Universität Hamburg

Fachbereich Informatik
Arbeitsbereich Verteilte Systeme und
Informationssysteme (VSIS)

Studienarbeit

Java in heterogenen Umgebungen: Konzepte der Interaktion zwischen Javaprogrammen und Web-Browsern zur Erfassung von Benutzeraktionen

Torsten Haß

September 2002

Betreuer:
Prof. Dr. Winfried Lamersdorf
Dipl.-Inform. Harald Weinreich

Inhaltsverzeichnis

1 Einleitung

Das Informationsangebot im Internet wächst und verändert sich ständig. In den Jahren von 1995 bis 2000 wuchs das Internet, gemessen an der Anzahl der Host-Rechner, annähernd exponentiell [ISC 2002]. Trotz der starken Veränderung des Internets haben sich die Browser in den letzten Jahren kaum verändert. Dadurch ist das Suchen von Informationen für den Benutzer zu einer Herausforderung geworden. Oftmals muss der Benutzer unzählige Seiten besuchen, bis er die gewünschte Information findet.

Um diesem Missstand abzuhelfen, werden Tools und Browser-Erweiterungen benötigt, die den Benutzer bei seiner Arbeit im Internet unterstützen. Damit derartige Werkzeuge möglichst effizient und hilfreich sind, müssen vor ihrer Entwicklung die Probleme des Benutzers beim Surfen erkannt werden.

Mit Hilfe von Studien zum Benutzerverhalten lassen sich derartige Probleme aufspüren. Dazu müssen möglichst alle Aktionen des Benutzers beim Surfen erfasst und ausgewertet werden. Das ist allerdings nicht einfach, da die Browser, mit denen der Benutzer im Internet arbeitet, diese Informationen nicht ohne weiteres zur Verfügung stellen.

Tauscher und Greenberg [Tau 1997] modifizierten den Quellcode eines Web-Browsers, um sämtliche Benutzeraktionen abfangen und speichern zu können. Dieses Verfahren lies sich seit dem nicht mehr anwenden, da der Quellcode des meistbenutzten Browsers [Int 2001], Microsofts Internet Explorer, nicht zur Verfügung stand und daher nicht modifiziert werden konnte. Die Modifikation eines wenig benutzten Browsers hätte den Nachteil, dass die daraus entstandenen Tools möglicherweise nur mit diesem Browser funktionieren würden. Die meisten Benutzer müssten sich dann zwischen ihrem favorisierten Browser ohne Unterstützung und dem „fremden“ Browser entscheiden. Neuerdings besteht allerdings Anlass zur Hoffnung, eine derartig genaue Studie wiederholen zu können: Der Browser Mozilla [Moz 2002a] wurde in der Version 1.0.1 im August

2002 fertiggestellt und ist den Browsern Internet Explorer und Netscape Navigator bezüglich des Funktionsumfangs und der Benutzungsoberfläche sehr ähnlich. Da dieser Browser im Rahmen eines Open-Source-Projekts entwickelt wurde, ist der Quellcode frei verfügbar und könnte, wie im Fall von Tauscher und Greenberg, modifiziert werden um alle Benutzeraktionen zu speichern.

Cockburn und McKenzie [Coc 2000] werteten in ihrer Studie zum Benutzerverhalten im Web die History- und Bookmark-Dateien aus, die der Netscape Navigator standardmäßig anlegt. Viele der Benutzeraktionen lassen sich allerdings auf diese Weise nicht eindeutig erkennen, wie z. B. die Aktion des Benutzers, die zum Seitenwechsel führte. Daher eignet sich dieses Verfahren nur bedingt, um die Probleme des Benutzers beim Browsen zu analysieren.

Aufgrund der momentanen Sachlage, die eine detaillierte Erfassung der Benutzeraktionen beim Browsen nicht gestattet, entstand eine Arbeit, die diese Lücke schließen sollte.

Das daraus entstandene Projekt „Access-Tracking“ soll die Erfassung der Benutzeraktionen bei Verwendung eines Web-Browsers möglichst unabhängig von Browser und Plattform gestatten. Daher wurde dieses Projekt in Java realisiert. Zusammen mit dem Framework Scone [Wei 2002], lassen sich Programme zur Erfassung von Benutzeraktionen, sowie Navigationswerkzeuge als Browsererweiterung implementieren. Der Entwickler wird dabei durch einen Document-Editor zum Verändern von Internet-Seiten, einen Robot zum Durchsuchen des Internets und weitere Komponenten unterstützt.

Das folgende Kapitel führt den Begriff der „Benutzeraktion“ beim Browsen im Web ein und gibt einen allgemeinen Überblick über Verfahren zum Sammeln von Benutzeraktionen.

Im dritten Kapitel werden automatisierte Verfahren besprochen, die auf unterschiedliche Arten die Benutzeraktionen und die Informationen über die aufgerufenen Internet-Seiten ermitteln. Die Kombination aus Java

Applets, JavaScript, einem erweiterten Proxy-Server und einer auswertenden Applikation wird besonders hervorgehoben.

Das vierte Kapitel geht auf die Details und die Realisierung des kombinierten Verfahrens, genannt „Access-Tracking“, ein. Nach einem Überblick über die Funktionsweise des Projektes wird kurz das Framework Scone beschrieben, in das die Lösung eingebettet ist. Darauf folgt die Beschreibung der einzelnen Teilkomponenten. Da mit der gewählten Technik einige Benutzeraktionen nur indirekt ermittelt werden können, werden die Rückschlüsse auf diese Benutzeraktionen ausführlich erklärt.

Die Probleme, die während der Realisierung dieses Projektes auftraten, wie z. B. die Fälle, in denen sich die Browser nicht an die Spezifikationen hielten, werden im fünften Kapitel beschrieben.

Abschließend folgt im sechsten Kapitel eine Zusammenfassung und ein Ausblick auf mögliche weitere Arbeiten, die sich an dieses Projekt anschließen können.

2 Grundlagen der Beobachtung des Benutzerverhaltens

In dieser Arbeit geht es um die Erfassung der Benutzeraktionen beim Browsen im Web. Abschnitt 2.1 erklärt den Begriff „Benutzeraktion“ und gibt an, wozu sie gebraucht werden und welche Aktionen für bestimmte Anwendungen wichtig sind. Im Abschnitt 2.2 werden dann Verfahren vorgestellt, mit denen die Benutzeraktionen beim Surfen im Web ermittelt werden können.

2.1 Benutzeraktionen

Benutzeraktionen allgemein sind alle Aktionen, die der Benutzer ausführt um mit Programmen auf seinem Computer zu interagieren. Dazu gehören sämtliche Eingaben über Maus, Tastatur oder andere Eingabegeräte.

2.1.1 Definition der „Benutzeraktion beim Browsen im Web“

Die Benutzeraktionen beim Browsen im Web umfassen sämtliche Aktionen des Benutzers, die der Interaktion mit dem Web-Browser dienen.

Diese Aktionen lassen sich in *Navigationsaktionen, inhaltsbezügliche Aktionen, Persistierungsaktionen, Organisierungsaktionen* und *browserbezügliche Aktionen* gliedern. Die Navigationsaktionen umfassen alle Aktionen des Benutzers, die dem Aufrufen einer neuen Seite dienen. Dazu gehören unter anderem angeklickte Links, eingegebene URLs und Sprünge innerhalb der besuchten Seiten. Zu den inhaltsbezogenen Aktionen zählen die, die den Inhalt der angezeigten Web-Seite betreffen: beispielsweise das Ausfüllen von Formularen. Persistierungsaktionen umfassen das Speichern und Drucken der angezeigten Web-Seite. Das Hinzufügen und Verwalten von Lesezeichen gehört in die Gruppe der Organisierungsaktionen. Bei dem Auswählen von Lesezeichen handelt es sich allerdings um eine Navigationsaktion, da hierbei ein früher besuchtes Dokument erneut aufgerufen wird. Das Öffnen, Schließen und Verschie-

ben von Browserfenstern, sowie das Ändern der Fenstergröße und das Scrollen des Inhalts gehören schließlich zu den browserbezüglichen Aktionen.

2.1.2 Ziele der Erfassung von Benutzeraktionen

Das Web hat sich in den letzten 7 Jahren sehr verändert. Das Informationsangebot ist zwischen 1995 und 2000 annähernd exponentiell gewachsen [ISC 2002] und es wurden viele neue Web-Dienste entwickelt und verbessert, um den Benutzer beim Surfen zu unterstützen. Beispiele dafür sind Suchmaschinen, die mit Hilfe eines Rankings die „besseren" Suchergebnisse zuerst zeigen [Goo 2002]. Lediglich die Benutzungsoberflächen der Web-Browser haben sich in der selben Zeit kaum verändert, obwohl auch hier viele Mängel seit längerem bekannt sind [Bie 1997].

Trotz der verbesserten Web-Dienste haben die Benutzer weiterhin Probleme beim Auffinden von Informationen und bei der Orientierung im Web. Das haben Untersuchungen, wie GVU's Tenth WWW User Survey [GVU 1998], ergeben.

Analysen des Benutzerverhaltens können darüber Aufschluss geben, wo der Benutzer auf Probleme beim Surfen trifft. Aus den erkannten Problemen können Browser-Erweiterungen oder Navigationswerkzeuge abgeleitet werden, die den Benutzer beim Surfen unterstützen sollen. Um entsprechende Verhaltensmuster erstellen zu können, werden die Aktionen des Benutzers benötigt.

Diese Aktionen werden ebenfalls für Navigationswerkzeuge benötigt, um die Schritte des Benutzers nachvollziehen zu können. Derartige Werkzeuge bieten dem Benutzer beispielsweise eine visuelle History [May 2000] oder Informationen über verlinkte Seiten an [Kre 1998].

2.1.3 Relevante Benutzeraktionen

Welche der Benutzeraktionen relevant sind, hängt von dem Verwendungszweck ab: Beispielsweise reicht bei einer Studie zur „relativen Häufigkeit bestimmter Web-Seiten" die Erkennung jeder aufgerufenen

URL aus, wogegen bei Studien zur „Benutzung von Web-Browsern“ jede Eingabe über Tastatur oder Maus relevant ist und gespeichert werden muss.

Um die Probleme der Benutzer beim Browsen im Web bestimmen zu können, sollten alle Benutzeraktionen, die die Navigation betreffen, erkannt werden. Des Weiteren ist die aktuelle URL relevant, um später die vom Benutzer besuchten Seiten rekonstruieren zu können. Zum Erkennen der Probleme, die der Benutzer mit der Benutzungsoberfläche des Browsers hat, sollten zusätzlich die Aktionen zum Organisieren der gefunden Informationen gespeichert werden. Das sind z. B. das Hinzufügen und das Verwalten der Lesezeichen und ähnlicher Browser-Funktionen[1]. Auch die Aktionen, die den Inhalt der angezeigten Seiten betreffen, z. B. Textsuche auf der Seite oder ausfüllen eines Formulars, können für derartige Studien relevant sein.

2.2 Methoden der Erfassung von Benutzeraktionen

Es gibt viele Möglichkeiten, die Benutzeraktionen beim Browsen im Web zu ermitteln. Die gängigsten werden hier vorgestellt und es wird kurz auf entsprechende Arbeiten eingegangen.

Einige der folgenden Verfahren werden persönlich durchgeführt. Da dies sehr zeitaufwändig ist, eignen sich derartige Verfahren nur für Studien mit kleinen Teilnehmergruppen. Automatisierte Verfahren erkennen meist nicht alle Benutzeraktionen, dafür lassen sich aber die Aktionen sehr großer Gruppen ermitteln und auswerten.

2.2.1 Befragungen

Eine Möglichkeit, das Benutzerverhalten im Internet zu ermittelt, ist die Befragung der Benutzer. Dazu wird ein Fragenkatalog entworfen, der alle

[1] Der Internet Explorer bietet beispielsweise eine Button-Leiste an, über die der Benutzer direkt zu anderen Seiten springen kann. Im Gegensatz zu den „normalen“ Lesezeichen muss der Benutzer nicht erst das entsprechende Menü öffnen.

gewünschten Themenbereiche abdeckt. Die Befragung kann entweder persönlich oder anonym, beispielsweise per E-Mail über das Internet erfolgen. Persönliche Befragungen sind sehr zeitaufwändig und damit ungeeignet, eine große Anzahl von Personen zu befragen. Befragungen per E-Mail erlauben eine größere Anzahl befragter Personen, allerdings lassen sich die Antworten nur bedingt automatisiert aus den Antwort-E-Mails extrahieren. Das liegt daran, dass das Konzept des Dienstes E-Mail entsprechende Möglichkeiten, wie formatierte Eingabefelder oder Überprüfung der Eingaben auf Vollständigkeit, nicht zur Verfügung stellt.

Pitkow und Recker [Pit 1995] wählten einen anderen Ansatz, um den Formatierungsproblemen in den Antwort-E-Mails zu entgehen: Der Benutzer rief mit seinem Browser eine Internet-Seite auf, die ihm zu jeder Frage ein entsprechendes Eingabefeld oder eine Auswahlmöglichkeit bot, das die Eingaben, wenn möglich, auf Gültigkeit prüfte. Mit Hilfe von CGI-Programmen wurden die Antworten entgegengenommen und in einer Datenbank gespeichert. Einen weiteren Vorteil dieser Befragungsmethode stellt die Möglichkeit der adaptiven Fragen dar. Durch sie lassen sich, abhängig von bisherigen Antworten, bestimmte Fragen überspringen. Gibt der Benutzer beispielsweise an, er würde keine eigenen Inhalte im Internet präsentieren, werden Fragen zu entsprechenden Hilfsprogrammen ausgelassen. Auf diese Weise muss sich der Befragte nicht mit Fragen auseinandersetzen, die ihn nicht betreffen.

Bei Befragungen muss beachtet werden, dass die Antworten subjektiv sind. Während dies in einigen Fällen gewünscht ist, wie z. B. bei der Frage nach den „Problemen der Benutzer beim Browsen im Web“, erhält man etwa bei der Frage nach dem Zeitaufwand des „Surfens aus Spaß“ nur sehr grobe Schätzungen. Im zweiten Fall ist Verfahren des nächsten Abschnitts vorzuziehen.

2.2.2 Beobachten des Benutzers und „Thinking-Aloud“

Die Beobachtung eines Benutzers bei seiner Arbeit im Internet kann sehr genaue Daten liefern. Beispielsweise kann die Zeit, die der Benutzer im Internet verbringt, genau festgehalten werden. Des Weiteren lässt sich er-

kennen, wie oft der Benutzer auf Links klickt oder den Zurück-Button benutzt. Auch die Namen der aufgerufenen Seiten und die Zeit, die der Benutzer auf jeder Seite verbringt, lässt sich festhalten.

Dieses Verfahren wird gern mit dem „laut denken“ (Thinking-Aloud) [Lew 1982] kombiniert. Der Benutzer wird gebeten, seine Gedanken zu verbalisieren, damit seine Überlegungen später nachvollzogen werden können. Dadurch erfährt man, was den Benutzer zu seinem nächsten Schritt bewegt hat oder wo der Benutzer auf Probleme traf.

Mit Hilfe dieses kombinierten Verfahrens haben Byrne, John und Joyce [Byr 1999] in ihrer Studie „A day in the life of ten WWW users“ 10 Personen beim Surfen beobachtet. Bei jeder der Personen wurde für einen Tag eine Videokamera installiert, die auf den Computerbildschirm gerichtet war. Die Testpersonen wurden gebeten, die Kamera einzuschalten, sobald sie im Internet arbeiteten und ihre Gedanken laut zu äußern. Byrne, John und Joyce haben auf diese Weise knapp 6 Stunden Videoprotokoll erhalten, dass sie bezüglich der Benutzeraktionen und der Benutzerabsichten analysiert haben.

Die Kombination aus Beobachten und „Thinking-Aloud“ ist sehr effektiv, da viele Informationen über das Benutzerverhalten ermittelt werden können. Allerdings ist dieses Verfahren sehr zeitaufwändig, da für jede Testperson mindestens ein „Beobachter“ abgestellt werden muss, der die relevanten Aktionen festhält. Besonders die Analyse der Protokolle kostet sehr viel Zeit, denn die Probleme und Beschwerden der Benutzer müssen katalogisiert werden. Daraufhin werden die Ursachen der Probleme ermittelt und Lösungsansätze erarbeitet.

2.2.3 Eye-Tracking

Durch das Eye-Tracking lässt sich anhand der Augenbewegungen erkennen, welcher Bereich des Bildschirms von der Testperson betrachtet wird. Zusätzlich kann unterschieden werden, ob die Testperson Text liest, ihn überfliegt oder etwas Spezielles sucht. Auf diese Weise wird der Aufbau von Internet-Seiten getestet [Sch 1998a] oder die Wirksamkeit

von Werbebannern bestimmt. Ein weiteres Einsatzgebiet für das Eye-Tracking ist das Testen von Benutzungsoberflächen.

Es gibt zwei Möglichkeiten, die Bewegungen der Augen zu ermitteln: das Reflektionsverfahren [Sch 1998] und die Electro-Oculographie (EOG) [Kau 1993]. Im Fall der Reflektionsmethode werden an den Ecken des Monitors infrarote Lichtquellen angebracht, die dem System die Grenzen des Monitorbildes zeigen. Die Augen des Probanten werden mit Hilfe von zwei Kameras beobachtet. Das infrarote Licht wird von den Augen reflektiert und ebenfalls von den Kameras aufgenommen. Aus der Position der reflektierten infraroten Marken relativ zur Pupille lässt sich die Blickrichtung errechnen.

Bei der Electro-Oculographie lassen sich die Augenbewegungen durch die Muskelströme der Augen ableiten. Mit Hilfe von Elektroden werden die Muskelströme links und rechts der Augen und darüber und darunter abgenommen. Eine Elektronik errechnet daraus die momentane Position der Augen.

2.2.4 Automated Cognitive Walkthrough for the Web

Eine weitere Möglichkeit, die Benutzbarkeit von Internet-Seiten zu bestimmen, ist der „Automated Conginitive Walkthrough for the Web“[2] [Bla 2002]. Dabei werden insbesondere die Bezeichnungen der Links der einzelnen Seiten daraufhin getestet, ob neue Benutzer zielsicher die gewünschte Information finden.

Um dies herauszufinden, werden die Testpersonen gebeten, innerhalb der entwickelten Web-Site[3] bestimmte Informationen zu finden. Jede besuchte Seite wird gespeichert, um nachträglich die Links erkennen zu können,

---

[2] Dabei handelt es sich um eine Variation des „Cognitive Walkthrough“, einem Verfahren aus der Softwareentwicklung, mit dem Benutzungsoberflächen auf ihre Benutzbarkeit hin getestet werden.

[3] „Web-Site“ bezeichnet ein zusammenhängendes Angebot mehrerer Internet-Seiten.

die den Probanten in die falsche Richtung gelenkt haben. Schlecht bezeichnete Links werden verändert und die Testreihe beginnt erneut. Die Testreihen werden so lange fortgesetzt, bis neue Benutzer die benötigten Informationen ohne langes Suchen finden.

Der automatisierte Teil an diesem Verfahren ist das Erkennen der besuchten Seiten. Dies lässt sich beispielsweise durch die Log-Dateien des Web-Servers erreichen. Die meisten Web-Server lassen sich so einstellen, dass jeder Zugriff zusammen mit der IP-Adresse des Benutzers gespeichert wird. Auf diese Weise lassen sich die nacheinander besuchten Seiten (Walkthrough) eines bestimmten Benutzers nachvollziehen.

2.2.5 Abfangen der Benutzeraktionen mit dem Web-Browser

Der Browser ist das Bindeglied zwischen dem Benutzer und dem Web. Da liegt es nahe, die Benutzeraktionen, die der Browser entgegen nimmt, durch den Browser abzufangen und zu speichern. Auch die Informationen über angezeigte Internet-Seiten könnten mit URL und Zeitpunkt in einer Datenbank abgelegt werden. Allerdings gibt es keinen einfachen Weg, derartige Informationen abzufangen, da die aktuellen Browser diese nicht ausgeben. Im dritten Kapitel werden Verfahren besprochen, die Teile der gewünschten Informationen ermitteln.

1995 modifizierten Catledge und Pitkow [Cat 1995] den Web-Browser XMosaic, um die direkten Benutzeraktionen und Informationen über die angezeigten Seiten erfassen zu können. Zwei Jahre später veränderten Tauscher und Greenberg [Tau 1997] eine neuere Version des XMosaic-Browsers. Dank dieser Modifikationen konnten Informationen über die angezeigte Internet-Seite, das Anklicken aller Buttons und alle Tastatureingaben des Benutzers abgefangen und gespeichert werden. Seitdem wurde keine Studie mit derartig genauen Daten mehr durchgeführt.

Cockburn und McKenzie haben in ihrer Studie „What Do Web Users Do“ [Coc 2000] History-Dateien analysiert, um das Benutzerverhalten beim Browsen im Web zu untersuchen. Sie benutzten die History- und die Bookmark-Datei des Netscape Navigators, die dieser standardmäßig

erzeugt. Der Browser legt darin unter anderem die URL jeder aufgerufenen Internet-Seite und den Zeitpunkt des Aufrufes ab. Aus der Bookmark-Datei lassen sich die Lesezeichen des Benutzers sowie der Zeitpunkt des letzten Zugriffs für jeden Eintrag auslesen. Allerdings werden Internet-Seiten, die über die Vor- oder Zurück-Taste aufgerufen wurden, in diesen Dateien nicht vermerkt. Auch lässt sich auf diese Weise nicht ermitteln, ob die Seite durch das Anklicken eines Links oder durch das Eingeben einer neuen URL ausgewählt wurde. Im Rahmen ihrer Studie haben Cockburn und McKenzie die Erlaubnis von 17 Benutzern eingeholt, deren History-Dateien der letzten vier Monate auswerten zu dürfen. Normalerweise besteht bei Benutzern, die wissentlich an Studien zum Benutzerverhalten teilnehmen, die Möglichkeit, dass sie sich anders verhalten als in unbeobachteten Momenten („Hawthorne“-Effect [Dra 2001]). Diese Möglichkeit haben Cockburn und McKenzie ausgeschlossen, indem sie History-Dateien aus der Zeit verwendeten, in der die Benutzer nichts von der Teilnahme an einer Studie wußten.

