Seminar Wintersemester 2000/2001

Plugin-Architektur der Browser
Netscape Communicator,
Microsoft Internet Explorer,
Opera

Michael Fischer

Dezember 2000

Lehrstuhl für Wirtschaftsinformatik und Softwaretechnik
der Universität Essen

Inhaltsverzeichnis

Zusammenfassung

WWW-Browser erlauben das dynamische Einbinden von Anwendungen ("Plugins") von Drittherstellern, beispielsweise von Adobe, um PDF-Dateien vom Browser entgegenzunehmen und innerhalb des Browser-Fensters darzustellen. Darüber hinaus existieren für verschiedene Skriptsprachen wie Perl oder TcL Plugins im Quelltext, sowie detaillierte Dokumentation, die über das Internet bezogen werden können. Damit können client-side Skripte über Javascript hinaus eingesetzt werden.

Thema dieser Seminararbeit ist das Einarbeiten in die Plugin-Architektur von Browsern, das Kennenlernen bestehender Plugins für Skriptsprachen und den Entwurf und die Implementierung von plattformunabhängigen Plugins für die Scriptsprache Rexx.

1 Grundlagen

Eine Vielzahl von Informationen im Internet benötigen spezielle Applikationen, um angezeigt, abgespielt, angehört oder ausgedruckt werden zu können. Durch die laufende Neu- und Weiterentwicklung von Dateiformaten muss ein Browser in der Lage sein, Informationen aus dem Internet nicht nur herunterzuladen und gegebenfalls als Datei auf einem Speichermedium abzulegen. Es muss eine Möglichkeit geben, die Information per Mausclick ihrer Bestimmung gemäß zu behandeln. Es ist jedoch nicht möglich, alle verfügbaren Dateiformate in einem Browser zu integrieren, einerseits, weil die grosse Masse an Formaten nicht von einem einzelnen Programm bewältigt werden kann, andererseits, weil durch neu entwickelte Formate der Browser ständig ersetzt werden müsste.

Alternativ zur Erweiterung des Browsers werden die Informationen bereits serverseitig aufbereitet und als HTML-Seite übergeben. Der Browser wird nur zur Ein- und Ausgabe benötigt. Allerdings wird dadurch sowohl die Netz- als auch die Serverbelastung erhöht, Video- und Audiodateien können nicht wiedergegeben werden und die Validierung von Benutzereingaben erfordert zusätzlichen Datenverkehr im Netwerk. Es gibt zwar unterschiedliche proprietäre Erweiterungen, sowohl von Microsoft als auch von Netscape, die Multimedia-Daten unter HTML ermöglichen (z.B. der Tag <BGSOUND> [8]). Diese sind aber nicht Teil des HTML-Standards [15].

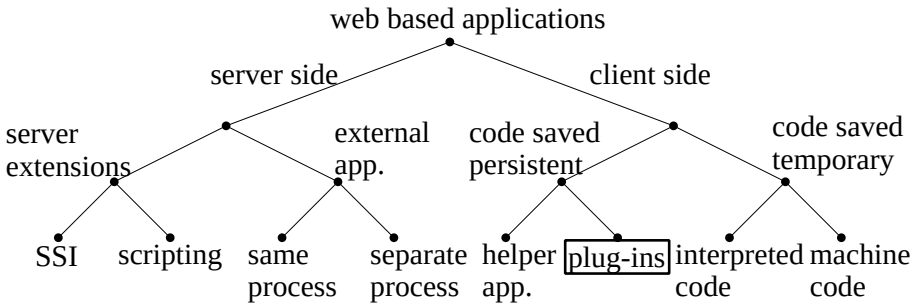

Abbildung 1: Klassifikation Web-basierter Anwendungen [13]

Abbildung 1 skizziert die Möglichkeiten client-seitiger und server-seitiger Erweiterungen. Der Fokus dieser Seminararbeit liegt auf der client-seitigen Erweiterung durch Plugins. Der Begriff "Plugin" ist nicht genau definiert. Der Auszug aus [14] erklärt den Begriff wie folgt:

"Starting with an application, the common characteristics are that:

- *It is a standalone program.*
- *A desktop program, including Web browsers, invokes an application in a separate window.*
- *An application normally implements a specific application protocol such as FTP, telnet, or SMTP.*

