between children.
void	setVgap(int aValue) Sets the vertical gap between children.
int	getVgap() Returns the vertical gap between children.

Tabelle A.17: Class CSFlowLayoutManager.

A.1.18 Class CSFramesetData extends CSBase

This class holds data for calculating the positions and sizes of frames in a frameset.

Return value	Method name, parameters and description
void	CSFramesetData(Object anHTMLFrameset, String aRowsCols, int aBorder, CSFramesetData[] aFrameList) Creates a new frameset data object.
void	setFrameList(CSFramesetData[] aValue) Defines the list of frames and framesets for this frameset. (Example: setFrameList([null, frameset2, frameset3]) defines that the first element is a frame, the second and third are framesets (CSFramesetData objects).)
CSFramesetData[]	getFrameList() Returns the list of frames and framesets for this frameset.
void	setRowsCols(String aValue) Defines the layout in the frameset (same syntax as HTML).

Tabelle A.18: Class CSFramesetData.

Return value	Method name, parameters and description
String	getRowsCols() Return the layout definition for the frameset (same syntax as HTML).
void	setBorder(int aValue) Defines the width in pixels of the borders between the frames (0 means no border).
int	getBorder() Returns the width in pixels of the borders between the frames (0 means no border).

Tabelle A.18: Class CSFramesetData.

A.1.19 Class CSHTMLComponent extends CSComponent

This class represents a component that holds HTML data.

Return value	Method name, parameters and description
void	CSHTMLComponent(CSPoint aPosition, CSDimension aSize, boolean isVisible, Object aDIV) This constructor creates a new HTML component object. It is possible to attach it to a "static" predefined HTML DIV. Here the problem is that in Netscape You can't fetch the HTML content of such a layer.
void	setStyle(Object aValue) Sets the CSS object defining the style for the HTML component.
Object	getStyle() Returns the CSS object defining the style for the HTML component.
void	setContent(String aValue) Sets the HTML content of the component.
String	getContent() Returns the HTML content of the component.
void	setURL(String aValue) Sets the URL from where the component loads its content. This tries to load the content immediately.
String	getURL() Returns the URL from where the component loaded its content.

Tabelle A.19: Class CSHTMLComponent.

A.1.20 Class CSLayoutManager extends CSBase

Superclass for all layout managers. Provides methods to calculate the preferred size of the container and laying out the child components. This class is the default layout manager for CSComposites if no other is specified. It implements a NullLayout (i.e. all the children are placed depending on their absolute coordinates). This layout mechanism ignores the layout constraints of the children.

Return value	Method name, parameters and description
void	CSLayoutManager() This Constructor creates a new layout manager object with NullLayout.
CSDimension	getPreferredSize(CSComposite aComposite) Returns the preferred size of the container. The width is calculated through the distance between the right border of the rightmost and left border of the leftmost element (similar for the height).
void	layout(CSComposite aComposite) Positions the child elements according to their absolute positions and sizes.
CSComponent[]	getChildrenAt(CSComposite aComposite) Returns an array of all the child components at the relative position (x/y), in order of appearance in the hirarchy. If the value lies outside of the container, it returns null. If the point hits a gap it returns the container object itself.

Tabelle A.20: Class CSLayoutManager.

A.1.21 Class CSMouseEvent extends CSEvent

This class adds mouse event specific features to the CSEvent class.

Return value	Method name, parameters and description
void	CSMouseEvent(String aType, CSBase aSource, boolean mayPropagate) This constructor creates a new mouse event object.
void	setScreenX(int aValue) Sets the vertical screen position of the mouse pointer.
int	getScreenX() Returns the vertical screen position of the mouse pointer.
void	setScreenY(int aValue) Sets the horizontal screen position of the mouse pointer.

Tabelle A.21: Class CSMouseEvent.

Return value	Method name, parameters and description
int	getScreenY() Returns the horizontal screen position of the mouse pointer.
void	setComponentX(int aValue) Sets the vertical position of the mouse pointer relative to the component where it occured.
int	getComponentX() Returns the vertical position of the mouse pointer relative to the component where it occured.
void	setComponentY(int aValue) Sets the horizontal position of the mouse pointer relative to the component where it occured.
int	getComponentY() Returns the horizontal position of the mouse pointer relative to the component where it occured.
void	setShiftKey(boolean aValue) Sets whether the Shift key was pressed when the event was created or not.
void	setCtrlKey(boolean aValue) Sets whether the Ctrl key was pressed when the event was created or not.
void	setAltKey(boolean aValue) Sets whether the Alt key was pressed when the event was created or not.
boolean	isShiftKey() Returns true if the Shift key was pressed when the event was created.
boolean	isCtrlKey() Returns true if the Ctrl key was pressed when the event was created.
boolean	isAltKey() Returns true if the Alt key was pressed when the event was created.
void	setLeftButton(boolean aValue) Sets whether the left mouse button was pressed when the event was created or not.
void	setRightButton(boolean aValue) Sets whether the right mouse button was pressed when the event was created or not.

Tabelle A.21: Class CSMouseEvent.

Return value	Method name, parameters and description
void	setMiddleButton(boolean aValue) Sets whether the middle mouse button was pressed when the event was created or not.
boolean	isLeftButton() Returns whether the left mouse button was pressed when the event was created or not.
boolean	isRightButton() Returns whether the right mouse button was pressed when the event was created or not.
boolean	isMiddleButton() Returns whether the middle mouse button was pressed when the event was created or not.

Tabelle A.21: Class CSMouseEvent.

A.1.22 Class CSPoint

This class provides a data structur for modeling a graphical point that posses an x- and a y-axis.

Return value	Method name, parameters and description
void	CSPoint(int aXCoord, int aYCoord) This constructor creates a new point object.

Tabelle A.22: Class CSPoint.

A.1.23 Class CSWindow extends CSEventSource

This class offers methods to define window behaviour. It is based on the WinContainer class by mmair (and works as an adapter for it) but it doesn't return a handle for changing the appearance of the window - this is done through methods of the class itself.

Return value	Method name, parameters and description
void	CSWindow(String aName, String aSrc, CSBase aParent, CSPoint aPosition, CSDimension aSize) This constractor creates a new window object. The first 3 parameters are obligatory.
void	setX(int aValue) Sets the X-coordinate of the window. (Must not be outside of screen because this causes security exception in Netscape!)

Tabelle A.23: Class CSWindow.