3 Lösungsansätze zur automatisierten Erfassung von Benutzeraktionen

In diesem Kapitel werden zunächst die relevanten Daten eingeführt, die für eine statistische Auswertung der Benutzeraktionen von Interesse sind. Zusätzlich zu den üblichen Daten wie Benutzeraktion, URL, vorige URL, Verweil- und Ladezeit, müssen noch weitere Angaben wie Browser-Fensternummer und Benutzernummer gespeichert werden. Dadurch ist es nachträglich möglich, die Seiten zu ermitteln, die der Benutzer an einem bestimmten Browser-Fenster aufgerufen hat.

Im Abschnitt 3.2 wird kurz die Problematik des Abfangens der Benutzerereignisse beschrieben.

In den darauf folgenden Abschnitten werden Verfahren beschrieben, mit denen die benötigten Daten gesammelt werden können. Abschnitt 3.3 beginnt mit dem von der Firma Spyglass Inc. entwickelten Software-Development-Interface, das eine Schnittstelle für die Interprozess-Kommunikation definiert. Diese wurde in vielen Browsern implementiert und erlaubt anderen Programmen, Daten auszulesen und Seiten aufzurufen. Danach wird im Abschnitt 3.4 eine Schnittstelle des Internet Explorers von Microsoft beschrieben, die über das Browser-Helper-Object eine weit mächtigere Möglichkeit bietet, Daten auszutauschen und Ereignisse abzufangen. Im Abschnitt 3.5 wird ein Verfahren beschrieben, das durch abgefangene Fensterereignisse das Anklicken von Button-Leisten und Menüs erkennt. Abschnitt 3.6 zeigt einen anderen Weg, um die Benutzeraktionen zu ermitteln. Hier fängt ein selbst entworfener oder modifizierter Browser alle gewünschten Ereignisse ab, um sie zu speichern oder auf beliebigem Wege weiter zu schicken. Die letzten beiden beschriebenen Techniken, Java-Applets (Abschnitt 3.7) und JavaScript (Abschnitt 3.8), laufen innerhalb der Browser ab und haben auf diese Weise Zugriff auf viele Daten der angezeigten Internet-Seiten. Auch einige Benutzeraktionen lassen sich auf diese Weise abfangen.

Im Abschnitt 3.9 werden mögliche Kombinationen der erwähnten Verfahren besprochen und abschließend im Abschnitt 3.10 die Lösung bestimmt, die in den folgenden Kapiteln ausgiebig behandelt wird.

3.1 Relevante Daten

Zur Steuerung von Navigationswerkzeugen und für statistische Auswertungen sind Daten über die Aktionen des Benutzers und die besuchten Internet-Seiten relevant. Dazu gehört die Art und Weise, auf die der Benutzer von einer Seite zur nächsten springt. Mögliche Aktionen sind das Eingeben einer neuen URL im Adressfeld des Browsers, das Anklicken eines Links, die Auswahl eines Bookmarks oder das Benutzen des Zurück-, Vor- oder Aktualisieren-Buttons um innerhalb der History des Browsers zu navigieren.

Um zu ermitteln, wie oft der Benutzer einzelne Seiten aufgerufen hat, muss die aktuelle URL gespeichert werden. Das Speichern der vorigen URL hilft, die Reihenfolge der besuchten Seiten zu rekonstruieren. Des Weiteren sind die Ladezeit sowie die Verweildauer auf einer Internet-Seite interessant, da in früheren Studien gezeigt wurde, das einige Seiten wieder verlassen wurden, bevor sie vollständig geladen waren [Coc 2000, S. 10,11].

Alle gesammelten Daten werden in der Datenbank abgelegt um sie zu einem späteren Zeitpunkt analysieren zu können. Deshalb müssen noch weitere Informationen gespeichert werden um die einzelnen Schritte des Benutzers nachvollziehen zu können.

So muss bei mehreren gleichzeitig geöffneten Browser-Fenstern jedes Fenster eine eindeutige Kennzeichnung haben, um die einzelnen Aktionen den entsprechenden Fenstern zuordnen zu können. Dies ist besonders bei mehreren Anzeigebereichen in einer Internet-Seite (Frames [Mün 2001]) unerlässlich.

Jede Aktion muss zusammen mit einer eindeutigen Benutzerkennung gespeichert werden, damit sie im Nachhinein den einzelnen Benutzern zugeordnet werden kann.

3.2 Abfangen der Benutzerereignisse

Das Abfangen der Benutzeraktionen beim Browsen im Web ist die Voraussetzung um später Statistiken zum Benutzerverhalten beim Browsen erstellen zu können. Des Weiteren werden derartige Aktionen für den Betrieb von Navigationswerkzeugen benötigt, die den Benutzer im Web unterstützen.

Die Aktionen des Benutzers, wie beispielsweise das Anklicken eines Links oder die Eingabe einer neuen URL, erzeugen Ereignisse im Browser. Diese Ereignisse lösen dann die entsprechenden Funktionen aus. Da der Browser derartige Ereignisse nicht an andere Programme weitergibt, müssen Wege gefunden werden, die ausgelösten Ereignisse abzufangen. In den folgenden Abschnitten werden Verfahren vorgestellt, die die Aktionen des Benutzers auf unterschiedliche Art und Weise ermitteln. Jedes dieser Verfahren hat seine Stärken und Schwächen.

3.3 Spyglass SDI

Das erste hier besprochene Verfahren zum Abfangen von Ereignissen stammt von der Firma Spyglass Inc[4]. Die Firma Spyglass definierte Mitte der 90er ein Software-Development-Interface [Spy 1995], das als Grundlage für die Interprozess-Kommunikation (IPC) von Web-Browsern diente. Neben vielen anderen implementierten auch Microsoft [Mic 2000] und Netscape [Net 1997] diese Schnittstelle in ihren Browsern.

Um die Interprozess-Kommunikation auf den verschiedenen Plattformen zu realisieren, wurden die jeweils verfügbaren IPC-Mechanismen verwendet: Unter Windows wurde die Schnittstelle zunächst durch Dynamic Data Exchange (DDE) [Mic 2000], [Net 1997] realisiert, später durch Object Linking and Embedding (OLE Automation). Auf Macintosh-Rechnern wurden die AppleEvents benutzt [Spy 1995].

---

[4] Die Firma Spyglass Inc. wurde von der Firma OpenTV aufgekauft und hat daher den Betrieb der Internet-Seite www.spyglass.com eingestellt.

Die Fähigkeiten dieser Schnittstelle sind allerdings begrenzt und variieren leicht, abhängig von den Möglichkeiten des IPC-Mechanismus. Die Browser können von anderen Programmen gestartet und beendet werden, erlauben das Aufrufen neuer Seiten und das Abfragen von Browser-Informationen wie der aktuellen URL. Registrierte Programme werden über bestimmte Ereignisse informiert, z. B. über die fertig geladene Internet-Seite.

Direkte Benutzeraktionen wie gedrückte Tasten oder Mausbewegungen können nicht als Ereignisse abgefangen werden. Auch werden weder geklickte Links noch ausgewählte Lesezeichen erkannt. Ein weiterer Nachteil der Spyglass-Schnittstelle ist, dass das Programm zum Abfangen der Informationen auf jeder Plattform neu implementiert werden muss, um die unterschiedlichen Kommunikationsmechanismen anzusprechen.

3.4 Der Internet Explorer und das Browser-Helper-Object

Eine weitere Möglichkeit zur Kommunikation mit dem Web-Browser ist das so genannte Browser-Helper-Object [Rob 1999] des Internet Explorers von Microsoft. Dabei handelt es sich um ein selbst geschriebenes Programm in einer DLL-Datei (Dynamic Link Library), die vom Internet Explorer eingebunden und gestartet wird. Dieses Programm erhält vom Internet Explorer einen Zeiger auf das **`IWebBrowser2`**-Interface, das den Zugriff auf den Browser erlaubt. Über dieses Interface kann der Browser gesteuert und dessen Ereignisse angefordert werden. Die abgefangenen Ereignisse können durch die DLL aufbereitet und auf beliebigem Wege an ein anderes Programm zur Auswertung weiter geschickt werden.

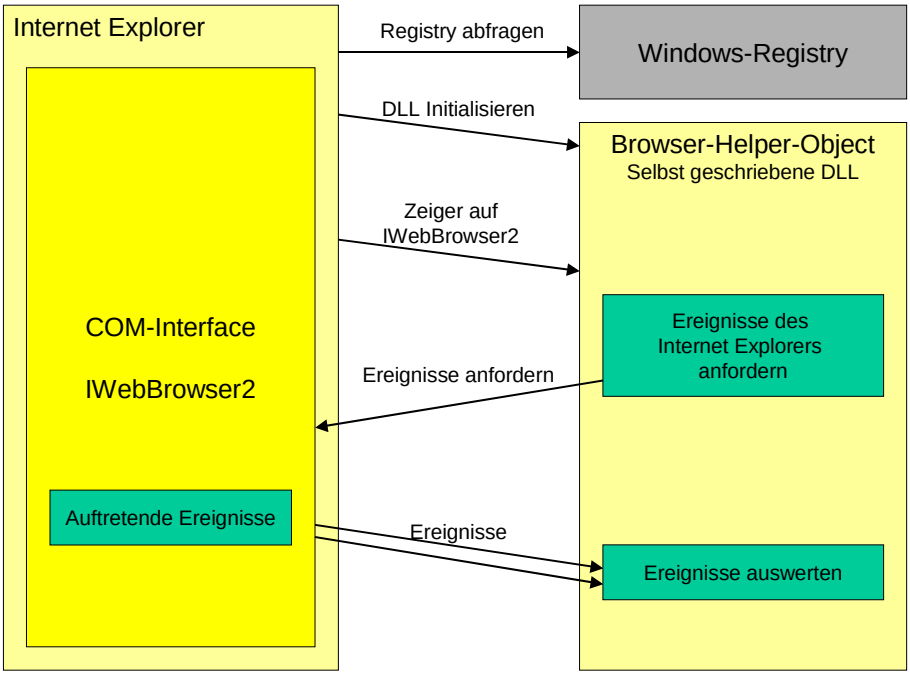

Abbildung 3.1: Erfassung der Benutzeraktionen mit JavaScript und Applet

Das **`IWebBrowser2`**-Interface wird von dem Component Object Model (COM) [Mic 2002] bereitgestellt, das Microsoft mit dem Betriebssystem Windows 95 eingeführt und seit dem ständig erweitert hat. COM bietet Entwicklern die Möglichkeit, Windows-Komponenten in eigene Projekte einzubinden oder selbst Komponenten zu erstellen und auf diese Weise die Wiederverwendbarkeit von Programmteilen zu erhöhen. Diese Komponenten lassen sich später austauschen, ohne das Projekt neu kompilieren zu müssen.

Auf das Interface einer laufenden Internet-Explorer-Instanz kann nicht, wie z. B. bei Microsoft Word, über eine Liste laufender Objekte zugegriffen werden. Dies lässt sich nur über das Browser-Helper-Object (BHO) [Rob 1999, Kapitel 12] erreichen. Zunächst muss eine DLL programmiert werden, die zwei Methoden bereitstellt: Die erste Methode wird vom Internet Explorer bei jedem Start aufgerufen und übergibt der DLL auf diese Weise einen Zeiger auf das **`IWebBrowser2`**-Interface. Über dieses Interface können nun die Ereignisse des Browsers angefordert werden. Im Falle eines Ereignisses ruft der Internet Explorer die

zweite Methode der DLL auf, die die Informationen über das Ereignis aufbereitet und auf beliebigem Wege an ein anderes Programm zur Auswertung weiter schickt.

Um dem Internet Explorer mitzuteilen, welche DLL er einbinden soll, muss diese an einer bestimmten Stelle der Windows-Registry eingetragen werden. Jede Instanz des Internet Explorers liest diese Stelle der Windows-Registry aus und initialisiert die entsprechende DLL während des Startvorgangs. Um die Ereignisse mehrerer Instanzen des Internet Explorers unterscheiden zu können, lässt sich über das **`IWebBrowser2`**-Interface die Fensternummer des Internet Explorers abfragen. Dahinter verbirgt sich eine Zahl, die das Hauptfenster des Browsers eindeutig identifiziert.

Die Ereignisse des Internet Explorers informieren über angewählte Seiten, Beginn und Ende des Ladevorgangs, Versionsnummer des Browsers, Änderung des Titelleisten- und Statustextes und vieles mehr.

Die einzelnen Buttons des Browsers lösen im BHO jedoch keine eigenen Ereignisse aus und können somit nicht abgefangen werden.

3.5 Abfangen der Fensterereignisse

Eine Möglichkeit die Ereignisse der einzelnen Buttons abzufragen, bieten die Fensterereignisse in fensterorientierten Betriebssystemen. Jede Mausbewegung, jedes Anklicken und jede gedrückte Taste lösen in fensterorientierten Betriebssystemen Ereignisse aus. Diese Ereignisse werden vom Betriebssystem an die entsprechende, z. B. unter dem Mauszeiger liegende, Applikation weitergegeben. Von dort aus wird das entsprechende Fenster und dann der angeklickte Button benachrichtigt. Die meisten Betriebssysteme bieten Hook-Funktionen [Mic 2002a] an, mit denen sich Programme in die Kette des Nachrichtenflusses der Ereignisse einhaken können, um über sämtliche Fensterereignisse informiert zu werden. Ursprünglich waren die Hook-Funktionen für die Fehlersuche gedacht, sie lassen sich aber auch zum Abfangen von Nachrichten der geklickten Buttons benutzen.

Die meisten Browser setzen sich aus mehreren Fenstern zusammen. Bei Microsofts Internet Explorer ist die Button-Leiste, die den Zurück-, Vor- und Aktualisieren-Button enthält, ein eigenes Fenster. Mit Hilfe von Fensterereignissen informiert die Button-Leiste die Applikation über die Mausbewegungen. Ebenso teilen die meisten Menüs und Favoritenlisten ihre Aktionen dem Programm mit. All diese Ereignisse können mit Hilfe der Hook-Funktionen abgefangen werden.

Die Hook-Funktionen fangen die Nachrichten für alle existierenden Fenster des Rechners ab. Da die Ereignisse der anderen Programme unwichtig sind, müssen die Nachrichten des Internet Explorers heraus gefiltert werden. Jedes Fenster hat eine eindeutige Nummer, die den Nachrichten als Adresse dient. Die Fensternummern des Internet Explorers lassen sich ermitteln und mit den „Adressen“ der einzelnen Fensternachrichten vergleichen, um die wichtigen Nachrichten zu erkennen. Da der Internet Explorer aus mehreren Fenstern besteht, die interessante Nachrichten empfangen, müssen entsprechend viele Fensternummern gespeichert werden. Jede neue Instanz des Internet Explorers erhält andere Fensternummern, daher muss für jede Instanz ein neuer Satz Fensternummern gespeichert werden.

Sollen die Ereignisse eines neuen Browsers abgefangen werden, dann müssen zunächst die Fenster identifiziert werden, die die interessanten Buttons und Menüs enthalten. Die auftretenden Nachrichten bestehen im Allgemeinen nur aus Zahlenpaaren, die den einzelnen Buttons oder Menüeinträgen zugeordnet werden müssen. Bei dem Internet Explorer muss sogar, abhängig von der Version des Browsers, auf unterschiedliche Nachrichten reagiert werden, da einige Buttons in bestimmten Versionen andere Nachrichten schicken.

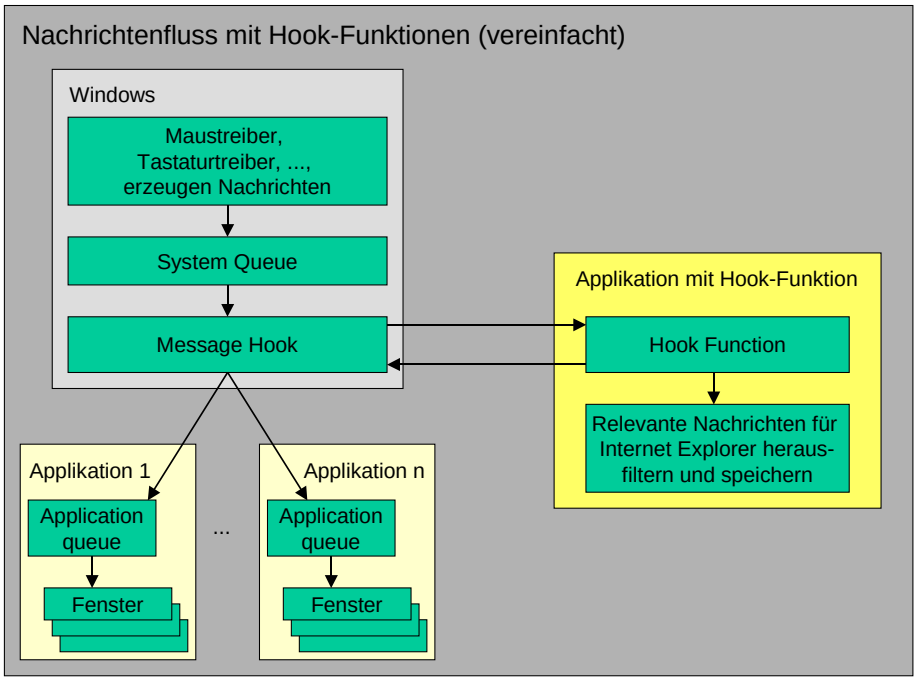

Abbildung 3.2: Hook-Funktionen am Beispiel von Microsoft Windows

Das Abfragen von Informationen über die angezeigte Internet-Seite, wie die aktuelle URL, enthaltene Links oder Formulare, ist auf diese Weise allerdings nicht möglich. Auch Ereignisse wie Beginn oder Ende des Ladevorgangs lassen sich durch Fensterereignisse nicht abfangen.

3.6 Modifizieren eines Browsers

Eine weitere Möglichkeit, die Benutzeraktionen zu ermitteln, ist ein selbst programmierter bzw. modifizierter Browser. Dieser kann alle interessanten Benutzeraktionen speichern oder auf beliebigem Wege weitergeben. Genauso lassen sich auch alle interessanten Daten über die aktuelle Internet-Seite auslesen. Beim Speichern der Daten können zusätzlich Informationen über den jeweiligen Benutzer und das von ihm benutzte Fenster abgelegt werden, um später die Aktionen zuordnen und rekonstruieren zu können.

Ein selbst programmierter Browser bedeutet allerdings einen großen Programmieraufwand, denn mit einem Programm, das HTML interpretieren kann, ist es nicht getan. Angefangen bei allen gängigen Grafikformaten,

über JavaScript und Applets, bis hin zu Flash und anderen Erweiterungen sollte der selbst geschriebene Browser alle Möglichkeiten bieten, die die gängigen Browser vorgeben.

Es besteht die Möglichkeit, die Browser-Komponente des Internet Explorers mit Hilfe des Common Object Model (COM) [Mic 2002] in eine eigene Applikation einzubinden. Die Browser-Komponente würde dann das Anzeigen der Internet-Seiten erledigen und die selbst programmierten Buttons könnten beim Anklicken bestimmte Funktionen zur Ereignisbehandlung aufrufen. Auf diese Weise würden zwar alle Benutzeraktionen auf der selbst geschriebenen Benutzungsoberfläche abgefangen werden, angeklickte Links lassen sich der Browser-Komponente jedoch nicht entlocken.

Trotz aller Bemühungen kann es passieren, dass sich der Benutzer an dem neuen Browser anders verhält als an seinem gewohnten Browser – das ist durch Studien belegt [Coc 2000, S. 4]. Dieses veränderte Verhalten kann mögliche Statistiken verfälschen.

Ein Weg mit geringerem Aufwand verspricht die Modifikation eines bestehenden und bewährten Browsers. Um einen Browser zu modifizieren, muss der Quellcode erhältlich sein. Hier bietet sich der Web-Browser Mozilla [Moz 2002] an, dessen Quellcode im Rahmen eines Open-Source-Projekts frei verfügbar ist und dessen Funktionsumfang dem der Browser von Netscape und Microsoft ähnelt. Der Quellcode kann nun nach belieben modifiziert werden, um die interessanten Daten und Ereignisse zu speichern oder an andere Programme zur Auswertung weiterzugeben.

Auch beim modifizierten Browser kann es passieren, dass der Benutzer sich anders als mit seinem gewohnten Browser verhält und dadurch mögliche Statistiken verfälscht.

3.7 Java Applets

Die zuvor beschriebenen Verfahren sammeln ihre Daten, indem sie „von außen“ mit dem Browser kommunizieren oder Ereignisse abfangen bevor

sie den Browser erreichen. In diesem und dem nächsten Abschnitt folgen zwei völlig andere Ansätze: Hier werden die Daten durch JavaScript-Code und Java Applets verarbeitet, die „innerhalb“ des Browsers als Teil der aufgerufenen Internet-Seite ablaufen.