On the other hand, a plug-in's characteristics are that:

- *It represents an extension to a Web browser.*
- *It implements a specific MIME type in an HTML document.*
- *It normally operates within the browser window.*

And then we have the Java applet. Is it a "small application," or is it something else? A Java applet

- *Is written in the Java language and compiled by a Java compiler*
- *Can be included in an HTML document*
- *Is downloaded and executed when the HTML document is viewed*
- *Requires the Java runtime to execute"*

Im Folgenden werden verschiedene Methoden kurz vorgestellt, die die Darstellungsmöglichkeiten der Browser Netscape Navigator bzw. Communicator , Microsoft Internet Explorer und Opera über HTML hinaus erweitern. Zur Einordnung des Begriffs Plugin sind neben Netscape's "Plugins" und Microsoft's "ActiveX" auch "Helper Applications" und "Java Applets" aufgeführt.

1.1 Helper Applications

Seit der ersten Version ermöglichen es der Netscape Navigator und Opera, jedem beliebigen Dateiformat eine Anwendung zuzuordnen. Bei der Auswahl eines Links auf eine Datei wird die zugeordnete Anwendung gestartet und stellt den Dateiinhalt dem Format entsprechend dar. Die Anwendung bzw. Helper Application "hilft" dem Browser bei der Verarbeitung des Dateiformats.

Durch die starke Integration des Internet Explorers in das Betriebssystem Microsoft Windows muss bei diesem Browser nicht explizit angegeben werden, welche Anwendung zu welchem Dateityp gestartet werden soll. Die Zuordnung erfolgt über Dateitypen im Explorer, ist also in der Betriebssystemkonfiguration hinterlegt, die Funktionalität ist aber vergleichbar mit der des Netscape Navigators.

Vorteile beim Einsatz von Helper Applications:

- Zu Anfang des Internet-Zeitalters lag der Vorteil von Helper Applications klar auf der Hand: Vereinfachte Handhabung von heruntergeladenen Dateien durch das Starten der zugehörigen Anwendung. Mittlerweile ist dieser Vorteil jedoch aufgrund neuer Methoden in den Hintergrund gerückt.
- Ist ein Bearbeiten der Datei erforderlich, sind Helper Applications nach wie vor sinnvoll einzusetzen.

Nachteile beim Einsatz von Helper Applications :

- Die komplette Datei muss erst heruntergeladen werden, bevor sie angezeigt werden kann (d.h.: kein Streaming bei Audio- und Videodateien möglich).
- Für jedes gewünschte Dateiformat muss eine Anwendung installiert sein, die in der Lage ist, den Dateiinhalt zu verarbeiten. Da es nicht für jedes Dateiformat einen Viewer gibt (wie z.B. Adobe Acrobat Reader, WordView), ist das Bereitstellen von Anwendungen für viele Dateiformate sehr resourcen-intensiv.
- Die Helper Application wird zusätzlich zum Webbrowser gestartet. Die heruntergeladenen Informationen werden also nicht im Kontext der HTML-Seite angezeigt, was zu einer unübersichtlichen Darstellung der Gesamtinformation führen kann.

1.2 Netscape Plugins

Der Begriff Plugin wird von Netscape wie folgt definiert [10]:

> *"A plug-in is a separate code module that behaves as though it is part of the Netscape Communicator browser. You can use the Plug-in API to create plug-ins that extend Communicator with a wide range of interactive and multimedia capabilities, and that handle one or more data (MIME) types. You can even use the Plug-in API to create plug-ins that work in browsers other than the Navigator component of Communicator."*

Mit Version 2 des Netscape Navigators wurde die Plugin-Technologie eingeführt. Im Gegensatz zu Helper Applications sind Plugins keine Stand-Alone-Applications sondern Softwaremodule, die bei Bedarf dynamisch hinzugeladen werden und die Funktionalität des Browsers erweitern. Sie werden unter Microsoft Windows immer als Dynamic Link Library (DLL) bzw. unter Unix/Linux als Shared Object (SO) erzeugt werden. Für die ebenfalls unterstützen Betriebssysteme Mac-OS und OS/2 werden vergleichbare dynamische Bibliotheken erzeugt.