Return value	Method name, parameters and description
int	getX() Returns the X-coordinate of the window.
void	setY(int aValue) Sets the Y-coordinate for the window. (Must not be outside of screen because this causes security exception in Netscape!)
int	getY() Returns the Y-coordinate of the window.
void	setWidth(int aValue) Sets the window's width.
int	getWidth() Returns the window's width.
void	setHeight(int aValue) Sets the window's height.
int	getHeight() Returns the window's height.
void	setSize(CSDimension aValue) Sets the window's size.
CSDimension	getSize() Returns the window's size.
void	setPosition(CSPoint aValue) Sets the window's position. (Must not be outside of screen because this causes security exception in Netscape!)
CSPoint	getPosition() Returns the window's position.
void	show() Makes the window visible on the screen. If it has not been created physically, this is done here with all the properties that have been defined so far.
void	dispose() Closes the window and releases all ressources.
boolean	isDisposed() Returns whether the window is closed or not. If it is closed one should not attempt its properties any more.
void	focus() Gives keyboard focus to the window. This brings the window to the front on most platforms.
void	blur() Takes keyboard focus from the window. This sends the window to the background on most platforms.

Tabelle A.23: Class CSWindow.

Return value	Method name, parameters and description
void	setStatusline(String aValue) Sets the text of the window's status line (if it posses one).
String	getStatusline() Returns the text of the window's status line (if it posses one).
void	setAppearance(String aProperty, boolean isActive) Assigns a certain appearance to the window while opening (value true or false). Appearances are e.g.: "resizable", "status", ... i.e. all options allowed by window.open method except "width" and "height".
boolean	hasAppearance(String aProperty) Checks if a certain appearance is assigned to the window while opening. Appearances are e.g.: "resizable", "status", ... i.e. all options allowed by window.open method except "width" and "height".
void	setResizable(boolean aValue) Defines whether the window should be resizable or not. Default value is "true".
boolean	isResizable() Returns whether the window is resizable or not.
CSBase	getParent() Returns the window's parent.
void	setSrc(String aValue) Sets the window's source.
String	getSrc() Returns the window's source.
void	pack() This method makes only sense if there is just one CSContentPane (without scrollbars) in this window. In this case it asks the CSContentPane to return its preferred size and tries to resize itself according to this value. If it is not possible to obtain all the required space it uses the total screen width and height (otherwise it does nothing).
CSContentPane[]	getContentPanes() Returns an array of content panes (i.e. frames) that are part of this window.
void	addContentPane(CSContentPane aPane) Called by a content pane to register itself at the window.

Tabelle A.23: Class CSWindow.

A.1.24 Interface CSAddRemoveListenerIfc

This interface provides the signature for observer methods concerning adding and removing components to or from a composite.

Return value	Method name, parameters and description
void	onAdd(CSAddRemoveEvent anEvent) Is called before a component is added to a container.
void	onRemove(CSAddRemoveEvent anEvent) Is called before a component is removed from a container.

Tabelle A.24: Interface CSAddRemoveListenerIfc.

A.1.25 Interface CSDragListenerIfc

This interface has to be implemented by objects that are drag sources.

Return value	Method name, parameters and description
void	onDragStart(CSDragDropEvent anEvent) Is called whenever the start of a drag operation is detected.
void	onDragEnter(CSDragDropEvent anEvent) Is called whenever the drag source enters the area of a drop target.
void	onDragOver(CSDragDropEvent anEvent) Is called whenever the drag source is moved over the area of a drop target.
void	onDragExit(CSDragDropEvent anEvent) Is called whenever the drag source leaves the area of a drop target.
void	onDragDrop(CSDragDropEvent anEvent) Is called whenever the drag source is dropped on a drop target.
String	getDraggedIcon() Returns the URL of the icon that should be displayed next to the mouse pointer when a drag occurs. If this method returns null (or is not implemented - a case that should not occur), the default icon is used.

Tabelle A.25: Interface CSDragListenerIfc.

A.1.26 Interface CSDropListenerIfc

This interface has to be implemented by objects that are drop targets.

Return value	Method name, parameters and description
void	onDropEnter(CSDragDropEvent anEvent) Is called whenever a drag source enters the area of the drop target.
void	onDropOver(CSDragDropEvent anEvent) Is called whenever a drag source is moved over the area of the drop target.
void	onDropStalled(CSDragDropEvent anEvent) Is called whenever a drag source stays over the area of the drop target for a several periods without moving (for invoking operations like opening folders in a tree etc.).
void	onDropExit(CSDragDropEvent anEvent) Is called whenever a drag source leaves the area of the drop target.
void	onDropDrop(CSDragDropEvent anEvent) Is called whenever a drag source is dropped on the drop target.

Tabelle A.26: Interface CSDropListenerIfc.

A.1.27 Interface CSMouseListenerIfc

This Interface has to be implemented by Objects that want to be notified about mouse events.

Return value	Method name, parameters and description
void	onMouseMove(CSMouseEvent anEvent) Invoked by the event source if the mouse pointer is moved over the component.
void	onMouseDown(CSMouseEvent anEvent) Invoked by the event source if a mouse button has been pressed over the component.
void	onMouseRemainingDown(CSMouseEvent anEvent) Invoked by the event source if the mouse button is pressed and the pointer has not been moved for a period of time (all happening over the component).
void	onMouseUp(CSMouseEvent anEvent) Invoked by the event source if a mouse buton is released over the component.
void	onMouseEnter(CSMouseEvent anEvent) Invoked by the event source if the mouse pointer enters the component.

Tabelle A.27: Interface CSMouseListenerIfc.

Return value	Method name, parameters and description
void	onMouseOut(CSMouseEvent anEvent) Invoked by the event source if the mouse pointer leaves the component.

Tabelle A.27: Interface CSMouseListenerIfc.

A.2 Verwendete Design Pattern

Zum leichteren Verständnis einzelner Design-Entscheidungen werden im folgenden alle implementierten Design Pattern angeführt, deren Bekanntheit jedoch vorausgesetzt wird[1].

A.2.1 Composite

Dieses Design Pattern bietet sich optimal für die Implementation der Container-Struktur des JavaScript AWT's an. Auch hier sollen einem Container zusätzliche Komponenten (darunter auch weitere Container) übergeben werden können, die vom Layout Manager des Containers in Folge richtig ausgerichtet und platziert werden.

Die Rollen in diesem Pattern sind folgendermaßen besetzt:

Component: CSComponent.

Leaf: CSHTMLComponent.

Composite: CSComposite.

Client: Unterschiedliche Objekt, die Komponenten verwenden bzw. referenzieren wie CSLayoutManager, CSEvent oder andere.