Java Applets sind kompilierte Programme, die in Internet-Seiten eingebettet sind und in der Java-Umgebung des Browsers ausgeführt werden. Diese Umgebung, das Sandkasten-Modell, hindert das Applet daran auf Daten und Programme außerhalb des Browsers zuzugreifen. Applets können grafische Benutzungsoberflächen besitzen, Dateien aus dem Internet laden und Socket-Verbindungen zu bestimmten Servern aufbauen.

Applets sollten, ebenso wie Java Applikationen, plattformunabhängig sein, in der Praxis verhalten sie sich in den einzelnen Browsern und ihren Plattformen allerdings sehr unterschiedlich. So sollten Applets laut Java-Spezifikation [Sun 2001] erkennen, ob die Seite, in der sie eingebettet sind, über den Zurück- oder Vor-Button aufgerufen wurde. Leider hält sich Microsofts Internet Explorer nicht an diese Spezifikation. Dieser und weitere Fälle, in denen sich die Browser nicht so verhalten, wie man es erwarten würde, werden im fünften Kapitel behandelt.

Bei der Programmierung des Applets können mehrere Methoden überschrieben werden, um bestimmte Ereignisse abzufangen. Besonders interessant sind hier die Methoden **`init()`**, **`start()`** und **`stop()`**, die der Browser beim Initialisieren und Starten des Applets und beim Verlassen der Internet-Seite aufruft. Auf diese Weise erkennt das Applet das Erstellen und das Verlassen einer Internet-Seite und kann diese Ereignisse per Socket-Verbindung an eine Applikationen zur Auswertung schicken.

Die Buttons der Browser, angeklickte Links oder abgeschickte Formulare lassen sich nicht direkt per Applet abfragen.

3.8 JavaScript

Netscape entwickelte die Skriptsprache JavaScript als Erweiterung des Netscape Navigators und erhob sie 1995 mit Hilfe der European Computer Manufacturers Association (ECMA) zum Standard ECMA-262 [Ecm 1999]. Die JavaScript-Version 1.3, die Netscape in die Navigator-Versionen 4.06 – 4.7 integriert hat, wird vom Internet Explorer ab 5.0 weitgehend verstanden. Es gibt aber stellenweise große Unterschiede zwischen den JavaScript-Implementationen der verschiedenen Browser-Produkte. Um diesem Missstand auf Dauer abzuhelfen, erarbeitete das W3-Konsortium ein allgemeines Modell für die Objekte eines Dokuments, das unter dem Namen „Document Object Model“ (DOM) [W3c 2002] seit Oktober 1998 offizielle W3-Empfehlung ist. Das DOM definiert den Zugriff und die Manipulationsmöglichkeiten der einzelnen Objekte eines Dokuments. Die aktuelle JavaScript-Version 1.5 unterstützt DOM zum großen Teil und wurde von Netscape in den Navigator 6.0 integriert. Microsofts Internet Explorer 5.5 setzt DOM in ähnlichem Maße um.

JavaScript-Befehle werden direkt in die HTML-Seiten eingebettet oder in einer separaten Datei abgelegt. Die Web-Browser erkennen die JavaScript-Sektionen anhand von speziellen HTML-Tags und interpretieren diese.

Mit JavaScript kann man viele der gesuchten Ereignisse abfangen: Links (Verweise auf andere Internet-Seiten) und Formulare werden in der HTML-Datei durch HTML-Tags definiert. Diese Tags können um Attribute erweitert werden, die im Fall eines Ereignisses JavaScript-Anweisungen oder Funktionsblöcke aufrufen. Zum Abfangen von angeklickten Links muss das Link-Tag um ein „**`onClick`**“-Attribut mit entsprechendem Funktionsaufruf erweitert werden. Sobald ein solcher Link angeklickt wird, führt der JavaScript-Interpreter die entsprechenden Anweisungen aus. Auf ähnliche Weise können Formulare beim Abschicken ihres Inhalts JavaScript-Funktionen aufrufen, indem das Formular-Tag um das „**`onSubmit`**“-Attribut erweitert wird. Mit Hilfe der Attribute „**`onLoad`**“ und „**`onUnload`**“ im Body-Tag der HTML-Seite können das

Ende des Ladevorgangs der Seite und das Verlassen der Seite mit JavaScript abgefangen werden.

JavaScript stellt eine Reihe vordefinierter Variablen zur Verfügung, über die man viele interessante Daten der aktuellen Internet-Seite und des Browsers abfragen kann. Zu den abfragbaren Daten gehören die aktuelle und die vorherige URL, ob die Seite aus mehreren Anzeigebereichen (Frames) besteht, wie viele Bereiche die Seite enthält und welche Namen die einzelnen Bereiche haben. Bei Netscape 4.x kann man per JavaScript sogar erkennen, ob eine Seite über die Lesezeichen (Bookmarks) aufgerufen wurde.

Auf die vom Benutzer angeklickten Buttons lässt sich von JavaScript aus nicht zugreifen. Allerdings lassen sich die wichtigsten Ereignisse, wie das Springen innerhalb der bereits besuchten Seiten oder das Eingeben neuer URLs, durch Rückschlüsse ermitteln. Ob der Benutzer zum Springen in der History den Zurück-Button oder den entsprechenden Menüpunkt aus dem Hauptmenü benutzt hat, bleibt solchen Verfahren allerdings verborgen.

3.9 Kombination mehrerer Techniken

In den vorigen Abschnitten wurden Verfahren beschrieben, die aufgrund ihrer differenzierten Ansätze unterschiedliche Aktionen des Benutzers erkennen. Tabelle 3.1 im Abschnitt 3.10 führt die einzelnen Verfahren auf und stellt ihre Möglichkeiten anhand von wichtigen Ereignissen dar.

Einige der Verfahren ermitteln entweder nur die direkten Benutzeraktionen oder können nur Informationen über die angezeigte Internet-Seite liefern. Hier bietet es sich an, derartige Verfahren zu kombinieren. Das Browser-Helper-Object des Internet Explorers und das Abfangen von Fensterereignissen bilden ein „gutes Team“, da sie sich ergänzen. Das Browser-Helper-Object ermittelt, wie oben beschrieben, Informationen über aufgerufene Internet-Seiten, und kann die Nummer des Hauptfensters abfragen. Die von den Hook-Funktionen erhaltenen Nachrichten werden anhand der Fensternummern gefiltert, um nur die Nachrichten für

den Internet Explorer zu erhalten. Mit Hilfe dieser Fensternummern lassen sich also die Ereignisse des Browser-Helper-Objects zu den Fensternachrichten der entsprechenden Internet-Explorer-Instanz zuordnen.

Durch diese Kombination, sie wird im Abschnitt 3.9.1 näher beschrieben, lassen sich fast alle relevanten Daten ermitteln. Leider funktioniert dieses Verfahren nur mit einigen Versionen des Internet Explorers und nur unter Windows. Das bedeutet, dass der Benutzer möglicherweise vor einem ungewohnten Browser sitzt und sich daher anders verhalten könnte.

Eine weitere Möglichkeit bietet die Kombination des Applets mit dem JavaScript. JavaScript kann auf viele Informationen über die angezeigte Seite zugreifen und einige Ereignisse Abfangen. Das Applet kann die ermittelten Daten mit Hilfe von LiveConnect [Sun 2002] entgegennehmen und über eine TCP/IP-Verbindung an ein anderes Programm weitergeben.

Zusätzlich werden noch zwei Funktionseinheiten benötigt: Zum Einen ein Programm, das die Daten empfängt, auswertet und speichert und zum Anderen ein Document-Editor, der das Applet-Tag, den JavaScript-Code und die Modifikationen an den Tags zur Ereigniserzeugung in die HTML-Datei einfügt. Diese Möglichkeit wird in dem Abschnitt 3.9.2 behandelt.

3.9.1 Das Browser-Helper-Object und Fensterereignisse

Die Kombination von abgefangenen Fensterereignissen und dem Datenaustausch mit dem Microsoft Internet Explorer über das Browser-Helper-Object [Rob 1999, Kapitel 12] erlaubt fast alle Benutzeraktionen zu ermitteln, ist aber auf Microsoft Windows und den Internet Explorer beschränkt.

Eine selbst geschriebene DLL, das sogenannte Browser-Helper-Object (BHO), wird aufgrund eines Eintrags in der Windows-Registry von jeder Instanz des Internet Explorers eingebunden. Der Internet Explorer übergibt dem BHO einen Zeiger auf das **`IWebBrowser2`**-Interface, über das das BHO auf die Funktionen des Browsers zugreifen und die Ereignisse

anfordern kann. Die abgefangenen Ereignisse werden aufbereitet und an eine auswertende Applikation geschickt.

Das Abfangen der Fensternachrichten geschieht unter Windows ebenfalls über eine DLL, die die Nachrichten des Internet Explorers heraus filtert und an eine auswertende Applikation schickt. Die genaue Funktionsweise der einzelnen Verfahren wurde bereits in den Abschnitten 3.4 und 3.5 erklärt.

Die empfangenen Ereignisse und Nachrichten lassen sich als neue Fensternachricht an die auswertende Applikation schicken. Die auswertende Applikation sorgt einmalig für die Eintragung der Browser-Helper-DLL in die Windows-Registry, damit die DLL vom Internet Explorer eingebunden wird und ruft die Hook-Funktionen [Mic 2002a] zum Abfangen der Fensternachrichten auf. Die eingehenden Fensternachrichten werden aufbereitet und mit der aktuellen Zeit in einer Datei gespeichert, um später die Verweil- und Ladezeit der einzelnen Seiten bestimmen zu können. Zusätzlich werden die offenen Instanzen des Internet Explorers mit eindeutigen Nummern versehen. Die Ereignisse und Fensternachrichten werden mit den entsprechenden Instanz-Nummern gespeichert, um sie später den Browser-Fenstern zuordnen zu können.

Auf diese Weise erhält die auswertende Applikation folgende Ereignisse und Nachrichten: Das Browser-Helper-Object übermittelt Start und Ende der Internet-Explorer-Instanz, den Ladebeginn einer neuen Internet-Seite, die fertig geladenen Internet-Seite und zugehörige URL. Weniger wichtig sind die Änderungen des Textes in der Titel- und Statusleiste. Über die Hook-Funktionen werden die angeklickten Buttons, die ausgewählten Menüpunkte und die gedrückten Tasten ermittelt.

Wenn mehrere Instanzen des Internet Explorers laufen, müssen die Fensternachrichten der Hook-Funktionen denen des Browser-Helper-Objekts zugeordnet werden. Das lässt sich durch die Fensternummer des Internet-Explorer-Hauptfensters machen, die über die Browser-Helper-DLL abgefragt werden kann und die zum Herausfiltern der richtigen Fensternachrichten ohnehin gebraucht wird.

3.9.2 Applets und JavaScript

Die zweite Möglichkeit ist eine Kombination aus JavaScript, Applets und Applikation, die die Daten und Ereignisse entgegennimmt und speichert. Zusätzlich wird ein stark erweiterter Proxy-Server benötigt, der als Document-Editor die aufgerufenen HTML-Seiten um JavaScript-Aufrufe erweitert, und als Web-Server weitere Dateien bereitstellt.

Auf diese Weise können die Buttons des Browsers nicht direkt, sondern nur über Rückschlüsse erkannt werden, dafür funktioniert diese Art der Erfassung der Benutzeraktionen auf jedem Betriebssystem das Java unterstützt und über einen TCP/IP-Stack verfügt. Der Browser des Benutzers muss JavaScript und Applets kennen und sich an die Spezifikationen halten.

Die genauen Funktionsweisen der Applets und des JavaScript-Codes wurden bereits in den Abschnitten 3.7 und 3.8 erklärt, in diesem Abschnitt wird die Zusammenarbeit der einzelnen Komponenten beschrieben.

Dieses Verfahren besteht aus drei Teilen: Den ersten Teil stellt der erweiterte Proxy-Server dar, der die angeforderten Internet-Seiten modifiziert und weitere Dateien bereitstellt. Teil zwei umfasst die Funktionalität des JavaScripts und des Applets innerhalb des Browsers. Die auswertende Applikation ist der dritte Teil.

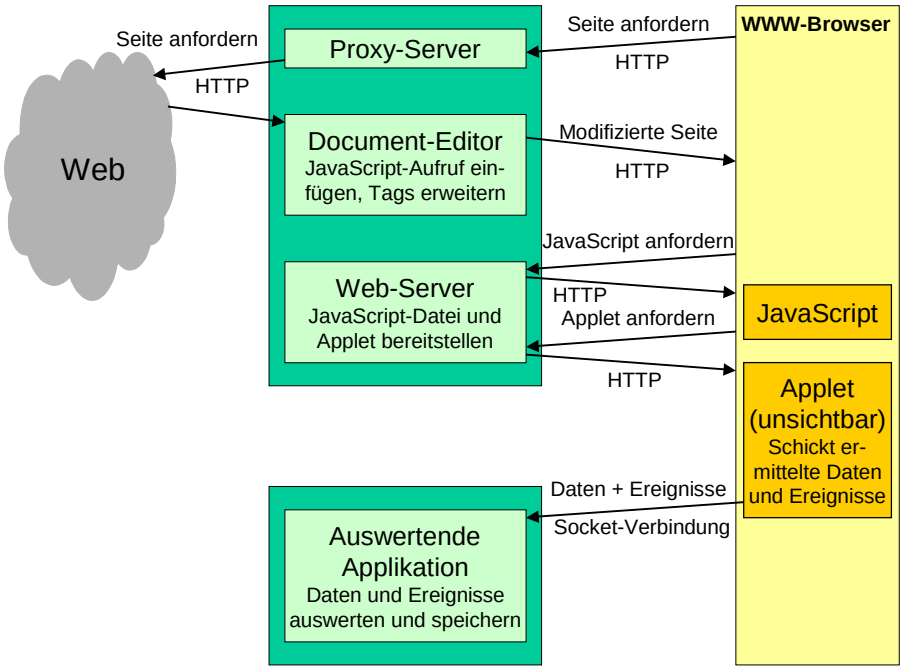

Abbildung 3.3: Erfassung der Benutzeraktionen mit JavaScript und Applet

Beim Aufruf einer Internet-Seite durch den Benutzer fordert der Browser diese Seite beim Proxy-Server an. Hier beginnt der erste Teil dieses Verfahrens seine Arbeit. Der integrierte Document-Editor fügt den Aufruf einer JavaScript-Datei in die HTML-Seite ein, die die benötigten Funktionen enthält. Des Weiteren werden alle Link- und Formulardefinitionen (Link- und Form-Tags) der HTML-Seite um Attribute erweitert, die beim Anklicken JavaScript-Funktionen aufrufen. Diese Funktionsaufrufe und weitere im Body-Tag sorgen dafür, dass die Ereignisse aufbereitet und an das Applet weitergeleitet werden. Die Funktionen zur Ereignisbehandlung befinden sich in einer externen JavaScript-Datei. Diese wird vom Browser beim Proxy-Server angefordert, sobald er den Aufruf der JavaScript-Datei in der HTML-Seite findet. Der Proxy-Server arbeitet dann als Document-Generator und Web-Server, der die JavaScript-Datei erstellt und dem Browser als Antwort auf seine Anforderung zuschickt.

Im zweiten Teil dieses Verfahrens stößt der Browser auf den JavaScript-Aufruf und fordert daraufhin die entsprechende Datei an. Neben den Ja-

vaScript-Funktionen zur Ereignisbehandlung enthält der JavaScript-Code auch den Aufruf eines Applets. Das Applet wird vom Proxy-Server geladen und baut nach dem Start eine TCP/IP-Verbindung zur auswertenden Applikation auf. Auf diesem Wege übermittelt es die ermittelten Daten der Internet-Seite und die eingetretenen Ereignisse. Mit Hilfe von speziellen HTML-Tags wird das Applet unsichtbar gestartet, um den Benutzer nicht beim Betrachten der Seite zu stören.

Die auswertende Applikation, der dritte Teil dieses Verfahrens, nimmt die Daten des Applets entgegen, wertet sie aus und speichert sie. Mit Hilfe von Daten über die Internet-Seite und einer selbst angelegten History lassen sich Rückschlüsse auf einige Buttons ziehen, die der Benutzer angeklickt hat.

Zwei Besonderheiten zeichnen diesen Lösungsansatz aus: Zum einen können auf diese Weise mehrere Benutzer gleichzeitig verwaltet werden. Mit Hilfe von gespeicherten Benutzer- und Fensternummer lassen sich nachträglich die Aktionen eines Benutzers an einem bestimmten Browser-Fenster nachvollziehen.

Zum anderen lässt sich dieses Verfahren mit wenigen Einstellungen verteilt betreiben, da sämtliche Kommunikation auf TCP/IP basiert. In diesem Fall läuft auf dem Rechner des Benutzers lediglich der Browser, während der Proxy-Server und die auswertende Applikation auf einem anderen Rechner ausgeführt werden. Das Applet darf aus Sicherheitsgründen nur zu dem Rechner eine TCP/IP-Verbindung aufbauen, von dem es geladen wurde. Deshalb müssen der Proxy-Server und die auswertende Applikation auf dem selben Rechner laufen.

Da die geklickten Buttons des Browsers auf diese Weise nicht abgefragt werden können, müssen sie durch Rückschlüsse erkannt werden. So wird z. B. das Anklicken der Zurück-Buttons dadurch erkannt, dass eine neue Internet-Seite im Browser angezeigt wird, die nicht neu geladen wurde, sondern aus dem Cache (Zwischenspeicher für Internet-Seiten und deren Dateien) des Browsers kommt. Auf ähnliche Weise lassen sich auch die Buttons „Vor“ und „Aktualisieren“ erkennen. Der Sprung über mehrere

Seiten vor und zurück und die Auswahl einer Seite aus den Lesezeichen (Bookmark) des Browsers lassen sich nur eingeschränkt erkennen.

3.10 Zusammenfassung

Resümierend bieten die genannten Verfahren aufgrund ihrer verschiedenen Ansätze unterschiedliche Stärken und Schwächen. Die folgende Tabelle 3.1 gibt einen Überblick über die wichtigsten „erkannten Ereignisse“. In diesem Abschnitt werden die Verfahren einander gegenüber gestellt und die geeignetste Lösung bestimmt, die in den nächsten Kapiteln näher beschrieben wird.

Auf den ersten Blick ist das Modifizieren eines Browsers, dessen Quellcode verfügbar ist, der beste Weg. Alle relevanten Daten lassen sich ermitteln und alle interessanten Ereignisse lassen sich direkt abfangen. Der einzige Schwachpunkt dieses Verfahrens ist die Tatsache, dass sich der Benutzer seinen Browser nicht aussuchen kann. Das kann dazu führen, dass er aufgrund einer leicht unterschiedlichen Benutzungsoberfläche bestimmte Funktionen nicht nutzt und dadurch mögliche Statistiken zum Benutzerverhalten verfälscht. Dieses Verfahren eignet sich ebenso wenig zur Entwicklung von Navigationswerkzeugen, da der Benutzer des resultierenden Werkzeugs auf diesen modifizierten Browser angewiesen wäre.

Den gleichen Nachteil hat das Verfahren, das das Browser-Helper-Object mit den abgefangenen Fensterereignissen kombiniert. Obwohl sich hierbei ebenfalls sehr viele Daten und Ereignisse sammeln lassen, hat der Benutzer keine Möglichkeit einen anderen Browser zu wählen. Es sprechen die gleichen Gründe gegen die Wahl dieser Lösung wie bei dem vorigen Verfahren.

Die Lösung mit dem Applet und dem JavaScript funktioniert mit jedem Browser, der sich an die Spezifikationen von JavaScript und Java Applets hält. Der Benutzer hätte die freie Wahl des Browsers, doch leider halten sich viele Browser nur teilweise an die Spezifikationen. Derartige Probleme werden im fünften Kapitel näher besprochen.

Tabelle 3.1: Möglichkeiten der einzelnen Verfahren

Möglichkeit / Verfahren	Erkennt Browser-Buttons Zurück und Vor	Erkennt Benutzer-ereignisse auf der HTML-Seite	Erkennt Laden neuer Seiten	Kann aktuelle URL auslesen	Erlaubt Aufruf neuer Seiten	Freie Wahl des Browsers
Spyglass SDI	nein	nein	nein	Ja	ja	einge-schränkt
Browser Helper Object[5]	nein	nein	ja	ja	ja	nein
Fenster-ereignisse (Window Events)	ja	teilweise	nein	nein	nein	einge-schränkt
Modifizierter Browser	ja	ja	ja	ja	ja	nein
Java Applets	nein	nein	ja	ja	ja	nein
JavaScript	nein	ja	ja	ja	ja	einge-schränkt
BHO und Fenster-ereignisse	ja	teilweise	ja	ja	ja	nein
JavaScript, Applet und Proxy	teilweise	ja	ja	ja	ja	einge-schränkt

[5] Das Browser-Helper-Object arbeitet nur mit Microsofts Internet Explorer ab Version 5.0

Die Kombination von Java Applets, JavaScript, Document-Editor und auswertender Applikation ist recht aufwendig und es werden nur die wichtigsten Benutzeraktionen erkannt, trotzdem ist dieser Ansatz nach der Meinung des Autors der universellste und wird deshalb in den folgenden Kapiteln ausgiebig behandelt.