Mit Version 3 des Navigators erweiterte Netscape die Plugin-Technologie um LiveConnect. LiveConnect verbindet die Plugin-Technologie mit Java und JavaScript.

Ab Version 3 des Internet Explorers wird die Plugin-Technologie von Netscape eingeschränkt unterstützt. Die Einschränkungen sind von Microsoft teilweise dokumentiert [4].

Der Browser Opera unterstützt in der Version 4.02 Plugins ausschliesslich unter Windows.

Vorteile von Netscape Plugins

- Plugins werden, im Gegensatz zu Java-Applets einmalig installiert und sind danach immer verfügbar.
- Plugins sind in C/C++ geschrieben und damit leicht auf verschiedene Plattformen portierbar.

Nachteile von Netscape Plugins

- Es existiert kein Sicherheitskonzept wie z.B. die Sandbox von Java. Durch den Einsatz von C/C++ stehen dem Plugin-Entwickler viele Möglichkeiten zur Verfügung, client-seitige Daten zu manipulieren.
- Die Portierbarkeit von Plugins wird durch die Verwendung von Betriebssystembibliotheken erschwert. Die graphische Oberfläche eines Plugins muss speziell für jedes Betriebssystem entwickelt werden.

1.3 Java Applets

Ebenfalls mit Netscape Navigator 2 und Internet Explorer 3 werden Java-Applets unterstützt. Ein Java-Applet ist ein Programm, dass als vorkompilierter Bytcode auf einem Web-Server vorliegt und vom Browser heruntergeladen wird. Zur Ausführung wird der Bytecode von der Java-Virtual-Machine [12], die im Browser integriert ist, interpretiert. Im Gegensatz zu Plugins werden Applets bei jedem Aufruf der HTML-Seite erneut heruntergeladen und ausgeführt.

Der Browser Opera unterstützt in der Version 4.02 Java ausschliesslich unter Windows

Vorteile von Java Applets

- Sicherheit durch Sandbox-Konzept, keine Dateizugriffe und kein Starten externer Programme möglich
- Plattform-Unabhängig
- Bereitstellung einer einheitlichen graphischen Benutzeroberfläche auf allen unterstützten Betriebssystemen

Nachteile von Java Applets

- Java Bytecode wird interpretiert, deswegen sind Java langsamer als z.B. Plugins oder ActiveX.
- Klassenbibliotheken, die von einem Java-Applet benötigt werden, aber nicht lokal verfügbar sind, werden beim Aufruf der HTML-Seite zusätzlich mit heruntergeladen. Dadurch kann sich die Download-Zeit erheblich verlängern.

1.4 ActiveX-Controls

Seit der Version 3 des Internet Explorers stellt Microsoft die ActiveX-Technologie zur Verfügung. ActiveX [6] basiert auf dem Component Object Model [7], eine Schnittstellentechnologie, die theoretisch auf jedem Betriebssystem umgesetzt werden könnte. Allerdings ist COM und damit auch ActiveX nur in Windows-Betriebssystemen verfügbar. Zusätzlich ist ActiveX nur mit dem Internet Explorer einsetzbar.

Um ActiveX-Controls auch für Netscapes Browser oder Opera verfügbar zu machen, hat unter anderem die Firma Esker[3] ein Plugin entwickelt, das als Parameter einen Verweis auf das einzusetzende ActiveX-Control erhält. Allerdings muss dazu der HTML-Code der Internet-Seite mit Hilfe von Javascript angepasst werden. Der Einsatz von ActiveX-Controls bleibt nach wie vor auf Windows-Betriebssysteme beschränkt.

Unter [11] ist anhand einer FAQ-Liste erläutert, warum ActiveX-Controls nicht von Opera unterstützt werden. Hauptgründe liegen in der Abhängigkeit vom Betriebssystem und in der mangelnden Sicherheit der ActiveX-Controls.

Ähnlich wie bei Plugins wird ein ActiveX-Control, falls nicht vorhanden, vom Browser einmalig heruntergeladen und installiert. Ein AktiveX-Control ist nach der Installation systemweit verfügbar und hat Zugriff auf alle Systemresourcen. Es existiert also kein Sicherheitskonzept wie z.B. bei Java-Applets. Andererseits kann ein ActiveX-Control in vielen verfügbaren Programmiersprachen zur Anwendungsentwicklung eingesetzt werden (z.B. MS Visual C++, Borland Delphi).