A.2.2 Strategy

Für den Umgang mit den Browser-Inkompatibilitäten bietet sich dieses Muster an – das betrifft den Bereich der Layer-Manipulation und den des internen Event-Handlings. Hier wird das Strategy Pattern auch implementiert, in Verbindung mit einer entsprechenden Abstract Factory (siehe nächstes Design Pattern).

Folgende Klassen sind daran beteiligt:

Strategy: CSAbstractLayerFunctions, CSAbstractEventDispatcher, CSDndEventDispatcher.

ConcreteStrategy: Alle weiteren von obigen Klassen abgeleiteten Klassen wie CSLayerFuntionsIE oder CSEventDispatcherNS4.

[1] Als Einführung und Referenz ist das Standardwerk von Erich Gamma et al. "Design Patterns – Elements of Reusable Object-Oriented Software" zu empfehlen.

Context: CSComponent und CSComposite nutzen das Interface der Klasse CSAbstractLayerFunctions für Layer-Manipulationen. (Die Event Dispatcher agieren von sich aus und entsprechen deshalb nicht ganz dem klassischen Strategy Muster.)

A.2.3 Abstract Factory

Im Zusammenhang mit den über das Strategy Pattern erzeugten Browser-Abstraktionen wird das Abstract Factory-Konzept im JavaScript AWT verwendet. So erfolgt die Generierung des Objekts, das sich um das interne Event Handling kümmert, über eine Factory, die abhängig vom Browsertyp ein entsprechendes ConcreteProduct instanziiert.

Die Rollen in diesem Pattern sind wie folgt belegt:

AbstractFactory: Dieses Interface verschmilzt im JavaScript AWT mit der implementierenden Klasse.

ConcreteFactory: CSBrowserAbstractionFactory.

AbstractProduct: CSAbstractEventDispatcher, CSAbstractLayerFunctions, CSDndEventDispatcher.

ConcreteProduct: Alle weiteren von obigen Klassen abgeleiteten Klassen wie CSLayerFuntionsIE oder CSEventDispatcherNS4.

Client: Benutzt wird dieses Pattern z.B. via CSBrowserAbstractionFactory.createEventDispatcher().

A.2.4 Observer

Das Event-Handling des JavaScript AWT's wurde nach dem Observer-Prinzip modelliert. Die Klasse CSEventSource entspricht dabei der Subject Klasse, alle ihre Subklassen sind ConcreteSubjects. Gemäß JavaScript-Konvention gibt es jedoch keine einheitliche Update-Methode, sondern onEvent-Methoden, wobei "Event" für den jeweiligen Event-Typ steht (z.B. onMouseMove).

Die verwendete Observer Struktur umfasst folgende Elemente:

Subject: CSEventSource.

Observer: CSDragListenerIfc, CSDropListenerIfc, CSAddRemoveListenerIfc, CSMouseListenerIfc.

ConcreteSubject: Alle von CSEventSource abgeleiteten Klassen.

ConcreteObserver: Jene Klassen, die eines der Observer Interfaces implementieren.

A.3 Erläuterung der Zustände

A.3.1 Das Flag valid

Dieses Attribut zeigt an, ob die tatsächliche physikalische Repräsentation (der Layer) mit dem internen Zustand eines CSComponent übereinstimmt (true) oder ob das nicht der Fall ist (false).

Im direkten Zusammenhang damit steht die Methode _invalidate(), die dieses Flag beim entsprechenden CSComponent und rekursiv bei seinem Vater auf false setzt.

Dies passiert immer dann, wenn durch eine Größen- oder Positionsänderung der interne und externe Zustand außer Schritt kommen bzw. bei einem Ändern der Sichtbarkeit. Zusätzlich dazu wird ein CSComposite auch dann invalidated, wenn ihm ein weiteres Kind hinzugefügt wird.

Via _setValid() wird das Flag auf true gesetzt. Dies geschieht nach einer _paint-Aktion (d.h. wenn die Synchronisation der beiden Zustände erfolgt ist).

Um unnotwendige Operationen zu vermeiden, soll vor jedem Aufruf von _layout() und _paint() geprüft werden, ob der Component nicht valid ist und somit die Aktionen überhaupt nicht notwendig sind.

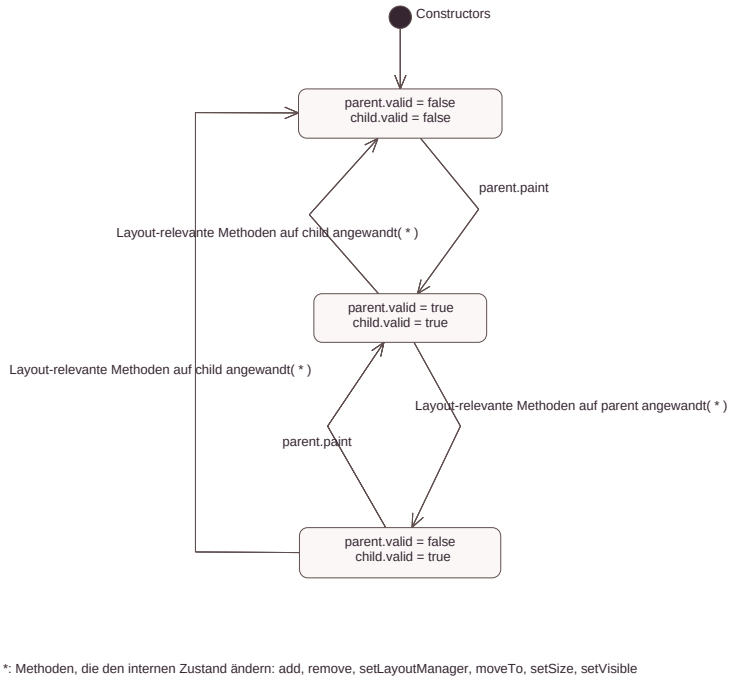

Abbildung A.2: Statechart Diagramm Flag valid.

A.3.2 Das Flag knowSize

Die CSComposite-Klasse besitzt ein weiteres wichtiges Attribut, knowSize. Dieses speichert, ob dem Container seine preferred size bekannt ist oder ob er diese über seinen Layout Manager neu bestimmen lassen soll.

Eine solche Bestimmung ist ein rekursiver Vorgang und kann zu exponentieller Laufzeit führen. Um dies zu vermeiden, merken sich CSComposites ihre gewünschte Größe (die ja von den Kindern und dem Layout Management abhängt) und errechnen diese nur dann neu, wenn das Flag knowSize auf false gesetzt wurde.