4 Funktionsweise des Access-Trackings

Dieses Kapitel geht auf die Details und die Realisierung dieses Projektes ein, das im Folgenden „Access-Tracking“ genannt wird. Abschnitt 4.1 gibt einen Überblick über die Funktionsweise des Access-Tracking. Im Abschnitt 4.2 werden kurz das Framework Scone [Wei 2002] und seine Komponenten beschrieben, in das dieses Projekt eingebettet ist. Die Komponenten von Scone, die das Access-Tracking nutzt, werden besonders hervorgehoben. Scone soll den Benutzer bei der Entwicklung von Navigationswerkzeugen unterstützen und wird durch das Access-Tracking um die Erfassung der Benutzeraktionen und die Steuerung des Browsers ergänzt.

Danach werden die drei Teile des Access-Trackings und deren Interaktion näher beschrieben. Den ersten Teil bildet der erweiterte Proxy-Server (Abschnitt 4.3), der die vom Browser angeforderten Seiten um JavaScript-Code und Funktionsaufrufe ergänzt. Den zweiten Teil stellt der Browser des Benutzers dar (Abschnitt 4.4), der die vom Proxy-Server modifizierte Internet-Seite anzeigt. Innerhalb des Browsers ermittelt der JavaScript-Code die relevanten Daten, fängt die Ereignisse ab und schickt sie mit Hilfe eines Applets weiter. Abschnitt 4.5 beschreibt den dritten Teil der die auswertende Applikation umfasst. Diese wird innerhalb von Scone ausgeführt und rekonstruiert die Aktionen des Benutzers aus den Daten des Applets. Die Applikation stellt den Plugins von Scone die Ereignisse „Ladebeginn“, „Ladeende“ und „Verlassen der Internet-Seite“ zur Verfügung, so dass diese wissen, wann neue Daten des Applets eingegangen sind. Des Weiteren erlaubt sie den Plugins die Steuerung des Browsers.

Abschnitt 4.6 beschreibt die Ereignisse, die vom Access-Tracking direkt erfasst werden. Einige der Ereignisse, auf die nicht direkt zugegriffen werden kann, lassen sich über Rückschlüsse ermitteln. Solche Ereignisse werden im Abschnitt 4.7 erklärt.

Abschließend werden im Abschnitt 4.8 Frames [Mün 2001] eingeführt, und deren Besonderheiten in Bezug auf die Realisierung des Access-Tracking angerissen.

4.1 Übersicht über das Access-Tracking

Dieses Projekt wurde in Java realisiert und ergänzt das Framework Scone (Abschnitt 4.2) um die Erfassung der Benutzeraktionen an Web-Browsern.

Sobald der Benutzer über seinen Browser eine Internet-Seite aufruft, fordert der Browser diese Seite bei dem in Scone integrierten Proxy-Server an. Die Einstellung im Browser, Internet-Seiten über den Scone-Proxy-Server zu laden, muss einmalig vorgenommen werden. Dieser Proxy-Server enthält einen Document-Editor und -Generator, mit dem sich angeforderte Dokumente modifizieren und zusätzliche Dateien erstellen lassen. Die können dann, wie bei einem Web-Server, vom Browser angefordert werden.

Mit Hilfe des Document-Editors wird in die vom Browser angeforderte HTML-Datei der Aufruf einer JavaScript-Datei eingefügt und einige der HTML-Tags werden um Funktionsaufrufe erweitert. Diese Funktionsaufrufe sorgen im Link-, Form- und im Body-Tag dafür, dass beim Anklicken eines Verweises (Link), beim Abschicken eines Formulars (Form), beim Beenden des Ladevorgangs der Seite und beim Verlassen der Seite eine passende Funktion aus der JavaScript-Datei aufgerufen wird, die dieses Ereignis bearbeitet. Über weiteren JavaScript-Code werden die interessanten Daten der Internet-Seite ausgelesen, die für statistische Erhebungen oder für die Entwicklung von Navigationswerkzeugen relevant sind. Dazu gehören die aktuelle URL, die Lade- und Verweildauer, die Anzahl der Anzeigebereiche innerhalb der Internet-Seite (Frames) und deren Namen. Des Weiteren enthält der JavaScript-Code noch ein Applet, das in die Internet-Seite eingebettet wird, um die gesammelten Daten per TCP/IP-Verbindung an die auswertende Applikation zu schicken. Das Applet wird unsichtbar gestartet, um den Benutzer nicht beim Betrachten der Seite zu stören.

Klickt der Benutzer auf einen Verweis oder schickt er ein Formular ab, dann wird diese Information über die aufgerufenen JavaScript-Funktionen per LiveConnect [Net 1997a] an das Applet und von dort aus an die Applikation zur Auswertung geschickt. Die Applikation wertet die empfangenen Daten aus und legt sie in einer Datenbank ab, um sie später weiter zu verwenden.

Das User-Handling, ein Teilprojekt des Access-Trackings, weist jedem Benutzer eine eindeutige Benutzernummer zu, die mit den Informationen über seine Aktionen gespeichert wird. Dadurch ist es nachträglich möglich, die gespeicherten Benutzeraktionen den einzelnen Benutzern zuzuordnen. Damit der Benutzer nicht vor jeder Sitzung erneut seinen Namen beim Access-Tracking angeben muss, wird seine Benutzernummer als Cookie in seinem Browser gespeichert. Dieser Cookie wird bei jedem Seitenaufruf überprüft, um einen neuen Benutzer ohne gültigen Cookie sofort zu erkennen. In diesem Fall wird automatisch die Seite des User-Handlings aufgerufen, bei der sich der Benutzer registrieren oder anmelden kann.

Arbeitet der Benutzer mit mehreren Browser-Fenstern gleichzeitig, dann sollten die Aktionen später unterscheidbar sein. Zu diesem Zweck wird jedem Browser-Fenster ein eindeutiger Name zugewiesen. Dieser wird mit den anderen Informationen über die Benutzeraktion gespeichert.

4.2 Das Framework Scone

Scone [Wei 2002] ist ein Framework zum Entwickeln und Testen von Navigationswerkzeugen. Über einen Plugin-Mechanismus können Erweiterungen für den Browser und ähnliches in das Framework integriert oder selbst erstellt werden. Das Projekt „Access-Tracking“ realisiert eine der Komponenten, die dem Entwickler beim Erstellen von Navigationswerkzeugen helfen sollen. Andere Komponenten, wie der erweiterte Proxy-Server, die Datenspeicherung und die vorbereiteten Socket-Verbindungen werden in den nächsten Abschnitten näher beschrieben, da sie vom Access-Tracking benutzt werden. Außerdem stellt Scone einen Robot zum Sammeln von Informationen aus dem Internet und einen Document-

Parser zum Analysieren von Internet-Seiten zur Verfügung. Die sind für diese Arbeit aber nicht relevant und sollen daher hier nur erwähnt werden.

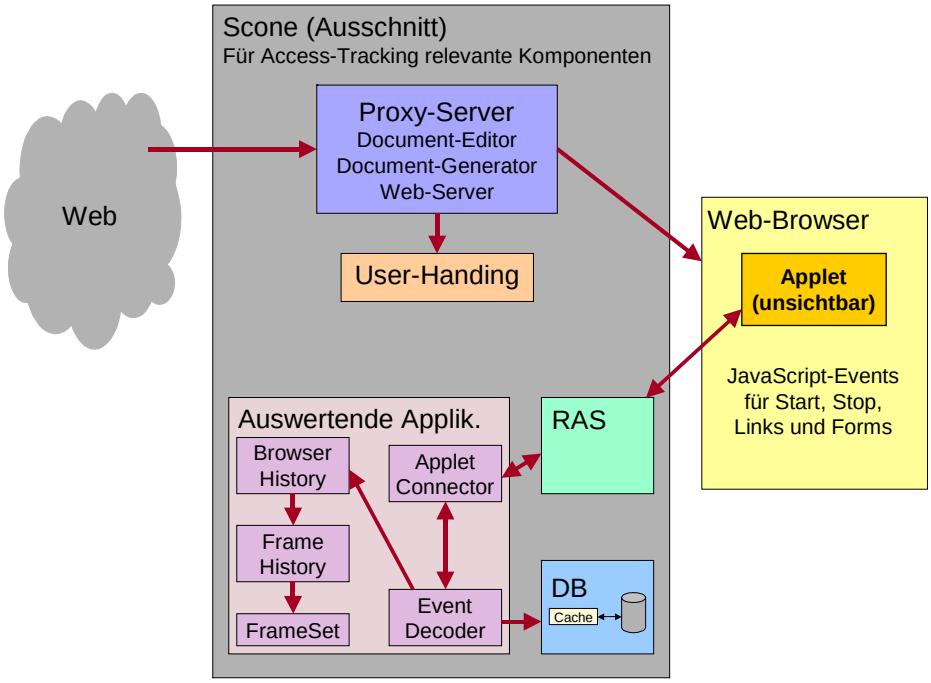

Abbildung 4.1: Für das Access-Tracking relevante Komponenten von Scone

4.2.1 Proxy-Server, Document-Editor, Document-Generator und Web-Server

Scone enthält eine stark erweiterte Version des Proxy-Servers WBI [IBM 1999] von IBM. Dieser Proxy-Server macht es möglich, angeforderte HTML-Dateien auf effiziente und komfortable Weise zu durchsuchen und auf der Ebene der HTML-Tags zu modifizieren (Document-Editor). So ist es z. B. mit geringem Aufwand möglich, auf das Auftreten bestimmter HTML-Tags zu reagieren und deren Attribute zu verändern. Des Weiteren bietet WBI die Möglichkeit, bestehende und bei Bedarf selbst generierte Dateien unter einer bestimmten URL anzubieten. In diesem Fall arbeitet der Proxy-Server dann als Web-Server und Document-Generator.

Die Art, in der die Dokumente von dem Proxy-Server modifiziert oder generiert werden sollen, wird in sogenannten MEGs (Modifier-Editor-Generator) in Form von Java-Dateien festgelegt. Diese MEGs werden innerhalb der Scone-Plugins eingebunden und jedes Mal gestartet, wenn eine HTML-Datei vom Browser angefordert wird. Die Datei wird den MEGs als Strom (Stream) aus HTML-Tags übergeben. Auf diese Weise lassen sich die einzelnen Tags in HTML-Dateien einfach bearbeiten und löschen. Auch neue Tags lassen sich in die HTML-Datei einfügen.

Soweit zur generellen Funktionsweise des Proxy-Servers, die speziellen Modifikationen für das Access-Tracking werden im Abschnitt 4.3 beschrieben.

4.2.2 Datenspeicherung

Scone stellt für alle wichtigen Daten persistente Objekte bereit, die über einen Zwischenspeicher-Mechanismus (Cache) sehr schnell zugreifbar sind. Diese Objekte enthalten unter anderem Daten der einzelnen Benutzer, Daten über die besuchten Internet-Seiten, die einzelnen Links einer Seite und die Benutzeraktion, die zum Seitenwechsel führte.

Der Entwickler kann in seinem Plugin angeben, welche derartigen Informationen er benötigt. Die entsprechenden Objekte werden dann von Scone erstellt, ohne dass er sich weiter darum kümmern muss. Beispiele für derartige Objekte sind das Link- und das NetNode-Objekt, die auch vom Access-Tracking benutzt werden. Scone erzeugt für jede aufgerufene Seite ein NetNode-Objekt, in dem die URL, der Zeitpunkt des ersten und des letzten Besuchs, die Anzahl der bisherigen Besuche und einige andere Daten gespeichert werden. Für jeden Verweis in der aufgerufenen Seite wird ein Link-Objekt erzeugt, in dem abgelegt wird, von welchem NetNode zu welchem NetNode verwiesen wird. Des Weiteren wird noch gespeichert, ob die Ziel-URL besondere Anhängsel hat, die durch das Fragezeichen oder das Doppelkreuz (Sharp) von der eigentlichen URL getrennt sind. Jedes NetNode- und Link-Objekt erhält eine eindeutige Kennnummer, Node-Id oder Link-Id genannt, über die das entsprechende Objekt angefordert werden kann.

Diese beiden Objekte werden vom Access-Tracking genutzt, um z. B. bei einem angeklickten Link auf einfache Weise alle den Link betreffenden Informationen mit nur einer Link-Id zu übergeben. Es gibt noch viele weitere Objekte, die von Scone aber nur dann erzeugt werden, wenn mindestens eines der installierten Plugins sie anfordert.

Zum Halten der persistenten Daten dient die Datenbank MySQL [Mys 2002]. Scone greift per TCP/IP auf die Datenbank zu, dadurch ist es möglich, Scone und die Datenbank auf verschiedenen Rechnern verteilt laufen zu lassen.

4.2.3 Vorbereitete Socket-Verbindungen

Scone bietet ein Verbindungspaket RAS (Remote Access Server) an, das dem Entwickler erlaubt, auf einfache Weise TCP/IP-Verbindungen aufzubauen oder entgegenzunehmen. Auf diese Weise lassen sich z. B. Daten mit Applets in Internet-Seiten austauschen oder Programme ansprechen, die nicht in Java geschrieben sind.

Das Access-Tracking benutzt dieses Paket, um die gesammelten Daten und Ereignisse von dem Applet in der Internet-Seite zu empfangen. Das Applet gibt beim Aufbau der Verbindung den Namen der Klasse an, für die diese Daten bestimmt sind. Der RAS erzeugt die entsprechende Instanz und übergibt ihr die Verbindung.

4.3 Modifikation der HTML-Dokumente im Proxy-Server

In diesem Abschnitt beginnt die Beschreibung der Access-Tracking-Realisierung mit der ersten Teilkomponente, dem Proxy-Server. Der zweite Teil umfasst die Funktionalität des JavaScripts und des Applets in der Internet-Seite, während der dritte Teil die auswertende Applikation behandelt. Der zweite und dritte Teil werden in den nächsten beiden Abschnitten beschrieben. Abbildung 3.3 zeigt alle drei Teilkomponenten sowie deren Kommunikation untereinander.

Der um einen Document-Editor und -Generator erweiterte Proxy-Server wurde bereits im Abschnitt 4.2.1 als Scone-Komponente vorgestellt. Hier werden nun die Modifikationen beschrieben, die mit Hilfe dieses Proxy-Servers an den vom Benutzer aufgerufenen Seiten vorgenommen werden.

Jedes Mal, wenn der Benutzer eine Internet-Seite aufruft, fordert der Browser diese Seite beim Proxy-Server an. Voraussetzung dafür ist, dass der Netzwerkname des Scone-Proxy-Servers vorher einmalig beim Browser als Proxy-Server eingetragen wurde.

Der Proxy-Server testet zunächst anhand der URL, ob die angeforderte Seite zu denen gehört, die er als Web-Server bereitstellen soll. In diesem Fall führt er die entsprechende Java-Datei aus, die den Inhalt generiert. Dann schickt er die neue Seite an den Browser, der sie angefordert hat.

Soll die Internet-Seite nicht vom Generator erstellt werden, wird sie aus dem Internet geladen und vom Editor auf der Ebene der HTML-Tags modifiziert.

4.3.1 Aufruf der JavaScript-Datei

Wird über den Browser eine Seite aus dem Internet angefordert, dann fügt der Document-Editor zunächst den Aufruf einer externen JavaScript-Datei in die geladene HTML-Datei ein. In der JavaScript-Datei befinden sich sowohl Funktionen zur Bearbeitung der Ereignisse, als auch Befehle zum Auslesen relevanter Daten aus dem Browser. Zusätzlich enthält der JavaScript-Code noch ein Applet, das bei der Ausführung dynamisch in die Internet-Seite eingefügt wird und für die Verbindung zur auswertenden Applikation sorgt.

Der Aufruf der JavaScript-Datei soll möglichst am Anfang der Internet-Seite eingefügt werden, damit die Datei vor eventuell bestehendem JavaScript-Code der geladenen Internet-Seite ausgeführt wird. Je früher der Aufruf erfolgt, desto eher wird das Applet erzeugt und damit die auswertende Applikation über das Laden der neuen Seite informiert. Durch die auswertende Applikation werden auch andere Plugins von Scone über

das Laden einer neuen Seite informiert. Je früher das Applet initialisiert wird, desto eher können andere Plugins darauf reagieren.

Der Aufruf der JavaScript-Datei darf allerdings nicht direkt am Anfang der Internet-Seite stehen, da direkt nach diesem Aufruf das Applet dynamisch in die HTML-Datei eingefügt wird. Das Applet am Anfang der Internet-Seite ist für den Browser ein Problem, wenn auf der selben Seite eine Frameset-Definition (Definition mehrerer Anzeigebereiche innerhalb einer Internet-Seite) folgt, da diese dann wegen des Applets ignoriert wird. Der Benutzer bekommt in diesem Fall eine leere Seite zu sehen. Um dieses zu umgehen, darf der Aufruf erst direkt vor dem Body-Tag erfolgen, da danach keine Frameset-Tags mehr akzeptiert werden. HTML-Dateien, in denen Anzeigebereiche definiert werden, erhalten keinen Aufruf der externen JavaScript-Datei. Sie erhalten somit auch kein Applet. Da diese Definitionsdateien keine Bedienelemente, wie Links oder Formulare, enthalten, sind der JavaScript-Code und das Applet hier unwichtig. Sie dienen lediglich der Definition der Anordnung der einzelnen Bereiche und der Festlegung deren Dateinamen.

Der JavaScript-Aufruf, der vor dem Body-Tag eingefügt wird, enthält einen Verweis auf die externe JavaScript-Datei. Der Name des virtuellen Netzwerkrechners, von dem diese Datei geladen werden soll, lautet „`_tracking.scone.de`“. Der Proxy-Server bearbeitet die Anfragen für diese Adresse und generiert die JavaScript-Datei sobald sie angefordert wird. Durch den Unterstrich vor dem Rechnernamen ist die Adresse im Internet ungültig. Auf diese Weise ist sichergestellt, dass durch den Proxy-Server keine im Internet existierenden Adressen verdeckt werden.

Für das Berechnen der Lade- und Verweildauer ist ebenfalls JavaScript-Code zuständig. Der Zeitpunkt des Ladebeginns sowie der Zeitpunkt, zu dem die Seite fertig geladenen ist, werden gespeichert. Die Differenz dieser beiden Zeitpunkte ergibt die zum Laden benötigte Zeit. Um die Zeitmessung nicht erst kurz vor dem Body-Tag zu starten, wird ein kleiner Teil des JavaScript-Codes, der den Befehl zur Speicherung des ersten Zeitpunktes enthält, direkt am Anfang der HTML-Datei eingefügt.

4.3.2 Funktionsaufrufe bei bestimmten Ereignissen

Nachdem der Aufruf für die JavaScript-Datei eingefügt ist, müssen nun die HTML-Tags, die besondere Ereignisse auslösen, mit Funktionsaufrufen versehen werden. Wie bereits in Abschnitt 4.2.1 beschrieben, ist es mit Hilfe des Document-Editors des Proxy-Servers möglich, die einzelnen HTML-Tags zu modifizieren.

Die HTML-Tags für Links und Formulare werden um Attribute erweitert, die beim Anklicken eine bestimmte Funktion des JavaScript-Codes zur Ereignisbehandlung anstoßen und weitere Informationen über das Tag übergeben. Link-Tags werden um das „**`onClick`**“-Attribut erweitert. Sie übergeben der aufgerufenen Funktion die Link-Id. Dabei handelt es sich um einen Textschlüssel, über den das Link-Objekt angefordert werden kann. Das Link-Objekt wurde von Scone angelegt und enthält alle Informationen über den Link. Form-Tags erhalten das „**`onSubmit`**“-Attribut. Sie rufen dadurch beim Abschicken eines Formulars die entsprechende JavaScript-Funktion auf und übergeben die Zieladresse des Formulars aus dem „**`action`**“-Attribut als Parameter. Die Link-Identifikation und die Adresse, an die die Daten des Formulars geschickt werden, werden im Falle eines Ereignisses über das Applet an die auswertende Applikation geschickt. Dort werden sie zusammen mit der Benutzeraktion in der Datenbank gespeichert.

Im Body-Tag der HTML-Seite werden „**`onLoad`**“- und „**`onUnload`**“-Attribute eingefügt, die nach dem Laden und vor dem Verlassen der Internet-Seite entsprechende JavaScript-Funktionen aufrufen um diese Ereignisse zu behandeln. Die JavaScript-Funktionen, die derartige Ereignisse bearbeiten, werden im Abschnitt 4.3.3 näher beschrieben.

Wenn in der angeforderten HTML-Datei bereits Attribute für die Ereignisbehandlung vorhanden sind, werden diese an die oben erwähnten Attribute angehängt, um die ursprüngliche Funktion der Seite nicht zu ändern.

Alle Variablen- und Funktionsnamen der JavaScript-Datei beinhalten das Wort „Scone“, um Namensgleichheiten zwischen diesem JavaScript-Code und dem der HTML-Seite zu vermeiden.

4.3.3 Die JavaScript-Datei

Sobald der Browser in der HTML-Datei auf den Aufruf der JavaScript-Datei stößt, fordert er diese unter der Adresse „**`_tracking.scone.-de`**“ an. Der Proxy-Server bearbeitet die Anfragen für diese Adresse und generiert die JavaScript-Datei für diesen Aufruf.

Zunächst wird getestet, ob der Browser einen gültigen Cookie übergeben hat. Der Cookie, eine Textdatei beschränkter Länge, die vom Browser auf der Festplatte abgelegt wird, enthält eine Nummer, die den Benutzer eindeutig identifiziert. Diese Benutzernummer wird mit Hilfe der Datenbank auf Gültigkeit überprüft und in den JavaScript-Code eingebaut, um sie bei jedem Ereignis im Datentransfer zur auswertenden Applikation angeben zu können. Falls kein gültiger Cookie übertragen wurde, hat sich der Benutzer an diesem Browser noch nicht angemeldet. In diesem Fall wird eine spezielle JavaScript-Datei generiert, die den Browser sofort zum User-Handling – der Benutzerverwaltung von Scone – umleitet, so dass sich der Benutzer anmelden kann. Nach erfolgter Anmeldung wird die Benutzernummer als Cookie im Browser des Benutzers gespeichert und beim nächsten Seitenaufruf erkannt. Die Seiten des User-Handlings werden ebenfalls vom Proxy-Server generiert.