Vorteile von ActiveX-Controls

- Ausnutzung der betriebssystemspezifischen Möglichkeiten von Windows
- es stehen eine Vielzahl von ActiveX-Komponenten zur Verfügung
- hohe Ausführungsgeschwindigkeit

Nachteile von ActiveX-Controls

- kein Sicherheitskonzept
- nicht plattform-unabhängig

1.5 Auswahl einer geeigneten Technik zur Entwicklung eines Rexx-Plugins

Tabelle 1 beschreibt die Einsatzmöglichkeiten der angesprochenen Technologien mit den möglichen Browser- und Betriebssystemkombinationen. Helper Applications werden hier nicht aufgeführt, da dieser Technik die notwendige Integration in eine HTML-Seite fehlt. Ein weiteres Auswahlkriterium ist die Notwendigkeit, ein exter-

Browser	Applets	ActiveX	Plugins
Internet Explorer	X	X	X
Netscape Navigator für MS Windows	X		X
Netscape Navigator für Linux	X		X
Opera für MS Windows	X		X
Opera für Linux			

Tabelle 1: Einsatzmöglichkeiten von Java Applets, ActiveX und Netscape Plugins auf MS Windows und Linux

nes Programm starten zu können. Da ein vom Webserver übergebenes Rexx-Script interpretiert werden muss, ist es natürlich notwendig, den Interpreter starten zu können. Java Applets sind dazu durch das Sandbox-Prinzip nicht in der Lage.

ActiveX ist ebenfalls nicht geeignet, da diese Technologie der Forderung nach einem plattformunabhängigen Plugin nicht gerecht wird. Zwar ist es möglich, ActiveX-Controls auch unter Netscape Navigator einzusetzen, es existiert aber keine Einsatzmöglichkeit auf anderen Betriebssystemen.

Im Folgenden wird also die Plugin-Technologie von Netscape beschrieben. Grundlage hierzu ist die Dokumentation des "Netscape Developer Plugin-Guide" [10] und das "Netscape Plugin Software Development Kit"[9].

2 Netscape Plugins im Detail

2.1 Einbindung in HTML

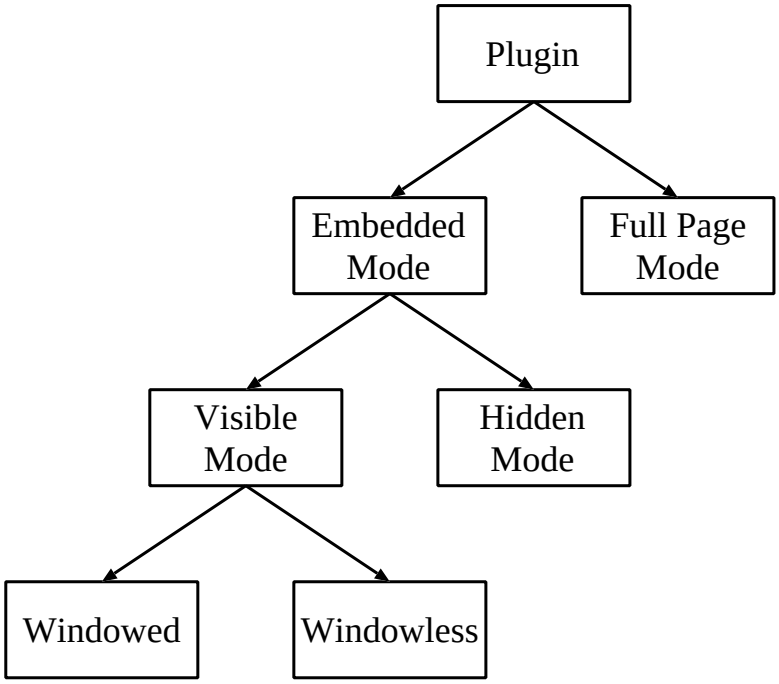

Abbildung 2: Darstellungsmodi von Plugins

Abbildung 2 zeigt die unterschiedlichen Darstellungsmöglichkeiten eines Plugins. Grundsätzlich wird zwischen Embedded Mode und Full Page Mode unterschieden.