Dies passiert beim Ändern der Größe und Sichtbarkeit einer Komponente sowie beim Hinzufügen eines weiteren Kindes ähnlich wie bei _invalidate rekursiv über die Methode _clearSizeKnowledge.

Mittels _setSizeKnowledge() wird das Flag auf true gesetzt – und zwar vom Layout Manager nachdem das Layout durchgeführt wurde bzw. nachdem die gewünschte Größe berechnet wurde.

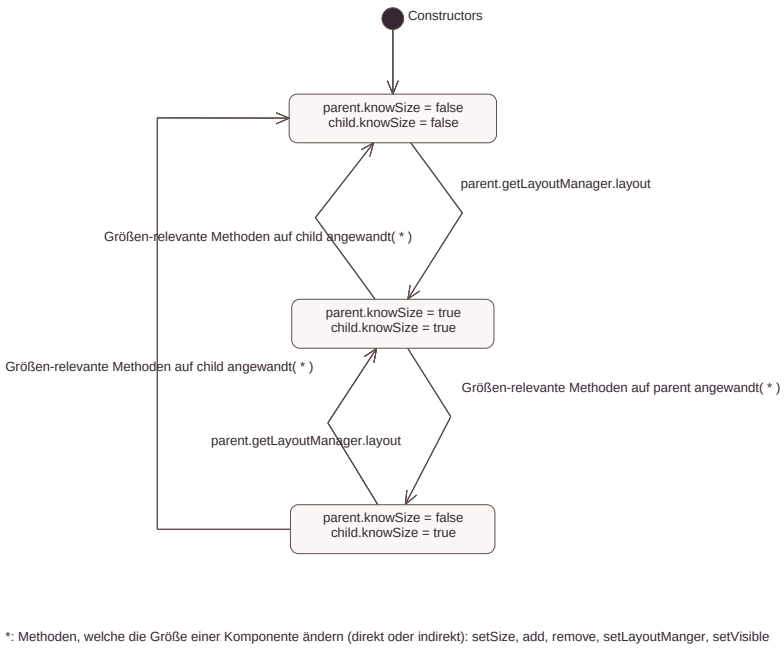

Abbildung A.3: Statechart Diagramm Flag knowSize.

Die beiden Ablaufdiagramme in Abbildung A.6 und Abbildung A.7 sollen helfen, den grundlegenden Layout- und Darstellungsmechansimus zu visualisieren. Trotz des einfachen Beispiels (ein Container mit 2 Kindern, von denen eines weitere zwei Kinder besitzt) kommt es zu einer Vielzahl von Methodenaufrufen[2], deren zeitliche Abfolge durch die vorangestellten Nummern definiert wird.

[2]Aus diesem Grund wurde der Ablauf auf zwei Diagramme aufgeteilt.

A.3.3 Konsistenz der Zustände

Ein direktes Eingreifen in die valid- und knowSize-Struktur ist nicht vorgesehen und nicht erwünscht. Damit könnte es zum einen zu Inkonsistenzen zwischen der Layer-Repräsentation und dem internen Zustand kommen oder aber es wird unnötige Layout- oder Paint-Arbeit geleistet (was zu unerwünschtem Flackern oder Performance-Problemen führen kann).

A.4 Event Sources

Einige Basisklassen des Toolkits sind von der Klasse CSEventSource abgeleitet und generieren Events für Objekte, die sich als Listener bei ihnen registrieren.

A.4.1 CSWindow

Objekte dieser Klasse erzeugen folgende CSEvents:

Event-Name	Zeitpunkt der Generierung
Show	am Ende der show()-Methode
Dispose	am Beginn der dispose()-Methode
Resize	nachdem die Größe geändert wurde
Move	nachdem die Position geändert wurde
Focus	nachdem dem Fenster der Fokus gegeben wurde
Blur	nachdem dem Fenster der Fokus genommen wurde

Tabelle A.28: Events von CSWindow.

A.4.2 CSComponent

Objekte dieser Klasse erzeugen keine CSMouseEvents, sondern reichen sie nur an ihre Listener weiter. Die Generierung der Event-Objekte passiert im Event Dispatcher, der fireEvent() bei der betroffenen Komponente aufruft, sofern diese Listener für den jeweiligen Typ besitzt.

Event-Name	Zeitpunkt der Generierung
MouseMove	wenn sich der Mauszeiger über der Komponente befindet (und sich kein weiteres Kind an dieser Stelle befindet)
MouseDown	wenn eine Maustaste gedrückt wurde
MouseRemainingDown	wenn für eine gewisse Zeit eine Maustaste gedrückt wurde und kein anderer Maus-Event ausgelöst wurde
MouseUp	wenn eine Maustaste losgelassen wurde
MouseEnter	wenn der Mauszeiger auf das Objekt bewegt wird

Tabelle A.29: Events von CSComponent.

Event-Name	Zeitpunkt der Generierung
MouseOut	wenn der Mauszeiger vom Objekt wegbewegt wird (verlangt ein globales (statisches) Speichern des bisherigen Objekts, über dem die Maus war)

Tabelle A.29: Events von CSComponent.

A.4.3 CSComposite

Objekte dieser Klasse erzeugen (zusätzlich zu den geerbten Events) folgende CSAddRemoveEvents:

Event-Name	Zeitpunkt der Generierung
Add	bevor eine Komponente hinzugefügt wird
Remove	bevor eine Komponente entfernt wird

Tabelle A.30: Events von CSComposite.

A.4.4 CSContentPane

Objekte dieser Klasse erzeugen (zusätzlich zu den geerbten Events) folgende CSEvents:

Event-Name	Zeitpunkt der Generierung
Scroll	nachdem sich der "Scroll-Scope" geändert hat

Tabelle A.31: Events von CSContentPane.

A.4.5 CSDragDrop

Objekte dieser Klasse (auch wenn sie kein Event Source sind) erzeugen folgende CSDragDropEvents, die sie an die betroffene Komponente weiterreichen:

Event-Name	Zeitpunkt der Generierung
DragStart	wenn eine Maustaste über einer DragSource gedrückt wird
DragOver	wenn sich der Mauszeiger während einer Drag-Operation über einem DropTarget befindet
DragEnter	wenn der Mauszeiger während einer Drag-Operation ein DropTarget erreicht
DragExit	wenn der Mauszeiger während einer Drag-Operation ein DropTarget verlässt
DragDrop	wenn alle Maustasten während einer Drag-Operation losgelassen wurden

Tabelle A.32: Events von CSDragDrop.