Neben der Benutzernummer wird noch eine eindeutige Seiten- und Zugriffs-Identifikationsnummer in den JavaScript-Code integriert. Es handelt sich hierbei um die Anzahl der Millisekunden seit dem 1. Januar 1970. Für aufgerufene Internet-Seiten ist diese Nummer annähernd eindeutig. Sie dient der Erkennung von Seiten, die nach dem Anklicken des Vor- oder Zurück-Buttons aus dem Cache des Browsers geladen werden. Wie derartige Seiten genau erkannt werden, wird in dem Abschnitt 4.7.1 behandelt.

Wie in dem Abschnitt 4.2.2 bereits beschrieben, bietet Scone die Möglichkeit, beim Laden einer Seite viele Informationen zu sammeln und sie in Form von Objekten bereitzustellen. Ein Beispiel dafür sind die NetNode-Objekte, die Daten über den Aufruf einer Internet-Seite enthalten. NetNode-Objekte speichern den Zeitpunkt des ersten und des letzten Zugriffs, die Anzahl der bisherigen Zugriffe, die URL und weitere Informationen. Durch die Node-Id, eine 16-stellige hexadezimale Zahl, die als Hashcode der URL berechnet wurde, kann ein solches Objekt eindeutig identifiziert werden. Die Node-Id für die aktuelle Internet-Seite wird in den JavaScript-Code eingebaut und später mit den Ereignissen an die auswertende Applikation übergeben, um dort eindeutig auf dieses NetNode-Objekt zugreifen zu können.

Nach diesem Teil der JavaScript-Datei, der bei jedem Aufruf neu generiert wird, folgt nun ein statischer Teil, der jedes Mal gleich ist. Diese JavaScript-Befehle lesen relevante Daten aus dem Browser aus, die später in der Datenbank gespeichert werden sollen. Zunächst wird die aktuelle URL auf Anhängsel überprüft, die mit dem Fragezeichen oder dem Doppelkreuz (Sharp) von der eigentlichen URL getrennt werden. Die URL selbst kann über die Node-Id aus dem NetNode-Objekt ermittelt werden.

Um die einzelnen Browser-Fenster bei mehreren Browser-Instanzen unterscheiden zu können, wird der Fenstername herangezogen. Die Besonderheit des Fensternamens ist, dass sich dieser durch das Aufrufen weiterer Internet-Seiten nicht ändert. Falls der Fenstername noch nicht gesetzt ist, wird durch den JavaScript-Code ein annähernd eindeutiger Name erzeugt. Dieser besteht aus der Zeichenkette „SCONE“, an die die Anzahl der Millisekunden seit dem 1.1.1970 und eine vierstellige Zufallszahl angehängt ist. Auf diese Weise ist es nahezu unmöglich, dass es auf einem Rechner zwei oder mehr Fenster mit der gleichen Bezeichnung gibt.

Bei mehreren Anzeigebereichen (Frames) [Mün 2001] innerhalb der Internet-Seite werden die einzelnen Bereiche mit Hilfe der Bereichsdefinition mit eindeutigen Fensternamen versehen. In diesem Fall ist es nötig, zusätzlich zum Namen des Bereichsfensters auch den Namen des Fens-

ters mit der Bereichsdefinition zu ermitteln, das den von Scone vergebenen Namen behalten hat. Durch die Verbindung des Namens des Bereichsdefinitionsfensters und denen der einzelnen Bereiche lassen sich diese eindeutig unterscheiden und nachträglich die zusammengehörenden Bereiche erkennen.

Des Weiteren ist noch der Name des Browsers interessant, um das unterschiedliche Verhalten verschiedener Browser zu kompensieren. Dies ist z. B. nötig um das Applet vor dem Benutzer zu verstecken. Die beiden meistbenutzten Browser [Int 2001], Internet Explorer und Netscape Navigator, erwarten unterschiedliche HTML-Tags, um ein Applet unsichtbar zu starten.

Für jedes Ereignis wie „das Beenden des Ladevorgangs der Internet-Seite“, „das Verlassen der Seite“, angeklickte Links oder angeklickte Formulare stellt die JavaScript-Datei eine spezielle Funktion bereit, die dieses Ereignis behandelt. Die Funktionen für die Link- und Form-Ereignisse nehmen als Parameter noch die Link-Id oder den Wert des „**`action`**“-Attributs auf. Der Wert des „**`action`**“-Attributs besteht meist aus dem Rechnernamen oder der E-Mail-Adresse, an die der Inhalt des Formulars übertragen werden soll. Der Name des Ereignisses wird zusammen mit dem eventuellen Parameter als Zeichenkette an das Applet übergeben. Das Applet stellt eine Methode bereit, die aus dem JavaScript-Code heraus aufgerufen und der eine Zeichenkette übergeben werden kann. Über diesen Mechanismus, genannt LiveConnect [Net 1997a], schickt der JavaScript-Code die abgefangenen Ereignisse an das Applet, das den Zeitpunkt des Ereignisses an die Zeichenkette anhängt und diese dann an die auswertende Applikation schickt.

Das Applet wird von dem JavaScript-Code bei seiner Ausführung in die HTML-Seite eingefügt. Dies geschieht mit Hilfe von dynamischem HTML. Während der Generierung der JavaScript-Datei durch den Proxy-Server wird der Applet-Tag und die zugehörigen Applet-Parameter mit den nötigen Daten erzeugt. In den Attributen des Applet-Tags muss stehen, wie das Applet heißt und von welchem Netzwerkrechner es geladen

werden soll. Diese Informationen erhält der Proxy-Server aus den Scone-Konfigurationsdateien, die bei der Einrichtung von Scone angelegt wurden. Das Applet selbst ist fertig kompiliert und wird von dem Proxy-Server bereitgestellt, der in diesem Fall als Web-Server arbeitet. Aufgrund des Sicherheitskonzeptes von Applets ist es nötig, dass das Applet von dem Rechner geladen wird, zu dem es später die Socket-Verbindung aufbauen will. Andernfalls lässt sich die Verbindung nicht aufbauen.

Für die Fehlersuche lässt sich das Applet auf der Internet-Seite sichtbar darstellen. Ansonsten ist es versteckt, um den Benutzer nicht beim Arbeiten mit dem Browser zu stören. Anhand von Scones Konfigurationsdateien erkennt der Proxy-Server, ob das Applet versteckt werden soll. Dann fügt er die passenden HTML-Tags vor dem Applet-Tag ein.

Damit das Applet nach der Initialisierung eine Socket-Verbindung zur auswertenden Applikation aufbauen kann, muss es wissen, auf welchem Netzwerkrechner Scone läuft und auf welchem Port die Verbindung erwartet wird. Diese Informationen werden dem Applet als Applet-Parameter übergeben.

Alle bereits ermittelten Daten über die aktuelle Internet-Seite und den Benutzer werden dem Applet ebenfalls als Parameter übermittelt. So kann es all diese Daten gleich nach seiner Initialisierung an die auswertende Applikation schicken.

4.4 Die Internet-Seite im Browser

Sobald der Benutzer eine Seite aufruft, wird sie durch den Browser beim Proxy-Server angefordert. Dieser schickt sie, wie im Abschnitt 4.3 beschrieben, modifiziert zurück.

Der Browser arbeitet nun nacheinander die einzelnen Tags der HTML-Datei ab. Am Anfang der Datei trifft er auf einen kurzen JavaScript-Teil, in dem der aktuelle Zeitpunkt gespeichert wird. Dieser Zeitpunkt wird später für die Berechnung der Lade- und Verweilzeit benötigt.

Zwischen den Kopfdaten der HTML-Datei und dem Bereich, in dem die sichtbaren Bestandteile definiert werden (Body-Bereich), steht der Aufruf der externen JavaScript-Datei. Handelt es sich bei der aufgerufenen HTML-Seite um die Definition von Anzeigebereichen (Frames), wird keine JavaScript-Datei eingebunden, da keine Ereignisse in dieser Seite abfangbar sind.

4.4.1 Ausführen der externen JavaScript-Datei

Nach Erreichen des JavaScript-Aufrufes wird die entsprechende Datei beim Proxy-Server angefordert und ausgeführt. Bis zu diesem Zeitpunkt werden keine weiteren Elemente der HTML-Datei bearbeitet. Zunächst werden die relevanten Daten über die Internet-Seite aus dem Browser ausgelesen. Dazu gehören eventuelle Anhängsel der aktuellen URL, die durch Fragezeichen oder Doppelkreuz von der eigentlichen URL getrennt sind. Des Weiteren wird der Name des Fensters ermittelt, um mehrere offene Browserfenster unterscheiden zu können. Außerdem wird die Anzahl der gezeigten Anzeigebereiche (Frames) und gegebenenfalls der Name des Fensters, das die Anzeigebereiche definiert hat (Top Frame) abgefragt. Der Name des Browsers wird benötigt, um Unterschiede der Browser in der Interpretation des HTML- und JavaScript-Codes zu kompensieren. Die Benutzernummer, die Node-Id und die eindeutige Identifikationsnummer wurden als Konstanten in den JavaScript-Code eingebaut, als dieser vom Proxy-Server generiert wurde.

Sobald alle relevanten Daten ermittelt sind, wird mit Hilfe von dynamischem HTML das Applet-Tag in die HTML-Seite eingefügt. Dies geschieht genau an der Stelle, an der der JavaScript-Aufruf steht. Für den Applet-Tag wichtige Daten, wie die Aufruf-Informationen und die Verbindungsparameter für die Socket-Verbindung zur auswertende Applikation, hat der Proxy-Server bereits während der Generierung in den JavaScript-Code eingebaut. Alle bereits bekannten Fakten über die aktuelle Internet-Seite, die der JavaScript-Code bereits ermittelt hat, werden dem Applet mit Hilfe der Applet-Parameter übergeben. Diese Parameter, be-

stehend aus je einem Parameternamen und einem Wert in Form einer Zeichenkette, werden zur Laufzeit des Applets ausgelesen.

Während der Browser das Applet über den Proxy-Server lädt und dann initialisiert, wird die HTML-Datei weiter bearbeitet und bei Bedarf weitere Dateien wie z. B. Grafiken geladen und angezeigt.

4.4.2 Starten des Applets

Sobald das Applet komplett geladen ist, wird es innerhalb des Browsers ausgeführt. Mit Hilfe des Div-Tags und einer entsprechenden Cascading-Stylesheets-Definition wird das Applet „versteckt“. Auf diese Weise wird das Layout der Internet-Seite nicht verändert und der Benutzer nicht beim Betrachten gestört.

Das Applet fragt zunächst die Parameter aus dem Applet-Tag ab, wobei es neben den Daten über den Benutzer und denen über die aktuelle Internet-Seite auch erfährt, wohin es die Socket-Verbindung aufbauen soll. Nach erfolgreichem Aufbau der Verbindung schickt das Applet alle bisher gesammelten Informationen an die auswertende Applikation. Dazu gehören die Kennnummer des Benutzers, die eindeutige Identifikationsnummer, die Anhängsel der aktuellen URL, der Name des aktuellen Browser-Fensters, der Name des obersten Browser-Fensters, die Anzahl der Anzeigebereiche, die Startzeit aus dem ersten JavaScript-Teil, der Name des Browsers und die Node-Id. Über die Node-Id kann die auswertende Applikation auf das entsprechende NetNode-Objekt zugreifen, das Scone beim ersten Aufruf dieser Internet-Seite angelegt hat.

4.4.3 Behandeln von Ereignissen

Wie im Abschnitt 4.3.2 bereits beschrieben, wurden der Body-Tag und die HTML-Tags für Verweise und Formulare durch den Document-Editor mit Attributen versehen, so dass diese im Fall eines Ereignisses bestimmte JavaScript-Funktionen aufrufen.

Die JavaScript-Funktionen, die die Ereignisse „Anklicken eines Links“ und „Abschicken eines Formulars“ bearbeiten, erzeugen eine Zeichenket-

te, die den Namen des Ereignisses und zusätzliche Daten enthält. Je nach Ereignis, handelt es sich bei diesen zusätzlichen Daten entweder um die Adresse, an die das Formular geschickt werden soll oder um die Link-Id, eine eindeutige Bezeichnung für das Link-Objekt, das alle Informationen über den Verweis enthält. Dieses Objekt ist eines der Objekte, die Scone automatisch anlegt, sobald die angeforderte Seite den Proxy-Server passiert.

Für die Ereignisse „Verlassen der Internet-Seite“ und „fertig geladene Seite“ existieren ebenfalls Funktionen, die lediglich den Namen des Ereignisses in die Zeichenkette schreiben. Hier sind keine weiteren Informationen nötig.

Alle Funktionen zur Ereignisbehandlung rufen eine Methode des Applets auf, und übergeben ihm so ihre Zeichenkette. Das Applet erweitert die Zeichenkette um den Zeitpunkt des Ereignisses, der z. B. beim Verlassen der Internet-Seite für das Berechnen der Verweilzeit benötigt wird.

Diese Zeichenkette wird dann über die TCP/IP-Verbindung an die auswertende Applikation geschickt, um dort das Ereignis in der Datenbank zu vermerken.

4.5 Die auswertende Applikation

Die Auswertende Applikation bildet den dritten und zugleich aufwendigsten Teil des Access-Tracking. Im Abschnitt 4.5.1 wird eine Übersicht über die einzelnen Klassen und deren Kommunikation untereinander gegeben. Danach wird das **`Access`**-Objekt (Abschnitt 4.5.2) eingeführt, in dem die meisten der gesammelten Daten über den Zugriff gespeichert werden. Die folgenden Abschnitte beschreiben dann die Klassen der auswertenden Applikation im Einzelnen.

4.5.1 Übersicht über die auswertende Applikation

Abbildung 4.2 zeigt eine grafische Übersicht der auswertenden Applikation und die Verknüpfung mit den Scone-Komponenten.

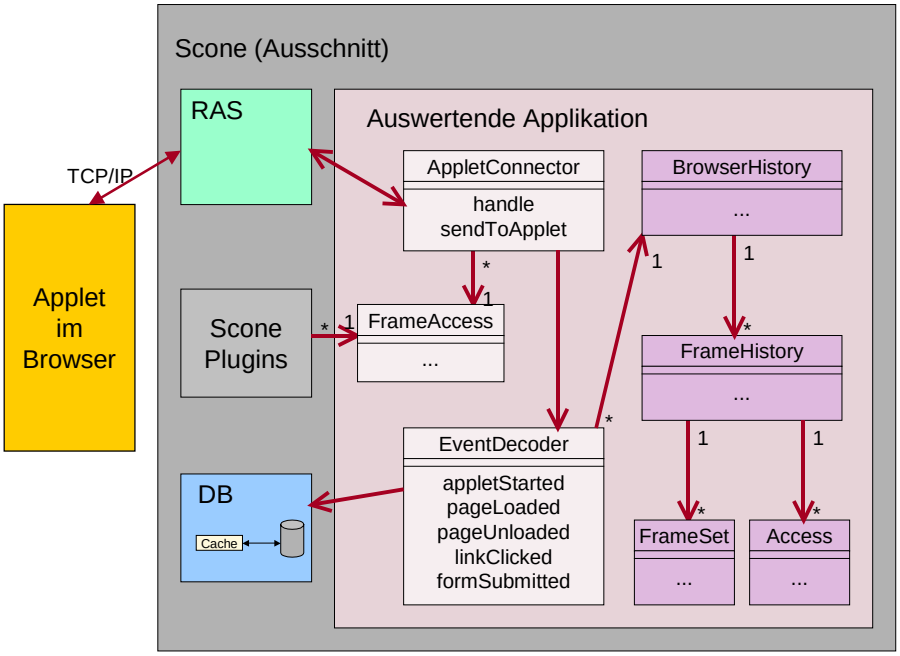

Abbildung 4.2: Übersicht über die auswertende Applikation

Auf der linken Seite der Abbildung 4.2 ist das Applet angedeutet, das im vorigen Abschnitt besprochen wurde. Der **`RAS`** (Remote-Access-Server) nimmt die TCP/IP-Verbindung des Applets entgegen und erzeugt eine Instanz des **`AppletConnectors`** (Abschnitt 4.5.3), der die einzelnen Daten aus den Zeichenketten des Applets rekonstruiert. Der **`AppletConnector`** ruft in der Klasse **`EventDecoder`** die entsprechende Methode auf, die dieses Ereignis bearbeitet. Des Weiteren bietet der **`AppletConnector`** den Plugins von Scone die Möglichkeit, Befehle an das Applet zu schicken (Abschnitt 4.5.5).

Der **`EventDecoder`** (Abschnitt 4.5.4) legt für jede geladene Seite ein **`Access`**-Objekt an, das in der Datenbank gespeichert wird. Hier werden alle Daten über die Internet-Seite und die Benutzerereignisse abgelegt.

Mit Hilfe der **`BrowserHistory`** (Abschnitt 4.5.6) werden die **`Access`**-Objekte abgelegt. Das ist nötig, um Internet-Seiten aus dem Cache (Zwischenspeicher) des Browsers anhand der eindeutigen Identifikations-

nummer zu erkennen. Dieser Mechanismus wird im Abschnitt 4.7.1 näher beschrieben.

Für jedes offene Browser-Fenster erzeugt die **`BrowserHistory`** eine Instanz der **`FrameHistory`**, in der die **`Access`**-Objekte dieses Fensters abgelegt werden. Hier wird praktisch der Verlauf der aufgerufenen Seiten gespeichert. Im Fall von Internet-Seiten mit mehreren Anzeigebereichen (Frames) wird in der **`FrameHistory`** ein FrameSet-Objekt abgelegt, das eine Liste der **`Access`**-Objekte enthält, die den einzelnen Bereichen entsprechen.

4.5.2 Das Access-Objekt

Das **`Access`**-Objekt dient der Speicherung der relevanten Daten für jeden Zugriff. Jedes Mal wenn der Benutzer eine Internet-Seite aufruft, wird ein entsprechendes **`Access`**-Objekt erzeugt. Alle Werte, die für eine Seite konstant sind, wie die Benutzernummer, Node-Id, Startzeit, aktueller und oberster Fenstername, Browsername, die vorige URL, die URL des obersten Fensters und die Anhängsel der aktuellen URL werden bereits bei der ersten Datenübertragung vom Applet mitgesandt und in das **`Access`**-Objekt eingetragen. Die Benutzeraktion, die zum Laden dieser Seite führte, wird mit Hilfe von Daten der zuletzt besuchten Seite ermittelt. Wie das genau geschieht, wird im Abschnitt 4.5.4 behandelt. Die Ladezeit wird gespeichert, sobald das Applet von der fertig geladenen Seite berichtet. Die Verweildauer wird beim Verlassen der Seite berechnet. Die Link-Id wird gespeichert, sobald der Benutzer einen Verweis angeklickt und das Applet die Link-Id übermittelt hat.

Im Fall von mehreren Anzeigebereichen (Frames) müssen mehrere **`Access`**-Objekte – für jeden Bereich eines – in der Datenbank gespeichert werden. Um später nachvollziehen zu können, welche **`Access`**-Objekte zusammengehören, wird ein oberstes **`Access`**-Objekt erzeugt, auf das alle Bereichs-**`Access`**-Objekte verweisen. Für diese Verweise stellt jedes **`Access`**-Objekt entsprechende Felder bereit. Eines für die eigene, automatisch erzeugte Access-Id und eines für die Access-Id des obersten

Access-Objektes. Das Feld für das oberste Objekt erhält nur dann einen von Null unterschiedlichen Wert, wenn dieses Objekt zu einem Anzeigebereich gehört.

4.5.3 Der AppletConnector

Die kommende Socket-Verbindung des Applets wird in Scone von der Komponente Remote-Access-Server (**RAS**) entgegengenommen. Das Applet übermittelt als erstes Informationspaket den Namen der Klasse, für die die Daten bestimmt sind. Der **RAS** erzeugt eine Instanz der Klasse und übergibt ihr die Verbindung.

Die Klasse, die für die Verbindung mit dem Applet zuständig ist, heißt **AppletConnector**. Der **AppletConnector** nimmt die Zeichenketten des Applets entgegen und extrahiert die einzelnen Informationen. Dann ruft er die entsprechende Methode der Klasse **EventDecoder** auf, in der die Daten ausgewertet und in die Datenbank eingetragen werden.

Der **AppletConnector** bietet anderen Plugins von Scone die Möglichkeit, Befehle über die Socket-Verbindung an das Applet zu schicken und dort auszuführen (Abschnitt 4.5.5). Dazu gehört das „Aufrufen neuer Internet-Seiten“, „öffnen neuer Browser-Instanzen“ und „ausführen beliebiger JavaScript-Befehle“. Auf diese Weise können andere Plugins dem Benutzer unter anderem anbieten, bestimmte Internet-Seiten aufzurufen, ohne dass sich der Benutzer langwierig dorthin klicken muss.

Der **AppletConnector** wurde so entwickelt, dass er bei Bedarf durch eine andere Klasse ersetzt werden kann. Dies wäre z. B. der Fall, wenn es einen anderen Weg gäbe, die Daten des Browsers in ähnlichem Umfang auszulesen und ihn zu steuern. Die neue Klasse könnte dann auf ihre Art die Daten vom Browser entgegennehmen und die Methoden des **EventDecoders** aufrufen.