2.1.1 Full Page Mode

Durch die HTML-Anweisungen

<A HREF=”myavi.avi”>myavi abspielen</A>

wird ein Link mit dem Text “myavi abspielen” dargestellt. Bei Click auf diesen Link wird das komplette Browser-Fenster vom Plugin-Fenster ausgefüllt. Alternativ kann myavi.avi auch über das Datei-Menü des Browsers geöffnet werden. Im Full Page Mode können keine weiteren Parameter an das Plugin übergeben werden.

2.1.2 Embedded Mode

Zur Einbindung des Plugins in eine HTML-Seite stehen zwei Tags zur Verfügung. Der EMBED-Tag ist nicht Teil der HTML-Spezifikation [15], wird aber von allen Browsern unterstützt.

EMBED-Tag Der Embedded Mode wird durch den HTML-Tag <EMBED> erzeugt. Das Plugin wird an der Stelle der EMBED-Definition in der durch die Optionen WIDTH und HEIGHT angegebenen Größe eingefügt. Die Option SRC gibt den URL der Datei an, die angezeigt werden soll. Falls ein Plugin gestartet werden soll, das keine Dateneingabe benötigt (z.B. ein Plugin, das die Uhrzeit ausgibt),

muss keine SRC-Option angegeben, sondern per TYPE der MIME-Typ des Plugins spezifiziert werden.

Jedes Plugin hat ein Fenster, in dem die graphische Ausgabe erfolgen kann. Das Fenster des Plugins ist in die HTML-Seite integriert, die Position wird durch die Option ALIGN und durch die Position des Tags innerhalb der HTML-Seite angegeben. Wenn das Schlüsselwort HIDDEN im EMBED-Tag angegeben wird, erscheint kein Plugin-Ausgabebereich auf der HTML-Seite. In diesem Fall muss auch keine Angabe über die Höhe und Breite des Plugins (bzw. seines Ausgabebereichs) gemacht werden.

Tabelle 2 listet die wichtigsten Optionen des EMBED-Tags auf: Zusätzlich kön-

Option	Verwendung
SRC	URL der Datei, die angezeigt werden soll
TYPE	MIME-Type des Plugins
ALIGN	Ausrichtung des Plugins: Left, Right, Top, Bottom
BORDER	Dicke des Rahmens um das Plugin-Windows
FRAMEBORDER	"NO", wenn kein Rahmen erwünscht ist
HEIGHT	Höhe des Plugin-Windows
WIDTH	Breite des Plugin-Windows
HIDDEN	Plugin ohne Ausgabe
HSPACE	Horzontaler Rand um das Plugin in Pixel
VSPACE	Vertikaler Rand um das Plugin in Pixel

Tabelle 2: EMBED-Tag

nen weitere Optionen angegeben werden, die vom Browser ignoriert und an das Plugin als Parameter weitergeleitet werden. Die HTML-Zeile

<EMBED SRC="movie.avi" HEIGHT=100 WIDTH=100 LOOP=TRUE>

lädt das mit dem Dateityp ".avi" verknüpfte Plugin und erzeugt einen Ausgaberbereich von 100x100 Pixel. Die Option LOOP hat für den Browser keine Bedeutung und wird ignoriert. Alle vier Optionen werden an das Plugin weitergeleitet und können dort ausgewertet werden.

OBJECT-Tag Im Gegensatz zum EMBED-Tag, der ausschliesslich Plugins einbindet, kann man mit dem OBJECT-Tag auch Applets, ActiveX-Controls, Bilder oder andere beliebige Resourcen anzeigen, die ausserhalb der HTML-Seite liegt und eingebunden werden soll. Der OBJECT-Tag ist mit HTML 4.0 eingeführt worden. Als allgemein gehaltene Objekt-Referenz soll verhindert werden, dass für jede weitere Resource, die zukünftig von HTML unterstützt werden soll, ein neuer Tag definiert wird. Mit Einführung des OBJECT-Tags wird z.B. der APPLET-Tag obsolet, der zur Einbindung von Java-Applets benötigt wurde. Tabelle 3listet die für Plugins notwendigen Optionen mit den Entsprechungen im EMBED-Tag auf. Zur Parameter-Übergabe wird innerhalb eines OBJECT-Tags ein PARAM-Tag definiert. Das EMBED-Beispiel wird als OBJECT-Tag folgendermaßen definiert:

```
<OBJECT DATA="movie.avi" HEIGHT=100 WIDTH=100>
<PARAM NAME="LOOP" VALUE="TRUE">
</OBJECT>
```

Weiterführende Dokumentation gibt es unter anderem bei [15] und [8].