Event-Name	Zeitpunkt der Generierung
DropOver	wenn sich der Mauszeiger während einer Drag-Operation über einem DropTarget befindet
DropStalled	wenn für eine gewisse Zeit während eines Drags die Maustaste gedrückt wird und die Maus nicht bewegt wird (und über einem DropTarget steht)
DropEnter	wenn der Mauszeiger während einer Drag-Operation ein Drop-Target erreicht
DropExit	wenn der Mauszeiger während einer Drag-Operation ein Drop-Target verlässt
DropDrop	wenn sich der Mauszeiger während einer Drag-Operation über einem DropTarget befindet und alle Maustasten losgelassen wurden

Tabelle A.32: Events von CSDragDrop.

A.5 Drag and Drop

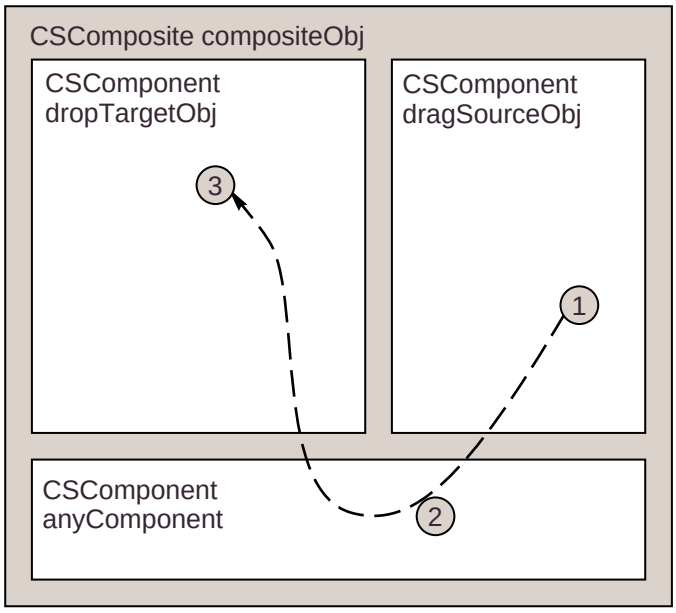

Abbildung A.4: Beispiel für Drag and Drop Events.

Anhand des folgenden Beispiels, das in Abbildung A.4 illustriert ist, soll der prinzipielle Ablauf einer Drag and Drop Operation veranschaulicht werden:

A.5.1 Initialisierung

Die Initialisierung der Komponenten – beschränkt auf die hier wesentlichen Operationen – sieht wie folgt aus:

```
1   // ... Erzeugen der Komponenten
2   compositeObj.add(dropTargetObj, CSBorderLayout.WEST_);
3   compositeObj.add(dragSourceObj, CSBorderLayout.EAST_);
4   compositeObj.add(anyComponent, CSBorderLayout.SOUTH_);
5   dropTargetObj.setDropTarget(true);
6   dragSourceObj.setDragSource(true);
7   var eventDisp = theBrowserAbstractionFactory.getDnDEventDispatcher();
8   compositeObj.show();
```

Interessant wird es bei den Zeilen 5 und 6, in denen die eine Komponente zum Ziel von Drop- und die andere zur Quelle von Drag-Operationen gemacht wird. Allerdings passiert dabei nicht mehr, als dass interne Flags gesetzt werden.

Bei der nächsten Zeile geschieht mehr, denn hier wird via Factory-Konzept ein Event Dispatcher für die Maus-Events im Fall eines Drag and Drops über Frames hinweg erzeugt. Dieser erzeugt implizit ein CSDragDrop-Objekt, das die Maus-Events in Drag and Drop spezifische Events transformiert.

A.5.2 MouseDown (1)

Der Benutzer drücke über dem als DragSource definierten Objekt die linke Maustaste. Der Event wird vom Dispatcher registriert – woraufhin dieser die CSContentPane fragt, welche Komponente sich an dieser Stelle befindet. Aus dem Rückgabewert kann der Dispatcher schliessen, dass dragSourceObj das oberste Objekt mit Interesse an Maus-Events ist.

Ein MouseDown allein bewirkt zuerst nur eine Selektierung des Objekts – erst eine Bewegung der Maus bei gedrückter Taste startet eine Drag-Operation. Tritt dieser Fall ein, so wird vom Dispatcher onMouseMove am CSDragDrop-Objekt ausgeführt.

A.5.3 MouseMove (2)

Der Benutzer führe den Mauszeiger entlang der gestrichelten Linie. Wählen wir als Zeitpunkt für die nächste genauere Betrachtung des Zustands des Systems den Punkt (2), wenn sich der Zeiger gerade über der Komponente anyComponent befindet.

Der Event Dispatcher versucht über die Content Pane das oberste Objekt, das ein Drop Target ist, zu finden. Nachdem aber anyComponent bei isDropTarget() false zurückliefert (und dies auch für alle anderen in der Hierarchie höher liegenden Komponenten gilt), muss das Feld dropTarget null bleiben.

A.5.4 MouseUp (3)

Lässt der Benutzer nun die Maustaste los, so wird nach dem Event Handling von eventDisp am CSDragDrop-Objekt die Methode onMouseUp mit einem CSDragDropEvent aufgerufen, in dem als Source dragSourceObj und als DropTarget dropTargetObj eingetragen sind.

Kern dieser Methode sind die folgenden Zeilen:

```
1  var evt1 = new CSDragDropEvent(...);
2  evt1.setType("DragDrop");
3  for ( var i = 0; i < anEvent.getSources().length; i++ )
4    anEvent.getSources()[i].onDragDrop(evt1);
5  var evt2 = new CSDragDropEvent(...);
6  evt2.setType("DropDrop");
7  anEvent.getDropTarget().onDropDrop(evt2);
```

Zuerst bekommen alle Drag Sources eine Benachrichtigung via onDragDrop, dass die Drag-operation zu Ende ist. Danach wird das DropTarget informiert, dass eine Drop-Operation auf ihm stattfand.

Für die Anzeige eines entsprechenden Icons, das während einer Drag-Operation Feedback bietet, ob gerade ein Drop möglich ist, müssen Drop Target und Drag Source gefragt werden, ob für beide ein Drop sinnvoll wäre. Hier sind allerdings drei Zustände zu beachten: "Erlaube Drop", "Erlaube Drop nicht" und "Weiss es noch nicht" – bei letzterem Fall kann z.B. erst ein Server-Request notwendig sein. Dies muss entsprechend angezeigt werden. Außerdem muss es möglich sein, bei längerem Verharren über einem Ziel einen solchen Request absetzen zu lassen und daraufhin eine Antwort zu erhalten (onDropStalled-Event).