4.5.4 Der EventDecoder

Der **`EventDecoder`** stellt für jedes Ereignis, das das Applet abfängt, eine eigene Methode bereit und legt die empfangenen Daten in dem **`Access`**-Objekt ab.

Das erste Ereignis, das das Applet übermittelt, ist der Start des Applets. Hierbei werden alle bisher ermittelten Daten, die für die Internet-Seite konstant sind, übertragen. Zu diesen Daten gehören die Benutzernummer, die Node-Id, der Name des aktuellen und möglicherweise der des obersten Fensters, die Startzeit der Seite und die vorige URL. Zusätzlich werden noch besondere Teile der aktuellen URL übermittelt, die mit einem Fragezeichen oder einem Doppelkreuz (#) an die eigentliche URL angehängt sind. Die aktuelle URL selbst kann über die Node-Id abgefragt werden. All diese Daten werden direkt in dem neu erzeugten Access-Objekt abgelegt. Zu der URL des obersten Fensters wird das entsprechende NetNode-Objekt gesucht, und die resultierende Node-Id im **`Access`**-Objekt gespeichert.

Die Anzahl der Anzeigebereiche in der Internet-Seite wird ebenfalls übertragen. Bei mehr als einem Bereich wird zusätzlich zu den **`Access`**-Objekten für jeden Bereich ein weiteres erzeugt, auf das alle Bereichs-Objekte verweisen. Dies ist nötig, um später die zusammengehörenden **`Access`**-Objekte erkennen zu können.

Jede neu geladene und verlassene Internet-Seite wird dem **`Observer`** gemeldet. Der **`Observer`** ist eine Teilkomponente von Scone, bei der sich Plugins anmelden können, um über bestimmte Ereignisse informiert zu werden. Auf diese Weise können die beim Observer registrierten Plugins die Benutzeraktionen erkennen und darauf reagieren. Die Plugins können dann das **`Access`**-Objekt anfordern und mit Hilfe dessen Fensternamen über die Klasse **`FrameAccess`** auf den richtigen **`AppletConnector`** zugreifen. Dadurch ist es den Plugins möglich, über das entsprechende Applet Befehle an den Browser zu schicken.

4.5.5 Das Objekt FrameAccess

Jede offene Internet-Seite sowie jeder offene Anzeigebereich besitzt ein eigenes Applet, das über eine Socket-Verbindung mit einer eigenen Instanz des **`AppletConnectors`** verbunden ist. Wenn ein Plugin von Scone einen der offenen Browser steuern möchte, muss zuerst der richtige **`AppletConnector`** gefunden werden. Diese Arbeit übernimmt die Klasse **`FrameAccess`** (siehe Abbildung 4.2).

Jede Instanz des **`AppletConnectors`**, die eine neue Applet-Verbindung entgegen nimmt, meldet sich bei dem **`FrameAccess`**-Objekt an. Das **`FrameAccess`**-Objekt trägt den Fensternamen zusammen mit dem Zeiger auf den entsprechenden **`AppletConnector`** in eine Liste ein. Sobald der **`AppletConnector`** die Verbindung zum Applet verliert, meldet sich dieser beim **`FrameAccess`** ab.

Die Scone-Plugins können sich vom **`FrameAccess`**-Objekt durch Angabe des Fensternamen den Zeiger auf den passenden **`AppletConnector`** holen, um dann Befehle an den entsprechenden Browser schicken. Den benötigten Fensternamen erhalten die Plugins unter anderem vom **`Observer`** (Abschnitt 4.5.4). Mit Hilfe des **`FrameAccess`**-Objekts können die Plugins beispielsweise alle momentan geöffneten Browser-Fenster anzeigen oder schließen.

4.5.6 Die History

Alle vom Benutzer besuchten Seiten werden in Form von **`Access`**-Objekten in einer History abgelegt. Die dafür zuständige Klasse heißt **`BrowserHistory`**. Der **`EventDecoder`** übergibt der **`BrowserHistory`** das zu speichernde **`Access`**-Objekt und den Namen des Browser-Fensters. Diese legt dann bei Bedarf eine **`FrameHistory`** an, in der alle Zugriffe dieses Browser-Fensters gespeichert werden.

Im Gegensatz zu dem **`AppletConnector`** und dem **`EventDecoder`**, von denen für jede Applet-Verbindung jeweils eine neue Instanz erzeugt wird, existiert von der **`BrowserHistory`** nur eine Instanz, auf die alle

EventDecoder-Instanzen zugreifen. Das ist nötig, da ein **EventDecoder** in speziellen Fällen auf alle **FrameHistory**-Objekte zugreifen muss.

Falls eine Seite aus mehreren Anzeigebereichen besteht (Abschnitt 4.8), wird statt des **Access**-Objekts ein **FrameSet**-Objekt an die **BrowserHistory** übergeben. Das **FrameSet**-Objekt enthält eine Liste aller zu den Anzeigebereichen gehörender **Access**-Objekte. Derartige **FrameSet**-Objekte werden genau wie **Access**-Objekte in der **FrameHistory** abgelegt.

Beim Navigieren mit mehreren Anzeigebereichen wird im Allgemeinen nur einer der Bereiche neu geladen, sobald ein Link angeklickt wird. In diesem Fall wird auch nur eine der Socket-Verbindungen beendet und von einem neuen Applet erneut aufgebaut. Mit Hilfe von JavaScript-Befehlen und HTML-Attributen kann ein Link aber auch mehr als einen Bereich aktualisieren. Da der **EventDecoder** nicht weiß, ob durch den Link eine oder mehrere Seiten neu geladen wurden, besorgt er sich über die **BrowserHistory** das letzte **FrameSet**-Objekt, und kopiert alle noch als geöffnet markierten Bereiche in ein neues **FrameSet**. Zuvor stellt er sicher, dass nur er Zugriff auf diese **FrameHistory** hat, damit nicht zwei oder mehr **EventDecoder** gleichzeitig Kopien des letzten **FrameSet**-Objektes erzeugen. Falls nun noch andere Bereiche aktualisiert werden, erkennt der jeweilige **EventDecoder**, dass das **FrameSet** bereits kopiert wurde und fügt sein eigenes **Access**-Objekt in das neue **FrameSet** ein.

4.6 Direkt abgefangene Ereignisse

In diesem Abschnitt werden die Ereignisse beschrieben, die direkt erfasst werden können. Dazu gehören beispielsweise angeklickte Verweise, die beim Anklicken eine JavaScript-Funktion aufrufen. Im Abschnitt 4.7 folgen die Ereignisse, die nur über Rückschlüsse ermittelt werden können. Ein Beispiel dafür ist das Anklicken des Vor- oder Zurück-Buttons, das vom JavaScript nicht bemerkt wird.

4.6.1 Neue geöffnete Seite

Das Öffnen einer neuen Internet-Seite erkennt der **`EventDecoder`** daran, dass das Applet die ersten Daten über die neue Seite schickt. Der **`EventDecoder`** legt daraufhin ein **`Access`**-Objekt an, in das er alle empfangenen Informationen einträgt.

Um den Grund für den Seitenwechsel, der das Laden dieser Seite verursachte, zu ermitteln, fordert der **`EventDecoder`** über die **`BrowserHistory`** den Grund für den letzten Seitenwechsel an. Diese Information wird von den Ereignissen „geklickter Link“ und „abgeschicktes Formular“ dort hinterlegt. Existiert keine derartige Information für dieses Browser-Fenster, dann muss der Grund für den letzten Wechsel mit Hilfe weiterer Daten über Rückschluss ermittelt werden. Das wird im Abschnitt 4.7.3 genauer beschrieben.

Nach dem Eintragen der Daten in das Access-Objekt, erzeugt der **`EventDecoder`** über den **`Observer`** ein neues Ereignis, das alle Plugins, die sich dafür beim **`Observer`** angemeldet haben, über das Laden einer neuen Seite informiert.

4.6.2 Fertig geladene Seite

Sind alle Dateien der angeforderten Internet-Seite geladen, dann tritt das HTML-Ereignis „**`OnLoad`**“ ein. Durch das entsprechende Attribut, das der Document-Editor im Body-Tag der HTML-Datei eingefügt hat, wird die entsprechende JavaScript-Funktion aufgerufen und eine Zeichenkette mit dem Namen des Ereignisses und dem aktuellen Zeitpunkt über die Socket-Verbindung an den **`EventDecoder`** geschickt. Hier wird von dem Zeitpunkt des Ereignisses der Zeitpunkt des Seitenstarts abgezogen und damit die Ladezeit berechnet.

4.6.3 Abgeschicktes Formular

Sobald der Benutzer ein Formular abschickt, wird mit Hilfe des JavaScript-Codes, des Applets, des RAS und des **`AppletConnector`**s die

entsprechende Methode des **EventDecoders** aufgerufen. Dabei werden der Zeitpunkt, zu dem das Formular abgeschickt wurde, und das Ziel des Formulars übertragen. Das Ziel eines Formulars steht im „**action**“-Attribut des jeweiligen Formulars. Dabei handelt es sich entweder um eine E-Mail-Adresse mit vorangestelltem „**mailto:**“ oder um die Adresse eines CGI-Scripts, das die Formulardaten auswertet. Um die Daten einheitlich in dem **Access**-Objekt ablegen zu können, wird ein **Link**-Objekt erzeugt, dem als Ziel das Ziel des Formulars übergeben wird. Auf diese Weise lässt sich das Formularziel im **Access**-Objekt in das Feld der Link-Id speichern, das in diesem Fall nicht gebraucht wird.

Um den Grund für den Seitenwechsel in das **Access**-Objekt der nächsten Seite eintragen zu können, wird das Abschicken eines Formulars als Grund für den Seitenwechsel in der **FrameHistory** gespeichert. Beim Öffnen der nächsten Seite wird der **EventDecoder** diese Information finden und sie in das **Access**-Objekt eintragen.

4.6.4 Angeklickter Verweis

Wenn der Benutzer einen Verweis der Internet-Seite anklickt, wird dem **EventDecoder** dies mit Hilfe der entsprechende JavaScript-Funktion, des Applets, des RAS und des **AppletDecoder**s mitgeteilt. Der **EventDecoder** erhält auf diesem Wege die Link-Id, die er in das **Access**-Objekt einträgt.

Mit der Link-Id wird das **Link**-Objekt angefordert, in dem unter anderem das Ziel des Verweises als Node-Id gespeichert ist. Über die Ziel-Node-Id lässt sich das entsprechende **NetNode**-Objekt anfordern und dessen URL auslesen. Diese Ziel-URL wird mit der aktuellen URL verglichen um Verweise auf die gleiche Seite (Selflink) oder auf einen bestimmten Teil der gleichen Seite (Target) zu erkennen. Die ermittelte Art des Verweises wird über die **BrowserHistory** in der **FrameHistory** als Grund für den letzten Seitenwechsel gespeichert, so dass der **EventDecoder** der nächsten Seite den Grund auslesen und in das **Access**-Objekt eintragen kann.

Im Fall von mehreren Anzeigebereichen muss beachtet werden, ob der angeklickte Verweis diesen oder einen anderen Bereich aktualisiert. Häufig wird ein Bereich als Inhaltsverzeichnis genutzt, über dessen Verweise in einem anderen Bereich die entsprechenden Seiten geladen werden (Abschnitt 4.8.2). In diesem Fall muss in dem **`Access`**-Objekt des Zielbereichs die Link-Id eingetragen werden, da dort die neue Seite geladen wird. Um dies zu erreichen wird der Wert des „**`target`**“-Attributs aus dem **`Link`**-Objekt ausgelesen, und somit der Name des Zielfensters, in das die Seite geladen werden soll, ermittelt. Damit lässt sich das richtige **`Access`**-Objekt anfordern und die Link-Id eintragen. Genauso muss in der **`FrameHistory`** dieses Bereiches der Grund für den letzten Seitenwechsel gespeichert werden.

4.6.5 Verlassen einer Seite

Beim Verlassen einer Seite wird über den **`AppletConnector`** die entsprechende Methode des **`EventDecoders`** aufgerufen. Hier wird mit Hilfe des Zeitpunktes des Ereignisses die Verweildauer auf der Seite berechnet und in das **`Access`**-Objekt eingetragen. Danach wird den Plugins über den **`Observer`** mitgeteilt, dass die Seite verlassen wird.

Sobald die Internet-Seite beendet ist, ist auch das Applet nicht mehr verfügbar. Damit die Plugins nicht mehr versuchen, über den **`AppletConnector`** auf das Applet zuzugreifen, meldet sich der **`AppletConnector`**, sobald die Verbindung zum Applet beendet ist, beim **`FrameAccess`**-Objekt (Abschnitt 4.5.5) ab.

4.7 Rückschlüsse auf Benutzeraktionen

Viele Benutzeraktionen können über den JavaScript-Code oder das Applet nicht direkt abgefangen werden. Mit Hilfe weiterer Informationen lassen sich aber Rückschlüsse ziehen, die meistens mit 100prozentiger Sicherheit auf eine bestimmte Aktion des Benutzers deuten.

4.7.1 Zurück- und Vor-Button

Mit den Buttons „Zurück“ und „Vor“ lassen sich bereits besuchte Internet-Seiten anzeigen. Die meisten Browser bieten dem Benutzer zusätzlich an, über ein kontextsensitives Menü (meist rechte Maustaste auf dem Zurück- oder Vor-Button) eine Seite aus der Liste der besuchten Seiten direkt auszuwählen.

Sowohl die angeklickten Buttons als auch die ausgewählten Menüpunkte lassen sich nicht per JavaScript abfangen. Das hier beschriebene Verfahren nutzt die Tatsache, dass der Browser die Seiten, die mit dem Zurück- oder Vor-Button ausgewählt werden, aus seinem Zwischenspeicher (Cache) lädt, um sie so schneller anzuzeigen. Die aktivierte Zwischenspeicherung des Browsers ist die Voraussetzung dafür, dass die Erkennung des Zurück- und Vor-Buttons funktioniert. Bei den meisten Browsern ist die Zwischenspeicherung standardmäßig aktiviert.

Wie bereits im Abschnitt 4.3 beschrieben, fügt der Proxy-Server bei der Generierung der externen JavaScript-Datei eine eindeutige Identifikationsnummer in den Code ein. Diese wird beim Start des Applets über die Socket-Verbindung an den **`EventDecoder`** übermittelt, der sie, zusammen mit dem aktuellen **`Access`**-Objekt, in der **`FrameHistory`** speichert. Jedes Mal wenn der Benutzer eine Seite über einen Link oder durch Eingabe einer neuen URL in der Adresszeile anfordert, wird die JavaScript-Datei neu generiert und mit einer neuen Identifikationsnummer versehen. Lediglich der Seitenaufruf über den Zurück- oder Vor-Button – oder deren kontextsensitive Auswahlmenüs – veranlassen den Browser dazu, die Seiten nicht über den Proxy-Server zu laden, sondern die lokale Kopie aus dem Cache zu benutzen. Die Kopie der Seite enthält noch die alte Identifikationsnummer. Diese wird nach dem Start des Applets an den **`EventDecoder`** übertragen und dort mit den Nummern in der **`FrameHistory`** verglichen. Wenn die Nummer bereits existiert, stammt die Internet-Seite aus dem Cache des Browsers und wurde über den Zurück- oder Vor-Button angewählt.

In der **FrameHistory** sind die einzelnen Zugriffe chronologisch abgelegt. Deshalb kann nun abgezählt werden, wie viele Seiten der Benutzer zurück- oder vorgesprungen ist.

Dieses Verfahren kann allerdings nicht unterscheiden, ob eine Seite über den Zurück-Button angesprungen wurde, oder ob der Benutzer den Menüpunkt „Zurück“ über eines der Menüs angewählt hat.

4.7.2 Öffnen eines neuen Browser-Fensters

Jede in einem Browser geöffnete Seite aus dem Internet startet ein Applet, das die Daten der Seite über eine Socket-Verbindung an die auswertende Applikation weitergibt. Für diese Seite wird ein **Access**-Objekt angelegt, das unter anderem in der zum Fensternamen passenden **FrameHistory** gespeichert wird. Zunächst wird abgefragt, ob diese **FrameHistory** bereits existiert, da sie andernfalls erzeugt werden muss. Wenn noch keine **FrameHistory** für diesen Fensternamen existiert, dann handelt es sich um die erste Seite dieses Browser-Fensters, das demnach gerade gestartet wurde.

Werden die Instanzen des Browsers standardmäßig mit einer leeren Seite oder einer lokalen Datei gestartet, dann kann dies nicht erkannt werden, da das Applet – und damit die Erkennung der Seite – nicht eingebunden wird. Informationen über die angezeigte Seite lassen sich nur dann sammeln, wenn die Seite bei einem Web-Server angefordert wird. Nur dann werden das Applet und der JavaScript-Code, die für den Transfer der Seitendaten zuständig sind, durch den Proxy-Server in die angeforderte Seite eingefügt.

4.7.3 Der Aktualisieren-Button

Das Anklicken des Aktualisieren-Button lässt sich folgendermaßen feststellen: Wenn der Benutzer einen Link anklickt oder ein Formular abschickt, wird dies in der **FrameHistory** als Grund für den Seitenwechsel gespeichert. Der **EventDecoder** der nächsten Seite kann diesen Grund auslesen und in dem **Access**-Objekt ablegen. Findet der nächste

`EventDecoder` keine derartige Information vor, dann wird das aktuelle **`Access`**-Objekt mit dem letzten verglichen. Stimmen die Node-Id und die URL-Anhängsel überein, dann wurde die gleiche Seite neu geladen.

Leider kann nicht unterschieden werden, ob der Benutzer die Funktion „Aktualisieren“ über den entsprechenden Button oder über eines der Menüs aufgerufen hat. Auch die Eingabe der gleichen URL in der Adresszeile wird vom Access-Tracking als angeklickter Aktualisieren-Button erkannt.

4.7.4 Gewählte Favoriten

Einige Browser, darunter der Netscape Navigator, erlauben die Erkennung von ausgewählten Favoriten (Bookmarks). Mit Hilfe von JavaScript lässt sich die URL der zuvor besuchten Internet-Seite abfragen. Seiten, die durch Auswahl eines Favoriten geladen wurden, geben als letzte URL eine bestimmte Zeichenkette an. Im Fall des Netscape Navigators lautet dieser Text „[unknown origin]“. Der **`EventDecoder`** erkennt diesen Text und trägt im **`Access`**-Objekt die Auswahl eines Favoriten als Benutzeraktion ein.

Microsofts Internet Explorer, der laut [Int 2001] meistbenutzte Browser, bietet keine derartige Möglichkeit an, ausgewählte Favoriten zu erkennen.

4.7.5 Eingegebene URL

Das Eingeben und Bestätigen einer URL in der Adressleiste lässt sich nicht durch JavaScript oder das Applet ermitteln. Wenn allerdings die in den Abschnitten 4.6 und 4.7 beschriebenen Benutzeraktionen ausgeschlossen werden können, bleibt nur noch die eingegebene URL als Grund für den Seitenwechsel.

Das Verfahren führt gelegentlich zu falschen Ergebnissen, da z. B. die Erkennung der „gewählten Favoriten“ (Abschnitt 4.7.4) beim Internet Explorer nicht funktioniert. Wenn also beim Internet Explorer nicht fest-

gestellt werden kann, wie die Seite aufgerufen wurde, kann es entweder ein gewählter Favorit oder eine eingegebene URL gewesen sein.

Leider lässt sich die Liste der nicht erkannten Aktionen noch weiter führen. Auf einigen Internet-Seiten wird der Browser per JavaScript nach einem bestimmten Zeitintervall zu einer anderen Seite weitergeleitet. Das lässt sich ebenso wenig erkennen wie z. B. angeklickte Buttons der Link-Leiste des Internet Explorers. Die Link-Leiste ist eine Reihe Buttons, die bestimmte Internet-Seiten aufrufen. All diese Benutzerereignisse werden nicht erkannt und somit als eingegebene URL gespeichert.

4.7.6 Fazit

Obwohl es einige Benutzeraktionen gibt, die vom Access-Tracking nicht erkannt werden, werden doch die wichtigsten Aktionen nahezu 100prozentig erkannt. Dabei handelt es sich um die angeklickten Links sowie die Funktionen „Zurück“ und „Vor“ innerhalb der History des Browsers. Catledge und Pitkow [Cat 1995] fanden heraus, dass angeklickte Links und der „Zurück“-Button 52% und 41% der Benutzeraktionen ausmachten. Tauscher und Greenberg [Tau 1997] gaben die relative Häufigkeit der benutzten Links mit ca. 50% und die des „Zurück“-Button mit ca. 30% an.

4.8 Umgang mit Frames

Die Möglichkeit, Internet-Seiten in frei definierbare Bereiche unterteilen zu können, wurde mit den Browserversionen Netscape 2.0 und Internet Explorer 3.0 eingeführt. Seit HTML 4.0 gehören die so genannten „Frames“ [Mün 2001] zum HTML-Standard. Mit Hilfe von Frames lässt sich beispielsweise in einem der Bereiche ein Text anzeigen, während in einem anderen Bereich weitere Informationen zu diesem Text angeboten werden.

Die Anordnung und die Größe der einzelnen Bereiche wird in einer HTML-Datei definiert, die vom Browser zuerst geladen wird. Hier wird für jeden Frame der Name einer HTML-Datei angegeben, deren Inhalt in

dem entsprechenden Bereich angezeigt werden soll. Des Weiteren werden den Frames eindeutige Namen zugewiesen, um sie unterscheiden und ansprechen zu können. Die Definitionsdatei enthält keine anzeigbaren Elemente. Abbildung 4.3 zeigt ein Beispiel für eine Frame-Definition.