OBJECT-Tag Option	Verwendung	EMBED-Tag Option
DATA	URL der Datei	SRC
TYPE	MIME-Type des Plugins	TYPE
ALIGN	Ausrichtung des Plugins	ALIGN
HEIGHT	Höhe des Plugin-Windows	HEIGHT
WIDTH	Breite des Plugin-Windows	WIDTH

Tabelle 3: OBJECT-Tag

2.1.3 Visible/Hidden Plugins

Der Modus "Visible" ist der Standard-Modus für Embedded-Plugins und wird mit der Option HIDDEN im EMBED-Tag ausgeschaltet. Da es keine Entsprechung für HIDDEN im OBJECT-Tag gibt, muss man sich hier mit den Angaben WIDTH=0 und HEIGHT=0 behelfen. Ein Plugin, dass mit HIDDEN eingebettet ist, hat kein Ausgabefenster und belegt somit auch keinen Platz auf der HTML-Seite.

2.1.4 Windowed / Windowless Plugins

Der Darstellungsmodus "Windowless" ist mit Netscape Communicator 4 eingeführt worden. Im Gegensatz zu Windowed Plugins, die einen definierten Ausgabebereich haben, können Windowless Plugins auf einen beliebigen Zeichenbereich zugreifen, der über ein Window-Handle dem Plugin bekannt gegeben wird. Windowless Plugins werden zur Zeit nicht auf X-Windows-Systemen unterstützt. Aus diesem Grund wird auf eine weitere Ausführung zu diesem Thema verzichtet.

2.2 Funktionsübersicht der NPAPI

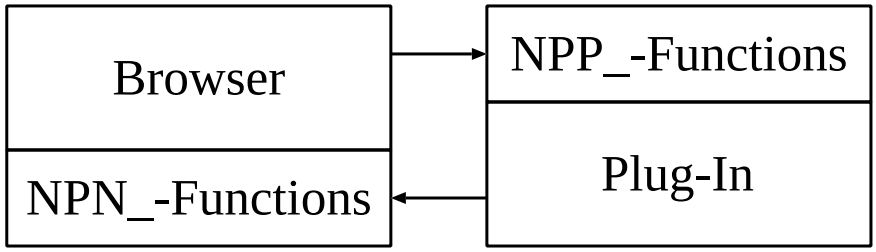

Abbildung 3: Schnittstelle zwischen Browser und Plugin

Die Schnittstelle zwischen Browser und Plugin ist im Netscape Plugin Application Programming Interface (NPAPI) definiert und in zwei Hauptbereiche unterteilt (siehe Abbildung 3). Funktionen, die der Browser zur Verfügung stellt und vom Plugin aufgerufen werden können sind mit "NPN_" benannt. Funktionen, die das Plugin dem Browser zur Verfügung stellt, beginnen mit "NPP_".

Im Folgenden werden die wichtigsten Funktionen aufgelistet. Detailierte Erklärungen der einzelnen Parameter können unter [10] nachgelesen werden. Ein praktisches Beispiel folgt im Kapitel 3.

Tabelle4 listet die wichtigsten Funktionen auf, die das Plugin bereitstellen muss und vom Browser aufgerufen werden. Diese Funktionen stellen die Grundfunktionalität eines Plugins bereit und müssen implementiert werden, da es sonst zu einer Fehlermeldung beim Aufruf des Plugins kommt. Tabelle 5 listet die wichtigsten Funktionen auf, die der Browser für das Plugin bereitstellt.