Eine detaillierte Übersicht über die Zustandsübergange bietet Abbildung A.5.

A.5.5 Anmerkung

Die Komplexität von Drag and Drop erweitert sich noch um Aspekte wie Multiple Selection oder Frame-übergreifende Funktionalität. Zu ersterem Punkt ist anzumerken, dass hierbei unterschiedliche Scopes (Container) definiert werden müssen, innerhalb derer mehrfache Markierungen möglich sind. So ist es z.B. nicht sinnvoll, ein Element eines Treeviews und eines einer Listview gleichzeitig zu markieren. Eine solche Operation muss nicht nur verhindert werden, sondern auch zu einem wohldefinierten Zustand führen.

Der Aspekt des Frame-übergreifenden Drag and Drops erfordert vom Steuer-Objekt zusätzliches Wissen über die Position der Objekte in absoluten Bildschirm-Koordinaten – zumindest unter den Netscape-Browsern und Internet Explorer 5.0. Diese Daten müssen zuerst errechnet (wobei dies meist nicht pixelgenau möglich ist) und beim ersten Maus-Event über diesem Frame upgedatet werden. Da dies die Komplexität der Operationen sehr erhöht, wurde für diese Anwendung die Klasse CSDnDEventDispatcher und ihre Subklassen eingeführt.

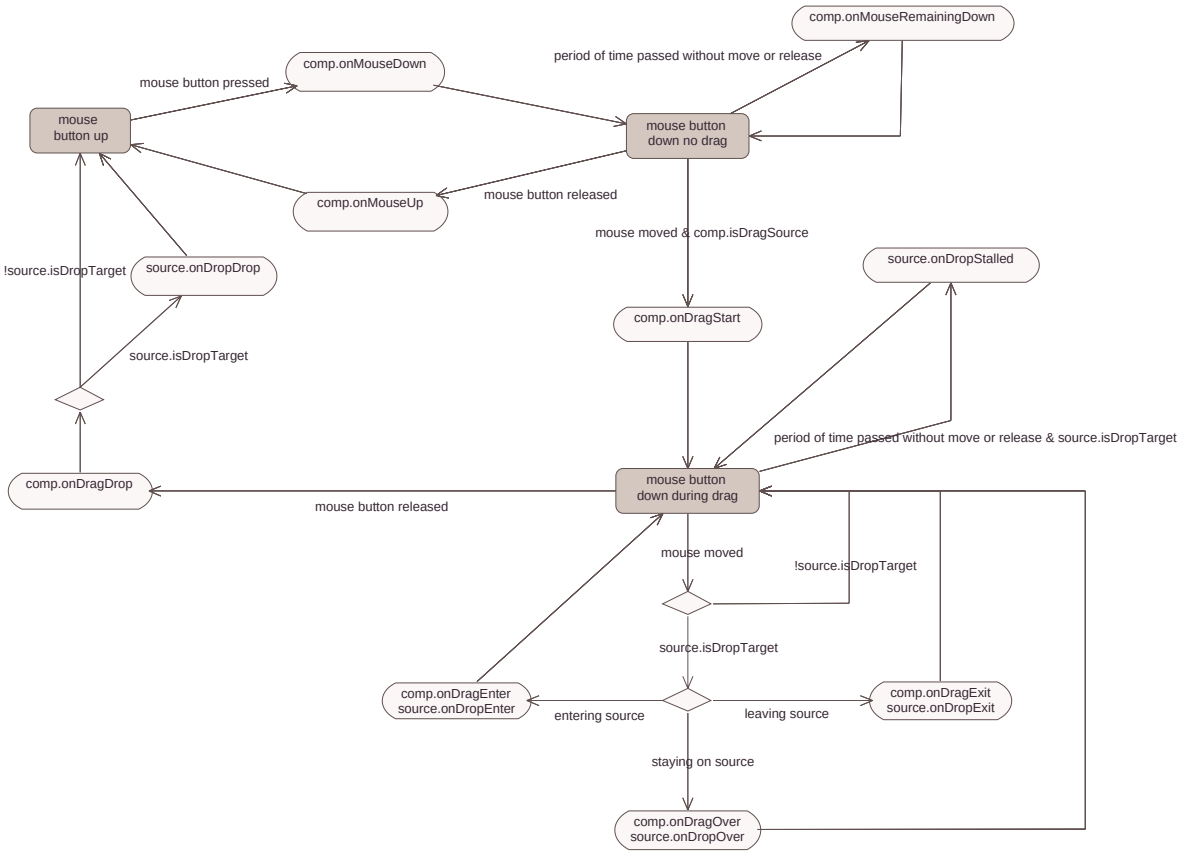

Abbildung A.5: Activity Diagramm Drag and Drop.

A.6 Erzeugung eines Framesets

Wie soll nun konkret die Erzeugung eines Framesets aussehen? Da die Entscheidung getroffen wurde, dass es – sofern kein Drag and Drop über Frames hinweg benötigt wird – für die Frame-Objekte nicht notwendig ist, ihre Position zu kennen, reicht ihnen eine Referenz auf den "physikalischen" Frame des DOM. Wie sie diese erhalten, soll ein kleines Beispiel erläutern:

```
1  <html>
2    <head><title>...</title>
3      <script>
4        // ... Laden der Klassen
5        var navFrame = new CSContentPane(...);
6        var contentFrame = new CSContentPane(...);
7      </script>
8    </head>
```

```
9    <framset cols="100,*">
10     <frame name="nav" src="redirect.htm?navFrame&nav" scrolling="auto">
11     <frameset rows="15%,*">
12       <frame name="static" src="any_url.htm" scrolling="no">
13       <frame name="content" src="redirect.htm?contentFrame&content" scrolling="yes">
14     </frameset>
15   </frameset>
16 </html>
```

Die Redirector-Page "redirect.htm" ist sehr simpel aufgebaut. Hier ihr einfachstes Aussehen[3]:

```
1 <script>
2   var frame_object = location.search.substring(1, location.search.indexOf('&'))
3   var html_frame = location.search.substring(location.search.indexOf('&')+1);
4   eval("top." + frame_object + ".init('" + html_frame + "')");
5 </script>
```

Via Such-Erweiterung der URL der Redirector-Page werden der Name des zugehörigen CSContentPane-Objekts und der HTML-Name des Frames weitergereicht. In den Zeilen 2 und 3 werden diese Daten in "redirect.htm" aus der URL gefiltert und anschließend die Callback-Methode init des CSContentPane aufgerufen.