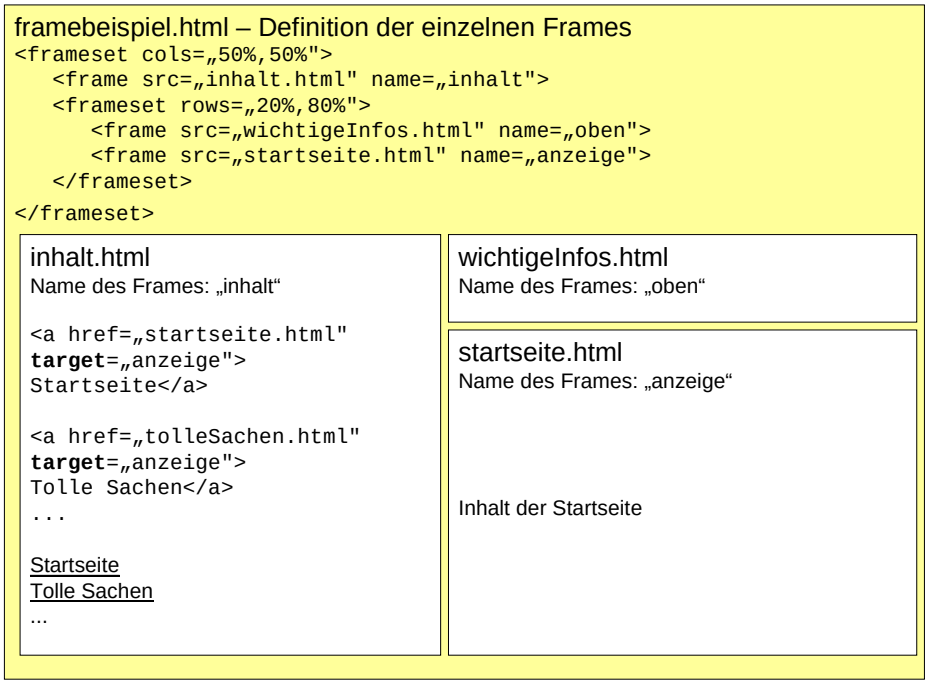

Abbildung 4.3: Beispiel zur Definition von Frames

4.8.1 Erkennen zusammengehöriger Frames

Wird eine Internet-Seite geladen, die mehrere Frames enthält, dann wird in jedem dieser Bereiche eine eigene HTML-Seite angezeigt. Jede dieser HTML-Seiten wurde zuvor vom Document-Editor (Abschnitt 4.3) um JavaScript-Code und ein Applet erweitert. Die Applets schicken nun die gesammelten Informationen zur auswertenden Applikation. Die auswertende Applikation nimmt also von jedem Applet eine Verbindung entgegen und legt je ein **`Access`**-Objekt in der Datenbank ab.

Um nachträglich zu bestimmen, welche der **`Access`**-Objekte als Frames gemeinsam angezeigt wurden, werden zusätzliche Informationen benötigt. Die Zusammenfassung gleichzeitig erstellter **`Access`**-Objekte zu einer Seite ist zu ungenau. Das Auslesen der Definitionsdatei würde viele

Informationen über die einzelnen Frames ergeben, allerdings darf diese Datei kein Applet enthalten, da sonst die Frames nicht aufgebaut werden würden.

Mit Hilfe der Fensternamen können die zusammengehörenden Frames eindeutig zugeordnet werden. Jedes Browserfenster erhält vom JavaScript-Code eine eindeutige Bezeichnung (Abschnitt 4.3.3). Die einzelnen Frames können sowohl ihren eigenen Frame-Namen als auch den Fensternamen ihres Definitionsfensters abfragen. Durch die Kombination dieser beiden Namen können die Frames eindeutig identifiziert werden. Alle Frames, deren Definitionsfenster den gleichen Namen haben, gehören zur gleichen Frame-Seite.

Mit Hilfe von JavaScript lässt sich die Anzahl der Frames abfragen. Dadurch erkennt die auswertende Applikation, ob die neue Seite zu einer Frame-Seite gehört. In diesem Fall wird ein zusätzliches **`Access`**-Objekt erzeugt, auf das sich alle zusammengehörenden **`Access`**-Objekte beziehen. Das **`Access`**-Objekt besitzt ein Feld namens Parent-Access-Id, in das die Identifikationsnummer des zusätzlichen **`Access`**-Objekts eingetragen wird. Später können anhand dieser Nummer alle zusammengehörenden Frames erkannt werden.

4.8.2 Verweise auf Frames

Verweise in Frames können in einem anderen Frame eine neue Seite anfordern. Ein Anwendungsbeispiel dafür ist eine Seite, die aus zwei Frames besteht. In einem Frame werden mehrere Links gezeigt. In dem anderen Frame wird Text dargestellt. Klickt der Benutzer einen der Links aus dem ersten Frame an, dann wird der dazu passende Text in dem zweiten Frame geladen. Dies ist möglich, indem das Link-Tag um das „**`target`**“-Attribut erweitert wird. In diesem Attribut wird der Name des Frames angegeben, auf den sich der Link beziehen soll.

Wird auf einer Seite, die keine Frames enthält, ein Link angeklickt, dann wird in der **`FrameHistory`** (Abschnitt 4.5.6) gespeichert, dass ein Link für den Seitenwechsel zuständig war. Dadurch kann die auswerten-

de Applikation beim Anlegen des nächsten **`Access`**-Objekts den Grund des Seitenwechsels auslesen und diesen in dem neuen **`Access`**-Objekt speichern. Im Falle mehrerer Frames muss zunächst das Attribut „**`target`**“ des angeklickten Links ausgelesen werden. Soll der Verweis einen anderen Frame aktualisieren, dann muss in dessen **`FrameHistory`** der Grund für den Seitenwechsel gespeichert werden. Dazu wird mit Hilfe des Frame-Namens aus dem „**`target`**“-Attribut die entsprechende **`FrameHistory`** angefordert und darin der Grund des Seitenwechsels abgelegt.

4.8.3 Frames in Frames

Internet-Seiten mit mehreren Frames, in denen einer der Frames die gleiche Frame-Definition erneut aufruft, werden im Access-Tracking nicht korrekt in die History (Abschnitt 4.5.6) eingetragen. Das Problem hierbei sind die nicht eindeutigen Namen der Frames. Auch viele Browser verhalten sich bei mehrdeutigen Frame-Namen unlogisch.

Jedes **`Access`**-Objekt (Abschnitt 4.5.2) speichert den Namen des aktuellen Frames und den des obersten Frames (Frame-Definition). Auf diese Weise lassen sich Internet-Seiten mit Frames speichern, sofern die Frames nicht ineinander verschachtelt sind. Für jede zusätzliche Frame-Ebene müsste ein weiterer Frame-Name gespeichert werden. Aufgrund der Seltenheit dieser Fälle hat der Autor die Erkennung der Frames auf eine Definitionsdatei mit beliebig vielen Frames beschränkt. Das reicht für die meisten Seiten, die Frames enthalten.

5 Probleme bei der Realisierung des Access-Trackings

Das Projekt Access-Tracking sollte ursprünglich mit allen gängigen Browsern funktionieren. Aufgrund der großen Unterschiede der Browser in der Ausführung von JavaScript und Applets hat sich der Autor auf die beiden meistbenutzten Windows-Browser beschränkt. Das sind, laut der Internet.com Corporation [Int 2001], der Microsoft Internet Explorer in der Version 5.0 bis 6.0 und der Netscape Navigator in der Version 4.x.

Einige Teile des Projektes mussten für jeden Browser separat programmiert werden, um sowohl auf dem Internet Explorer als auch auf dem Netscape Navigator die gleiche Wirkung zu erzielen. Abschnitt 5.1 befasst sich mit den Unterschieden dieser beiden Browser im Bezug auf JavaScript und Applets. Im Abschnitt 5.2 werden die Fälle beschrieben, in denen sich der Internet Explorer nicht an die Spezifikationen hält. In diesen Fällen mussten andere Wege gefunden werden, um benötigte Informationen zu ermitteln. Abschnitt 5.3 handelt von den Problemen mit dem Netscape Navigator. Dieser funktioniert fehlerhaft bei der Ausführung von JavaScript in Verbindung mit mehreren Anzeigebereichen.

5.1 Unterschiede der Browser

JavaScript wurde 1995 von Netscape entwickelt und mit Hilfe der European Computer Manufacturers Association (ECMA) im April 1998 zum Standard [Ecm 1999] erhoben. 1996 baute Microsoft eine eigene Entwicklung, namens JScript, in den Internet Explorer 3.0 ein. JScript interpretiert einen großen Teil der JavaScript-Funktionen und noch weitere Befehle. Das führt dazu, dass aufwendigere JavaScript-Programme auf dem einen Browser funktionieren, auf dem anderen Browser aber möglicherweise fehlerhaft oder gar nicht laufen.

5.1.1 Verstecken des Applets

Ein Beispiel für die Unterschiede der beiden Browser trat beim Verstecken des Applets auf. Das Applet wird durch den Document-Editor und den JavaScript-Code in die Internet-Seite eingebaut und schickt die ermittelten Daten an die auswertende Applikation (Abschnitt 4.4.2). Um den Benutzer nicht beim Betrachten der Internet-Seite zu stören, soll das Applet unsichtbar sein.

Im Internet Explorer wird das Applet-Tag innerhalb einer „div“-Sektion angegeben. Das „div“-Tag wird um das Attribut „style“ mit der Zeichenkette „visibility:hidden;“ erweitert, um den Bereich unsichtbar darzustellen. Zusätzlich muss dem „div“-Tag über das „style“-Attribut eine absolute Position zugewiesen werden.

Im Netscape Navigator lässt sich das Applet nicht vollständig verstecken, hier wird lediglich die Größe des Applets auf einen Pixel in der Höhe und zwei Pixel in der Breite reduziert. Dadurch verschiebt sich der Inhalt der Internet-Seite leicht nach unten, was aber kaum auffällt.

5.1.2 Benutzte Lesezeichen erkennen

Wenn der Benutzer ein Lesezeichen (Favorit) anklickt, lässt sich dies laut Spezifikation nicht durch JavaScript abfangen. Glücklicherweise hält sich der Netscape Navigator der Version 4.x nicht daran. Die Benutzung eines Lesezeichens des Navigators lässt sich am **`Referrer`** erkennen. Der **`Referrer`** [Mün 2001a] ist eine Eigenschaft des **`Document`**-Objekts, das über JavaScript abgefragt werden kann. Nach dem Aufruf eines Lesezeichens sollte der **`Refferer`** laut Spezifikation eine leere Zeichenkette enthalten. Der Navigator legt stattdessen die Zeichenkette "[unknown origin]" im **`Referrer`** ab. Diese Eigenschaft wird per JavaScript ausgelesen und an die auswertende Applikation geschickt. Der **`EventDecoder`** erkennt anhand dieser Zeichenkette, dass die aktuelle Seite durch ein Lesezeichen aufgerufen wurde.

Der Internet Explorer hält sich in diesem Fall strikt an die Vorschriften und übergibt dem **`Referrer`** nach dem Aufruf eines Favoriten eine

leere Zeichenkette. Hier lässt sich nicht feststellen, ob die Seite über ein Lesezeichen aufgerufen wurde.

5.1.3 Probleme beim Speichern von Cookies

Cookies [Mün 2001c] sind Textdateien beschränkter Länge, die auf der Festplatte des Benutzers gespeichert und ausgelesen werden können. Dies kann durch JavaScript-Befehle oder durch den Web-Server geschehen. Der Zugriff auf andere Daten der Festplatte ist mit Hilfe von Cookies nicht möglich.

Scone benutzt Cookies, um die Benutzernummer auf dem Rechner des Benutzers zu speichern. Das hat den Vorteil, dass sich der Benutzer nur einmal anmelden muss. Danach wird sein Browser anhand des Cookies erkannt.

Die einzelnen Browser sind recht wählerisch, was den Aufbau des Cookies angeht: Netscape akzeptiert den Cookie vom Web-Server nur dann, wenn der Host-Name des Absenders mit führendem Punkt übergeben wird. Internet Explorer 5.5 erwartet keinen Host-Namen. Der Internet Explorer 6.0 trägt den Cookie nur ein, wenn der Host-Name ohne führenden Punkt geschrieben wird und dieser Host als vertrauenswürdige Site im Browser eingetragen ist.

Um nicht erst abzufragen, für welchen Browser der Cookie bestimmt ist, werden alle drei Cookie-Versionen zum Browser geschickt. Der Browser trägt den Cookie mit der richtigen Syntax ein und ignoriert die beiden anderen.

5.2 Fehler des Internet Explorers

Im Laufe der Arbeit haben sich mehrere Fehler des Internet Explorers gezeigt, die hier nun beschrieben werden. Um trotzdem die Funktion des Access-Trackings sicherzustellen, mussten Umwege gefunden werden, um z. B. die benötigten Daten zu ermitteln.

5.2.1 Unzuverlässiges Verhalten der History

Die Größe der History des Browsers lässt sich per JavaScript über die Eigenschaft **`length`** des Objekts **`History`** abfragen. Diese Angabe wäre hilfreich gewesen, um Sprünge innerhalb der History erkennen zu können: Die History-Größe bliebe konstant, wenn der Zurück- oder Vor-Button angeklickt wird.

Der Internet Explorer erlaubt zwar das Auslesen der Größe der History, allerdings hängt der ermittelte Wert um einen Seitenaufruf zurück. Dadurch würde eine angeklickte Zurück-Taste erst beim nächsten Seitenaufruf erkannt werden.

Um nicht auf die Größe der History angewiesen zu sein, speichert das Access-Tracking alle aufgerufenen Seiten in einer eigenen History (Abschnitt 4.5.6). Die Buttons „Zurück“ und „Vor“ werden mit Hilfe der eigenen History erkannt, indem die geladenen Seiten mit denen in der History verglichen werden (Abschnitt 4.7.1).

5.2.2 Abweichungen von der Spezifikation für Applets

Ursprünglich sollte das Applet erkennen, ob eine Seite über den „Zurück“- oder „Vor“-Button aufgerufen wurde. In diesem Fall sollte das Applet gestartet, aber nicht initialisiert werden. Der Internet Explorer hält sich allerdings nicht an die Spezifikationen.

Applets sollten unabhängig von der Plattform sein. In der Praxis verhalten sie sich in den einzelnen Browsern (Plattformen) allerdings sehr unterschiedlich. So sollten Applets laut Java-Spezifikation erkennen, ob die Seite, in die sie eingebettet sind, während der aktuellen Sitzung bereits aufgerufen wurde. Die Seite würde dann aus dem Seitenspeicher des Browsers (Browser-Cache) gelanden werden. Diese Erkennung läuft, laut dem Java Tutorial [Sun 2001], folgendermaßen ab: Beim Programmieren eines Applets können die Methoden „**`init()`**“ und „**`start()`**“ erstellt (überschrieben) werden. Der Browser ruft diese Methoden zum Initialisieren und Starten des Applets auf, sobald die Internet-Seite das erste Mal geladen wird. Der Browser legt nun eine Kopie der Internet-Seite,

eine des Applets und weitere der zugehörigen Dateien in dem Seitenspeicher ab. Ruft der Benutzer eine Seite auf, die der Browser in seinem Seitenspeicher hat, werden die Dateien aus dem Seitenspeicher gelesen. In diesem Fall wird das Applet nicht initialisiert sondern in dem Zustand wieder hergestellt, in dem es beendet wurde. Danach wird die Methode „**`start()`**“ aufgerufen, damit das Applet auf seinen Start reagieren kann. Das Applet kann die einzelnen Aufrufe der Methode „**`start()`**“ zählen und erkennt ab dem zweiten Aufruf, dass die Seite aus dem Seitenspeicher kommt.

Auf diese Weise könnte man feststellen, ob eine Seite über den „Zurück“-Button aufgerufen wurde. Microsofts Internet Explorer hält sich allerdings nicht an diese Spezifikation. Hier wird das Applet bei jedem Aufruf der Seite initialisiert und gestartet, unabhängig davon, ob die Seite aus dem Cache kommt.

Um das Anklicken des Zurück- oder Vor-Buttons beim Internet Explorer zu erkennen, wird in der History (Abschnitt 4.5.6) des Access-Trackings jede aufgerufene Seite abgelegt. Mit Hilfe von eindeutigen Nummern in dem JavaScript-Code werden Seiten aus dem Cache des Browsers erkannt (Abschnitt 4.7.1).

5.2.3 Probleme mit dem Caching der besuchten Dokumente

Werden mit dem Internet Explorer nacheinander die Seiten A, B und wieder A besucht, dann wird nicht, wie bei Netscape, die Seite A zweimal im Cache gespeichert, sondern die alte Kopie von A wird durch die „neue Version“ ersetzt. Auf diese Weise wird die Identifikationsnummer der ersten Seite A durch die der zweiten Seite A überschrieben. Das Access-Tracking benötigt die Identifikationsnummer zur Erkennung der Zurück- und Vor-Buttons (Abschnitt 4.7.1). Das hat zur Folge, dass das Access-Tracking beim Springen von B zum ersten A mit dem „Zurück“-Button die Benutzeraktion „Vor-Button“ erkennt. Das liegt daran, dass die geöffnete, erste Seite A die Identifikationsnummer der zweiten Seite A enthält, weil der Internet Explorer die ältere Kopie durch die neue ersetzt hat.

Ein ähnliches Verhalten tritt auf, wenn in einem weiteren Browser-Fenster eine Seite geöffnet wird, die im ersten Browser-Fenster bereits besucht wurde. Die im Zwischenspeicher existierende Seite wird, zusammen mit der Identifikationsnummer, durch die neue überschrieben. Beim Zurückspringen auf die besagte Seite im ersten Browser-Fenster wird das Access-Tracking dann eine neue Seite erkennen, da die ersetzte Identifikationsnummer der Seite nicht in der **`FrameHistory`** für dieses Browser-Fenster existiert. Diese wurde ja zuvor von dem anderen Browser-Fenster überschrieben.

Für den zweiten Fall, in dem die „Zurück“-Button-Erkennung nicht funktioniert, weil in einem anderen Browser-Fenster die gleiche Seite aufgerufen wurde, gibt es eine Lösung: Für jede geladene Seite muss überprüft werden, ob sie bereits in einer anderen **`FrameHistory`** vorkommt. Dann müssen alle Vorkommnisse dieser Seite die Identifikationsnummer der neuen Seite erhalten. Nun passt die Identifikationsnummer der **`FrameHistory`** wieder zu der der ersetzten Kopie im Browser-Cache. Das Problem der mehrfach besuchten Seite in einem Browser-Fenster wird dadurch aber nicht behoben.

5.3 Fehler des Netscape Navigators

Hier werden die Fehler beschrieben, die im Laufe der Studienarbeit beim Netscape Navigator aufgetreten sind. Im Gegensatz zu den Fehlern des Internet Explorers konnten hier alle Fehler umgangen werden.

5.3.1 Probleme mit Frames und JavaScript

Beim Laden einer Internet-Seite in der Frames enthalten sind, führt Netscape Version 4.x gelegentlich den JavaScript-Code der Frames nicht aus. Dadurch wird das Applet nicht gestartet und keine Verbindung zu der auswertenden Applikation aufgebaut. Die auswertende Applikation erhält keine Daten und kann somit die aufgerufenen Seiten nicht in der Datenbank ablegen.

Die einzelnen Ereignisse, die spezielle JavaScript-Funktionen aufrufen, erzeugen Fehlermeldungen sobald sie ihre Daten an das nicht existierende Applet übergeben wollen.

Dieser Fehler tritt besonders bei umfangreichen externen JavaScript-Dateien auf. Es konnte aber nicht festgestellt werden, welche Faktoren tatsächlich zum Abbruch des JavaScript-Programms führen. Seit die JavaScript-Datei des Access-Trackings auf ein Minimum reduziert wurde, werden Frames in den getesteten Netscape-Versionen 4.x sicher erkannt.

5.3.2 Probleme bei der Veränderung der Fenstergröße

Sobald die Fenstergröße des Netscape Navigators verändert wird, wird das Applet gestoppt und neu gestartet. Dadurch wird die Socket-Verbindung zur auswertenden Applikation beendet und neu aufgebaut. Beim Erzeugen des **`Access`**-Objektes für die neue Socket-Verbindung erkennt die auswertende Applikation, dass dieses Objekt bereits existiert, da Benutzername, URL und Startzeit der Seite in dieser Kombination bereits in der Datenbank vorkommen. Nun wird noch überprüft, ob das letzte **`Access`**-Objekt der **`FrameHistory`** mit diesem identisch ist, denn dann handelt es sich um ein erneut gestartetes Applet, dessen Browser-Fenster in der Größe verändert wurde.

6 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Arbeit im Abschnitt 6.1 kurz zusammen und gibt danach einen Überblick über die Möglichkeiten des Access-Trackings (Abschnitt 6.2). Hier werden die wichtigsten Funktionen anhand einer Bildschirmkopie noch einmal zusammengefasst. Abschnitt 6.3 beschreibt Situationen, in denen das Access-Tracking nicht korrekt arbeitet. Im Abschnitt 6.4 werden Anwendungen angerissen, die durch das Access-Tracking ermöglicht oder vereinfacht wurden. Abschnitt 6.5 erläutert zukünftige Erweiterungen, mit denen sich möglicherweise noch mehr Informationen über die Benutzeraktionen oder die angezeigten Internet-Seiten sammeln lassen.