Funktion	Erklärung
NPP_Initialize	Initialisierung des Plugins
NPP_Shutdown	Beendigung des Plugins, nachdem die letzte Instanz gelöscht wurde
NPP_New	Erstellen einer neuen Instanz des Plugins
NPP_Destroy	Löschen einer Instanz
NPP_NewStream	Initialisierung des Datenempfangs vom Browser an das Plugin
NPP_WriteReady	Nachfrage, ob und wieviele Daten empfangen werden können
NPP_Write	Daten werden vom Browser an das Plugin gesendet
NPP_DestroyStream	Beendigung des Datenempfangs durch den Browser
NPP_SetWindow	Aktualisierung des Plugin-Windows
NPP_GetMIMEDescription	MIME-Type des Plugins
NPP_GetValue	Abfrage des Namens und der Beschreibung des Plug-ins
NPP_GetJavaClass	Rückgabe der mit dem Plugin verknüpften Java-Klasse
NPP_Print	Ausdrucken des Plugin-Fensters
NPP_URLNotify	Benachrichtigung über fertiggestellten URL-Request
NPP_StreamAsFile	Liefert Dateiname über lokal gespeicherten Stream

Tabelle 4: Funtionen, die das Plugin zur Verfügung stellt

Funktion	Erklärung
NPN_MemAlloc	Speicher allokieren
NPN_MemFree	Speicher freigeben,
NPN_GetUrl	Datenanforderung an den Browser
NPN_NewStream	Initialisierung des Datenempfangs vom Plugin an den Browser
NPN_Write	Daten werden vom Plugin an den Browser gesendet
NPN_Destroy	Beendigung des Datenempfangs durch das Plugin

Tabelle 5: Funktionen, die der Browser zur Verfügung stellt

2.3 Programmablauf bei Aufruf einer HTML-Seite

Beim Start des Browser werden alle verfügbaren Plugins registriert. Verfügbare Plugins liegen im Unterverzeichnis "plugins" des Browser-Programmverzeichnisses. Unter Unix wird zusätzlich das Verzeichnis "$(HOME)/.netscape/plugins" vom Netscape-Browser durchsucht. Zur Registrierung des unterstützten MIME-Types werden unter Windows und Unix zwei unterschiedliche Methoden verwendet:

Windows Die Information über den unsterstützten MIME-Type sind in die DLL integriert. Dazu muss eine Resource-Datei mit Versioninformationen erstellt werden, die der Linker zur DLL bindet. Zur Registrierung liest der Browser die Version-Informationen aus der DLL.

Unix Plugins unter UNIX müssen die NPAPI-Funktionen NPP_GetMIMEDescription und NPP_GetValue enthalten, die vom Browser aufgerufen werden. NPP_GetMIMEDescription

gibt einen String der Form "MIME-Type:Datei-Endung:Beschreibung" zurück. NPP_GetValue wird zur Ermittlung des Plugin-Namens und eines Beschreibungs-Textes aufgerufen.

Eine Liste aller verfügbaren Plugins erhält man unter Netscape Navigator/Communicator über das Menü "Hilfe->About Plug-Ins". Opera bietet einen Plugin-Dialog unter "File->Preferences" an. Microsoft Internet Explorer ermöglicht keine Abfrage der installierten Plugins. Man kann sich aber mit Javascript behelfen, indem man Methoden des Objekts "navigator" benutzt. Das Objekt "navigator" steht auch unter MS IE zur Verfügung. Folgendes Beispiel ist [8] entnommen:

```
<html><head><title>Test</title>
</head><body>
<script language="JavaScript">
document.writeln("<table border>");
for(i=0; i<navigator.plugins.length; i++)
{
document.writeln("<tr>");
document.writeln("<td>" + navigator.plugins[i].name + "</td>");
document.writeln("<td>" + navigator.plugins[i].description + "</td>");
document.writeln("<td>" + navigator.plugins[i].filename + "</td>");
document.writeln("</tr>");
}
document.writeln("</table>");
</script>
</body></html>
```