Mittels dieser Methode bekommt ein CSContentPane-Objekt eine Referenz auf seinen physikalischen Repräsentanten und außerdem die Information, dass jetzt die Seite geschrieben werden kann.

[3] Sollen Styles oder Hintergrundfarben bzw. -Bilder gesetzt werden, muss sich die Komplexität entsprechend erhöhen.

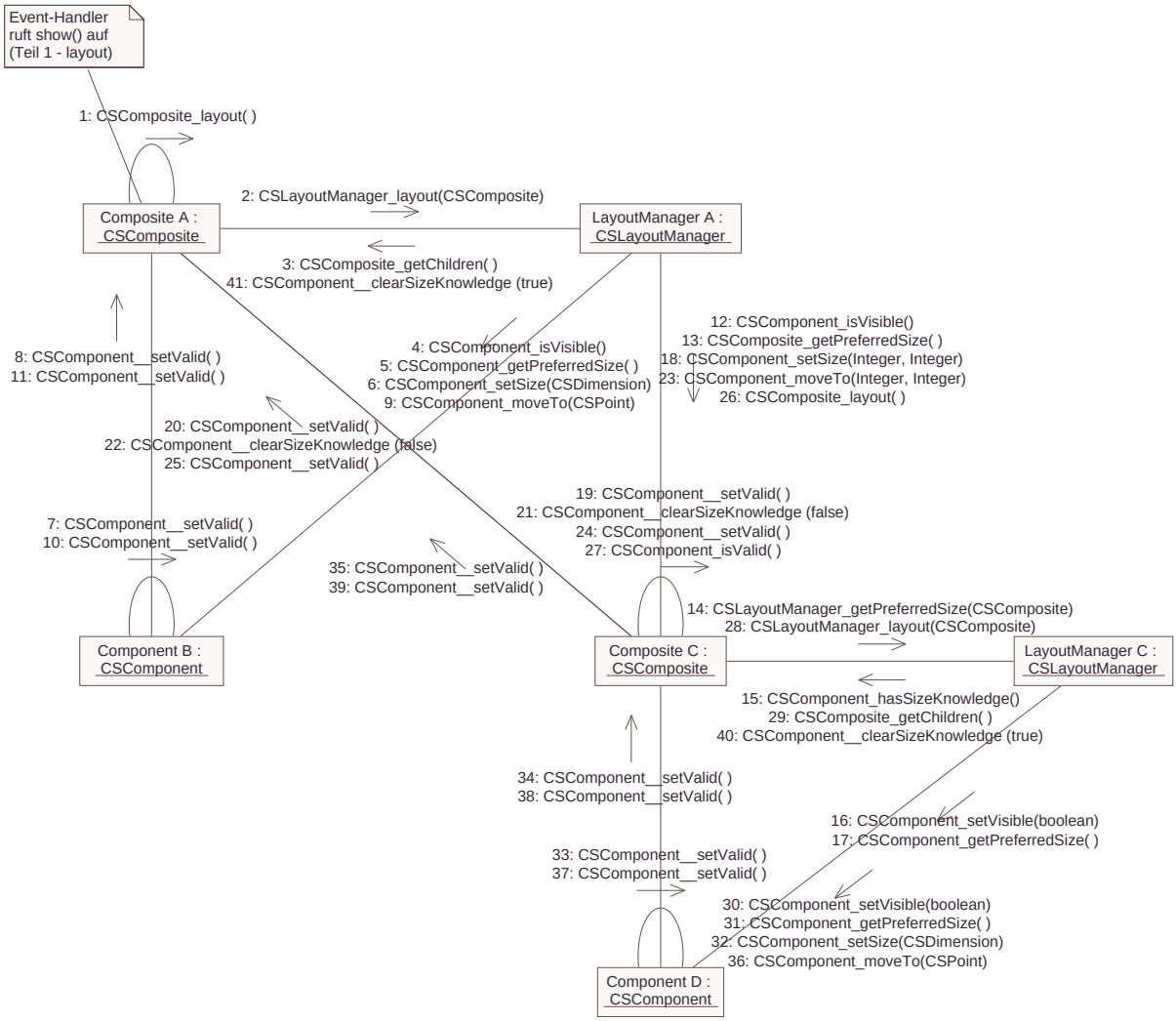

Abbildung A.6: Collaboration Diagramm Methode show, Teil 1.