6.1 Zusammenfassung

Das Ziel dieser Arbeit ist es, die Benutzeraktionen beim Surfen im Web mit einem beliebigen Browser abfangen zu können. Die abgefangenen Aktionen sollen detailliert genug sein, um damit Navigationswerkzeuge zu steuern oder statistische Auswertungen zum Benutzerverhalten durchführen zu können.

Im dritten Kapitel wurden zunächst mehrere Ansätze beschrieben, mit deren Hilfe sowohl die Benutzeraktionen als auch einige Daten über die angewählten Internet-Seiten ermittelt werden können. Aus diesen Ansätzen wurde der universellste ausgewählt und implementiert.

Das im Rahmen dieser Studienarbeit programmierte Projekt „Access-Tracking“ ergänzt das Framework Scone um die Erfassung der Benutzeraktionen (Kapitel 4). Neben dem Access-Tracking bietet Scone dem Entwickler viele weitere Komponenten, die bei der Entwicklung von Navigationswerkzeugen und Browser-Erweiterungen behilflich sind.

Die meisten Probleme dieser Arbeit bescherten Bugs in den Web-Browsern. Die Funktionen, die in mindestens einem der Browser fehlerhaft arbeiten, deren korrektes Ergebnis aber für das Access-Tracking wichtig

ist, mussten auf Umwegen ersetzt werden. Diese Probleme und deren Lösungen wurden im fünften Kapitel ausführlich beschrieben.

6.2 Der Funktionsumfang des Access-Trackings

In diesem Abschnitt wird der Funktionsumfang des Access-Trackings beschrieben. Die einzelnen Funktionen arbeiten allerdings nur dann korrekt, wenn der Benutzer den Internet Explorer 5.0 bis 6.0 von Microsoft oder den Netscape Navigator 4.x zum Surfen verwendet.

Die folgenden Aktionen des Benutzers werden erkannt: Das Starten einer neuen Browser-Instanz, das vor- oder zurückspringen in den bereits besuchten Seiten (History), Sprünge über mehrere Seiten in einem Schritt, das Aktualisieren der Seite, das Anklicken eines Verweises, das Abschicken eines Formulars und das Verlassen einer Seite. Die Erkennung eines ausgewählten Lesezeichens (Favoriten) funktioniert nur beim Netscape Navigator.

Die besuchten Seiten werden in einer Datenbank abgelegt, zusammen mit der jeweiligen Benutzeraktion, die zum Seitenwechsel führte. Später können die einzelnen Schritte des Benutzers anhand der Datenbank nachvollzogen und ausgewertet werden. Im Fall von mehreren Browser-Instanzen können die besuchten Seiten sowie die Benutzeraktionen an einem bestimmten Browser-Fenster rekonstruiert werden.

Das Access-Tracking kann die Benutzeraktionen mehrerer Benutzer gleichzeitig erfassen. Dazu muss sich jeder Benutzer einmalig an seinem Browser anmelden. Das Access-Tracking teilt ihm dann eine eindeutige Benutzernummer zu, die, zusammen mit den besuchten Seiten und den Benutzeraktionen, in der Datenbank gespeichert wird. Dadurch können nachträglich die einzelnen Aktionen eines bestimmten Benutzers nachvollzogen werden. Dies ist beispielsweise für statistische Erhebungen interessant, bei denen bestimmte Benutzergruppen getrennt analysiert werden sollen.

Das Access-Tracking lässt sich verteilt betreiben. Während Scone und das Access-Tracking auf einem Rechner laufen, können die Benutzer mit

ihren Browsern an beliebigen anderen Rechnern arbeiten. Voraussetzung ist lediglich eine TCP/IP-Verbindung zwischen den Benutzerrechnern und dem des Access-Tracking.

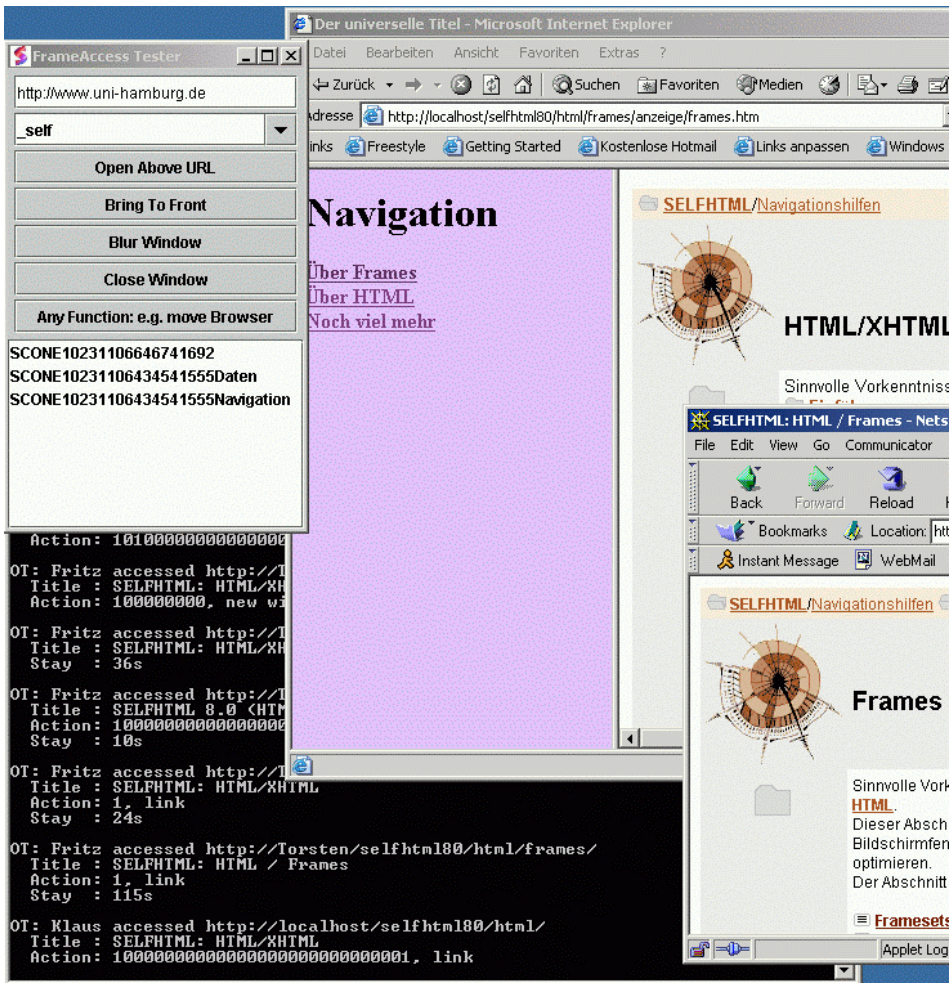

Abbildung 6.1: Kopie des Bildschirms: Access-Tracking mit zwei Browser-Fenstern

Mit Hilfe der Scone-Komponente „Observer“ können sich die Plugins von Scone über bestimmte Ereignisse des Benutzers informieren lassen. Dazu gehören das „Öffnen einer neuen Seite“ und das „Verlassen der aktuellen Seite“. Über das Objekt **`FrameAccess`** des Access-Trackings können die Plugins Daten über die im Browser angezeigte Internet-Seite anfordern und beliebige JavaScript-Befehle an den Browser schicken. Dadurch haben die Plugins die Möglichkeit, die Aktionen des Benutzers

zu verfolgen und bei Bedarf z. B. ein Browserfenster zu schließen oder eine neue Seite aufzurufen.

Abbildung 6.1 zeigt das Access-Tracking in Aktion. Das Fenster in der linken, oberen Ecke zeigt den FrameAccess-Tester. Dieser wurde ursprünglich programmiert, um die Schnittstelle des Access-Trackings zu testen, über die die Plugins auf die aktiven Browser-Fenster zugreifen können. Jedes Plugin, das diese Schnittstelle benutzt, kann auf alle geöffneten Browser-Fenster zugreifen (unterer Teil des FrameAccess-Testers). Einfache Internet-Seiten, wie im Netscape Navigator (rechter Browser), werden mit ihrem eindeutigen Fensternamen in die Liste der offenen Browser-Fenster eingetragen. Wenn in einem Browser-Fenster mehrere Bereiche angezeigt werden, wie im Internet Explorer (oberer Browser), wird der eindeutige Fenstername mit dem Namen des Bereiches verbunden und jeder dieser Bereiche in den FrameAccess-Tester eingetragen. Mit Hilfe der Buttons des FrameAccess-Testers lässt sich in einem beliebigen Browser-Fenster oder einem beliebigen Bereich eine neue Seite anfordern. Des Weiteren lassen sich Browser-Fenster öffnen, schließen, in den Vorder- oder Hintergrund bringen. Es kann auch ein beliebiger JavaScript-Befehl an den Browser geschickt werden.

Das Fenster unten links zeigt einige Daten der geöffneten Seiten an, die die Plugins über den Observer anfordern können. Dazu gehört der Name des Benutzers, die URL der aufgerufenen Seite, der Titel der Seite und die Benutzeraktion, die zum Seitenwechsel führte.

6.3 Grenzen des Access-Trackings

Das Access-Tracking funktioniert lediglich mit dem Browser Netscape Navigator in der Version 4.x und dem Microsoft Internet Explorer Version 5.0 bis 6.0. Die Einschränkung auf diese beiden Browser liegt an den großen Unterschieden der einzelnen Browser in der Ausführung von JavaScript und Applets. Da sich die Unix-Version des Netscape Navigators 4.x anders verhält als die entsprechende Windows-Version, ist Access-Tracking auf das Betriebssystem Microsoft beschränkt.

Wird eine bestimmte Internet-Seite während einer Sitzung mehrfach besucht, hält der Internet Explorer immer nur die Kopie der letzten Seite im Cache (Zwischenspeicher für Internet-Seiten). Das führt dazu, dass die Erkennung des Zurück- und Vor-Button beim Zurückspringen auf das erste Auftreten dieser Seite fehlerhaft arbeitet (Abschnitt 5.2.3).

Des Weiteren können ausgewählte Favoriten des Internet Explorers nicht erkannt werden, da sich ein derartiges Ereignis nicht mit Hilfe von JavaScript von z. B. der Eingabe einer URL unterscheiden lässt. Auch die Link-Leiste, eine Reihe von Buttons, die beim Anklicken bestimmte Seiten aufrufen, lässt sich durch das Access-Tracking nicht erkennen. All diese Ereignisse werden vom Access-Tracking als vom Benutzer eingegebene URL erkannt.

Ein weiteres Problem stellen spezielle Internet-Seiten dar, die den automatischen Start des JavaScripts des Access-Trackings durch eigenen JavaScript-Code verhindern. Das hat zur Folge, dass das Applet nicht in diese Seite eingebaut wird und somit keine Daten an die auswertende Applikation übertragen werden. Diese Fälle kommen allerdings selten vor.

6.4 Anwendungsgebiete des Access-Trackings

Das Access-Tracking ergänzt das Framework Scone um die Erfassung der Aktionen von Benutzern beim Browsen. Es ermittelt Daten über die aufgerufenen Internet-Seiten und erlaubt die Steuerung des Browsers.

Zum einen erlauben die ermittelten Informationen die Steuerung von Navigationswerkzeugen. Mit Hilfe der anderen Komponenten von Scone lassen sich derartige Werkzeuge und andere Browser-Erweiterungen entwickeln und testen. Anhand der Informationen des Access-Trackings können die entwickelten Plugins auf die Aktionen des Benutzers reagieren. Denkbar sind Werkzeuge, die dem Benutzer eine visuelle History anbieten oder für ihn zu bestimmten Seiten navigieren. Das Access-Tracking wird bereits bei der Entwicklung derartiger Projekte im Fachbereich Informatik der Universität Hamburg eingesetzt.

Zum anderen ist es mit den vom Access-Tracking erfassten Daten möglich, Studien zum Benutzerverhalten beim Surfen im Web durchzuführen. Die erfassten Informationen des Access-Trackings sind detaillierter als die Daten, die auf ausgewerteten History-Dateien basieren. Im Gegensatz zu der Studie „What Do Web Users Do“ [Coc 2000], erkennt das Access-Tracking wiederbesuchte Seiten und den Gebrauch der Buttons „Zurück“ und „Vor“. Obwohl vom Access-Tracking nicht alle Aktionen des Benutzers beim Surfen erkannt werden, werden fast alle wichtigen Aktionen erfasst. Besonderer Wert wurde dabei auf die Aktionen gelegt, die die Navigation betreffen.

6.5 Zukünftige Erweiterungen

Der JavaScript-Code und das Applet sorgen in der aufgerufenen Seite für die Erkennung der Benutzeraktionen. Diese Kombination kann allerdings nicht alle Aktionen erfassen.

Das Design ist so gehalten, dass die Kommunikation über das Applet, durch ein anderes Verfahren ersetzt werden kann. Dazu muss lediglich eine Klasse ausgetauscht und eine zweite angepasst werden. Dies ist sinnvoll, sobald es einen besseren Weg gibt, die Benutzeraktionen und Seitendaten abzufangen.

Ein vielversprechender Ansatz ist das Verfahren der abgefangenen Fensterereignisse, kombiniert mit dem Browser-Helper-Object des Internet Explorers (Abschnitt 3.8.1). Dieser Ansatz ist etwas mächtiger als das Access-Tracking, funktioniert aber nur mit dem Internet Explorer und nur unter Windows. Sobald die Daten und Ereignisse auf diese Weise sicher erfasst werden können, könnte das Access-Tracking auf dieses Verfahren umgestellt werden. Bisher arbeitet die Erfassung der Ereignisse allerdings unzuverlässig.

7 Quellenverzeichnis

[Bie 1997] Bieber, M. & Vitali, F. & Ashman, H., & Balasubramanian, V. & Oinas-Kukkonen, H.: *Fourth Generation Hypermedia: Some Missing Links For The World Wide Web.* In International Journal of Human Computer Studies, Vol. 47 (1), Academic Press, pp. 31-65, 1997

[Bla 2002] Blackmon; M. H.: *Automatically Evaluating The Usability Of Web Sites - Automated Cognitive Walkthrough for the Web (AutoCWW).* In CHI 2002 Workshop, 2002, http://www.usabilityfirst.com/autoevaluation/paper_Blackmon_etal.html

[Bou 2002] Bouvin, N. O. & Zellweger, P. T. & Grønbæk K. & Mackinlay, J. D.: *Fluid Annotations Through Open Hypermedia: Using and Extending Emerging Web Standards*. In Proceedings of the 11th World Wide Web Conference 2002, Honolulu, USA, 2002, http://www2002.org/CDROM/refereed/656/

[Byr 1999] Byrne, M. D. & John, B. E. & Joyce, E.: *A day in the life of ten WWW users*, 1999, http://rice.edu/byrne/Pubs/byrneJohnWeb.pdf

[Cat 1995] Catledge, L. D. & Pitkow, J. E.: *Characterizing browsing strategies in the world-wide web*. In Proceedings of the Third International World Wide Web Conference, volume 28 of Computer Networks and ISDN Systems, Darmstadt, Germany, 1995, http://www.igd.fhg.de/www/www95/proceedings/papers/80/userpatterns/UserPatterns.Paper4.formatted.html

[Coc 2000] Cockburn, A. & McKenzie, B.: *What Do Web Users Do? An Empirical Analysis of Web Use*. In International Journal of Human Computer Studies, 2000, http://www.cosc.canterbury.ac.nz/~andy/papers/ijhcsAnalysis.pdf

[Dra 2001] Draper, S. W.: *The Hawthorn effect: a note*. 2001, http://staff.psy.gla.ac.uk/~steve/hawth.html

[Ecm 1999] ECMA: *ECMAScript Language Specification.* 1999, ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf

[GVU 1998] GVU: *GVU's Tenth WWW User Survey Graphs – Problems Using the Web*. 1998, http://www.gvu.gatech.edu/user_surveys/survey-1998-10/graphs/use/q11.htm

[IBM 1999] IBM: *WBI Development Kit for Java.* 1999, http://www.alphaworks.ibm.com/tech/wbidk

[IBM 2002] IBM: *Web Intermediaries (WBI).* 2002, http://www.almaden.ibm.com/cs/wbi/index.html

[Int 2001] Internet.com Corporation: *BrowserWatch – Stats Station.* 2001, http://browserwatch.internet.com/stats/stats.html

[ISC 2002] Internet Software Consortium: *Internet Domain Survey Host Count.* 2002, http://www.isc.org/ds/hosts.html

[Kau 1993] Kaufman, A. & Bandopadhay, A. & Shaviv, B.: *An Eye Tracking Computer User Interface*. In Virtual Reality Workshop Proceedings, IEEE Computer Society Press, October 1993, pp. 120-121 http://www.cs.sunysb.edu/~vislab/projects/eye/Papers/short.paper.pdf

[Kre 1998] Kreutz, R. & Conradi, H. & Spitzer, K.: *Improved Visual Navigation in Web-Documents*. In Proceedings of AACE ED-Media '98, Freiburg, Germany, 1998, pp. 755-760 http://www.klinikum.rwth-aachen.de/webpages/mib/cbt/papers/1315.ps

[Lew 1982] Lewis, C.: *Using the "Thinking-aloud" Method in Cognitive Interface Design*. IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1982.

[Mic 2000] Microsoft: *DDE Support in Internet Explorer Versions.* 2000, http://support.microsoft.com/support/kb/articles/Q160/9/57.ASP

[Mic 2002] Microsoft: *About Microsoft COM.* 2002, http://www.microsoft.com/com/about.asp

[Mic 2002a] Microsoft: *Microsoft Developer Network – Hooks.* 2002, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/hooks_09kj.asp

[Mic 2002b] Microsoft: *Microsoft Developer Network - About Messages and Message Queues.* 2002, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/messques_3pgz.asp

[Moz 2002] The Mozilla Organization: *Mozilla - Open-source web browser.* 2002, http://www.mozilla.org

[Moz 2002a] The Mozilla Organization: *Mozilla 1.0.1.* 2002, http://www.mozilla.org/releases/stable.html

[Mün 2001] Münz, S.: *SELFHTML 8.0 – Frames.* 2001, http://selfhtml.teamone.de/html/frames/index.htm

[Mün 2001a] Münz, S.: *SELFHTML 8.0 – Referrer.* 2001, http://selfhtml.teamone.de/javascript/objekte/document.htm#referrer

[Mün 2001b] Münz, S.: *SELFHTML 8.0 – Cookies.* 2001, http://selfhtml.teamone.de/javascript/objekte/document.htm#cookie

[Mün 2001c] Münz, S.: *SELFHTML 8.0 – History.* 2001, http://selfhtml.teamone.de/javascript\objekte\history.htm

[May 2000] Matthias Mayer: *Kontextvisualisierung: Browsing-Icons und BrowsingGraphs zur Verbesserung der Orientierung und Navigation im World Wide Web*. In Proceedings of 6. Tagung der Deutschen Sektion der Internationalen Gesellschaft für Wissensorganisation Hamburg, Germany, pp. 267-280, http://asi-www.informatik.uni-hamburg.de/personen/mayer/publications/mayer_isko2000.pdf

[Mys 2002] The MySQL AB Company: *MySQL – The World's Most Popular Open Source Database.* 2002, http://www.mysql.org

[Net 1997] Netscape: *Netscape's DDE Implementation.* 1997, http://developer.netscape.com/docs/manuals/communicator/DDE/index.htm

[Net 1997a] Netscape Communications Corporation : *JavaScript Guide – Chapter 5 LiveConnect.* 1997, http://developer.netscape.com/docs/manuals/communicator/jsguide4/livecon.htm

[Pit 1995] Pitkow J. E. & Recker M. M.: *Using the Web as a survey tool: results from the second WWW user survey*. In Proceedings of Computer Networks and ISDN Systems, Volume 27, pp. 809-822, 1995, http://www.cc.gatech.edu/gvu/user_surveys/survey-09-1994/html-paper/survey_2_paper.html

[Rob 1999] Roberts, S.: *Programming Microsoft Internet Explorer 5.* Microsoft Press, 1999

[Sch 1998] Schroeder, W.: *User Interface Engineering – What Is Eye Tracking Good For*. 1998, http://world.std.com/~uieweb/eyetrack2.htm

[Sch 1998a] Schroeder, W.: *User Interface Engineering – Testing Web Sites With Eye Tracking*. 1998, http://world.std.com/~uieweb/eyetrack1.htm

[Spy 1995] Spyglass Inc.: *Software Development Interface*. 1995, http://www.di.uminho.pt/~fln/NetscapeAPI/iapi.html

[Sun 2001] Sun Microsystems Inc.: *The Life Cycle of an Applet, The Java Tutorial - A practical guide for programmers.* 2001, http://java.sun.com/docs/books/tutorial/applet/overview/lifeCycle.html

[Sun 2002] Sun Microsystems Inc.: *How Java to JavaScript Communication Works in Java™ Plugin*. 2002, http://java.sun.com/products/plugin/1.2/docs/jsobject.html

[Tau 1997] Tauscher, Linda & Greenberg, Saul: *How people revisit web pages: empirical findings and implications for the design of history systems*. In International Journal of Human Computer Studies, Special issue on World Wide Web Usability 47 (1), 1997, S. 97-138 http://ijhcs.open.ac.uk/tauscher/tauscher-nf.html

[W3c 2002] W3Consortium: *Document Object Model (DOM)*. 2002, http://www.w3.org/DOM/

[W3c 2002a] W3Consortium: *Cascading Style Sheets*. 2002, http://www.w3.org/Style/CSS/

[Wei 2002] Weinreich, Harald: *Scone - A Java Framework to Build Web Navigation Tools*. 2002, http://www.scone.de