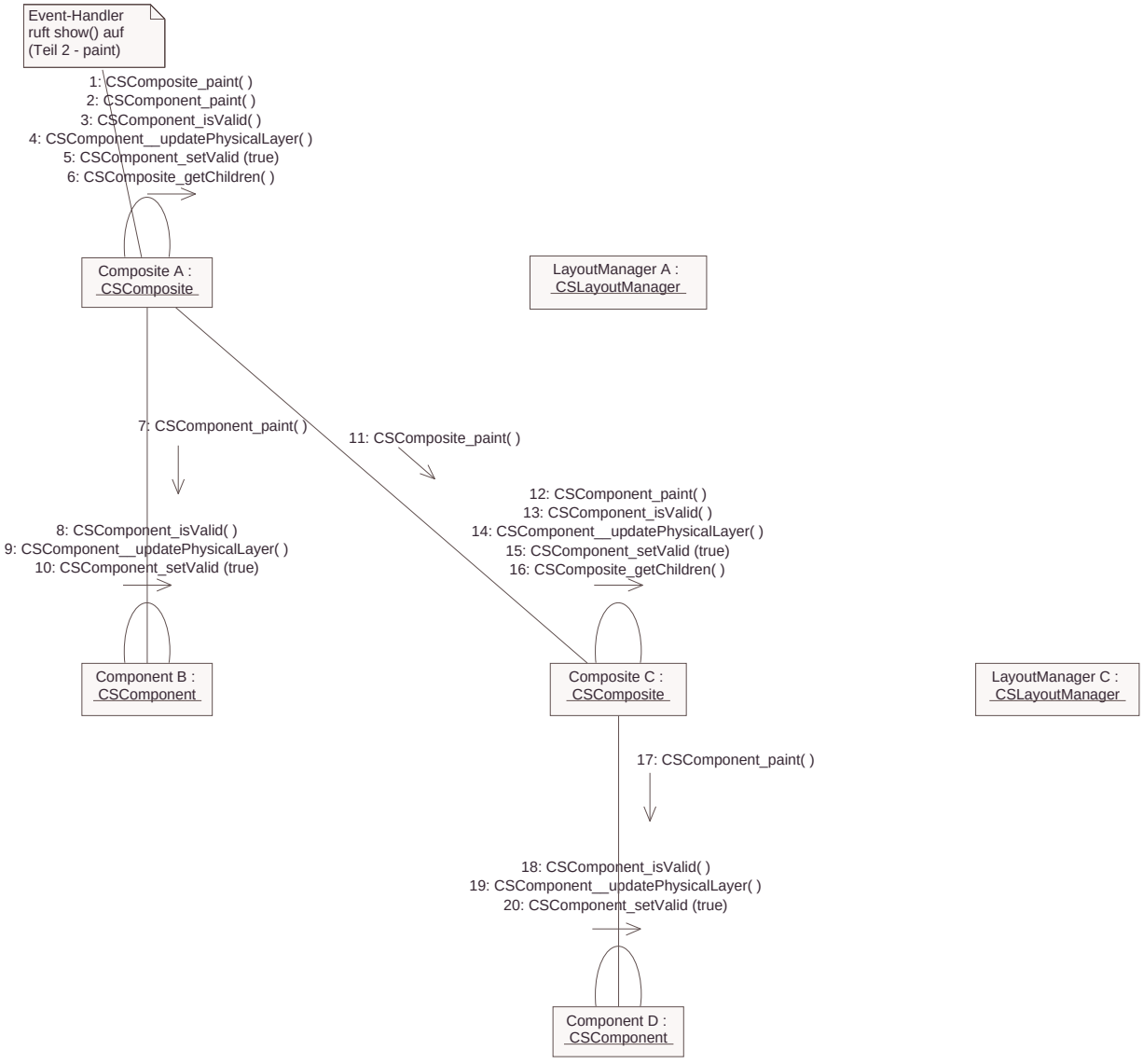

Abbildung A.7: Collaboration Diagramm Methode show, Teil 2.

Literaturverzeichnis

[1] Kent Beck. *Extreme Programming explained: embrace change*. Addison-Wesley, United States of America, first edition, 2000.

[2] Jim Conallen. *Building Web Applications With UML*. Object Technology. Addison-Wesley, United States of America, second edition, 1999.

[3] Jim Conallen. Modeling web applications with uml, March 9, 1999.
http://www.conallen.com/whitepapers/webapps/ModelingWebApplications.htm.

[4] Rational Software Corporation. Rational — the e-development company, February 2001.
http://www.rational.com/.

[5] Rational Software Corporation. Rational unified process, February 2001.
http://www.rational.com/products/rup/index.jsp.

[6] TogetherSoft Corporation. Togethersoft, February 2001.
http://www.togethersoft.com/.

[7] Christopher Alexander et al. *A Pattern Language*. Oxford University Press, New York, 1977.

[8] Erich Gamma et al. *Design Patterns — Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley, United States of America, 20 edition, 1995.

[9] Kent Beck et al. *Planning Extreme Programming*. The XP Series. Addison-Wesley, United States of America, first edition, 2000.

[10] Ron Jeffries et al. *Extreme Programming installed*. The XP Series. Addison-Wesley, United States of America, first edition, 2000.

[11] David Flanagan. *JavaScritpt, The Definitive Guide*. Activate Your Web Pages. O'Reilly, United States of America, third edition, 1998.

[12] Visualization & Usability Center at Georgia Tech Graphics. Gvu's www user surveys — cookie policy, December 15, 1998.
http://www.gvu.gatech.edu/user_surveys/survey-1998-10/graphs/use/q81.htm.

[13] Struts Development Group. Struts user's guide, August 10, 2000.
http://jakarta.apache.org/struts/users_guide.html.

[14] W3C Working Group. Document object model (dom), February 9, 2001.
http://www.w3.org/DOM/.

[15] Michael Houghton. xml for <script>, 2000.
http://www.idle.org/experimental/.

[16] Jason Hunter. The problems with jsp, January 25, 2000.
http://www.servlets.com/soapbox/problems-jsp.html.

[17] Jason Hunter. Reactions to "the problems with jsp", February 10, 2000.
http://www.servlets.com/soapbox/problems-jsp-reaction.html.

[18] Sun Microsystems Inc. Doclet api, February 2001.
http://java.sun.com/j2se/1.3/docs/tooldocs/javadoc/doclet/.

[19] Sun Microsystems Inc. Javadoc 1.3, February 2001.
http://java.sun.com/j2se/1.3/docs/tooldocs/javadoc/.

[20] Jeremie. Xparse, October 19, 2000.
http://www.jeremie.com/Dev/XML/.

[21] Netscape. W3c standards support in ie and the netscape gecko browser engine, 2000.
http://home.netscape.com/browsers/future/standards.html.

[22] Fessel-GfK Peter Diem. Online research in austria 2000, August 25, 2000.
http://www.gfk.at/service/download/files/PRESS/Internet%20in%20Austria%20August%2000.doc.

[23] Shelley Powers. X-objects, June 15, 2000.
http://www.yasd.com/dynatech/xobjsintro.htm.

[24] Aamod Sane. Patterns home page, October 20, 1999.
http://hillside.net/patterns/.

[25] Wolfgang Schwarz. Ausgelst. *iX*, October 2000.

[26] WinWin Software. Web application development, 1999.
http://www.winwinsoft.com/articles/wad.html#java_yes.

[27] Selena Sol. Introduction to the web application development environment (tools), May 31, 1999.
http://wdvl.com/Authoring/Tools/Tutorial/.

[28] Security Space. Web server survey.
http://www.securityspace.com/s_survey/data/index.html.

[29] John Lam und Aaron Skonnard. Architecting your web applications. *Microsoft Internet Developer*, September 1999.

[30] Martin Fowler und Kendall Scott. *UML konzentriert*. Professionelle Softwareentwicklung. Addison-Wesley, United States of America, second edition, 2000.

[31] Alexander Nakhimovsky und Tom Myers. *Professional Java XML Programming with Servlets and JSP*. Programmer to Programmer. Wrox Press Ltd., Great Britain, 1999.

[32] Nathan Wallace. Design patterns in web programming, March 8, 2000. http://www.e-gineer.com/articles/design-patterns-in-web-programming.phtml.

[33] Java World. Web application server, November 9, 2000. http://www.javaworld.com/javaworld/tools/jw-tools-appserver.html.