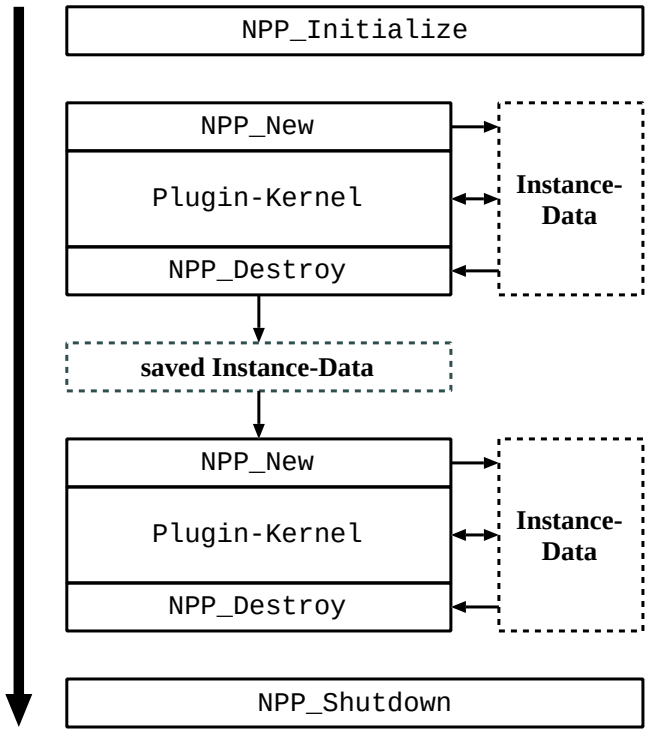

Abbildung 4: Programmablauf eines Plugins

Abbildung 4 zeigt ein grobes Schema der Initialisierung und Instanziierung eines Plugins, das zweimal innheralb einer Seite aufgerufen wird.

NPP_Initialize wird beim ersten Aufbau der HTML-Seite aufgerufen, NPP_Shutdown wird beim Schliessen der Seite aufgerufen, nachdem die letzte Instanz des Plugins durch NPP_Destroy entfernt wurde.

Nach NPP_Initialize wird pro Plugin-Instanz NPP_New aufgerufen. Falls zum Beispiel ein Plugin zweimal auf derselben Seite implemntiert ist, werden beim Anzeigen der Seite zweimal NPP_New aufgerufen, da auch zwei Instanzen gebildet werden. Gleiches gilt, falls ein zweites Browserfenster geöffnet.

Die Instanzdaten werden von jeder Plugin-Instanz in NPP_New erzeugt und in NPP_Destroy wieder entfernt. Eine Instanz kann Daten an eine nachfolgende Instanz übergeben, indem sie in NPP_Destroy einen Zeiger auf einen erzeugten Speicherbereich an den Browser übergibt. Ein Plugin, das durch dieselbe URL aufgerufen wird, erhält im NPP_New-Aufruf diesen Zeiger als Parameter und kann so auf die Daten zugreifen.

Teil des Plugin-Kerns ist das Aktualisieren des Plugin-Fensters durch NPP_SetWindow und der Datenaustausch mit dem Webserver. Wenn vom Webserver Daten zur Verarbeitung durch das Plugin gesendet werden, wird die Funktion NPP_NewStream aufgerufen. Hier kann entschieden werden, in welcher Weise die Daten verarbeitet werden sollen. Grundsätzlich unterscheidet man zwischen stream-orientierter und datei-orientierter Verarbeitung.

Bei stream-orientierten Plugins werden NPP_WriteReady und NPP_Write aufgerufen, bis alle Daten vom Webserver an das Plugin gesendet wurden und der Datenstrom mit NPP_DestroyStream beendet wird. Mit NPP_WriteReady kann das Plugin bestimmen, wieviele Daten in einem Block verarbeitet werden. Diese Anzahl an Daten wird mit dem nachfolgenden Aufruf von NPP_Write vom Webserver gesendet. Der Datenstrom kann vom Plugin jederzeit durch Aufruf der Funktion NPN_DestroyStream gestoppt werden.

Bei datei-orientierten Plugins erzeugt der Browser eine Datei, in der die Daten vom Webserver abgespeichert werden. Anschliessend wird die Funktion NPP_StreamAsFile aufgerufen, die den Dateinamen an das Plugin übergibt. Die Verarbeitung und das Löschen der Datei erfolgt innerhalb des Plugins.

2.4 LiveConnect

1. call Java methods from plug-ins
2. call native methods implemented in plug-ins from Java
3. call Java methods from JavaScripts
4. call JavaScript from Java methods

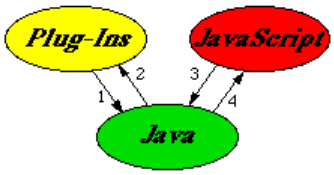

Abbildung 5: LiveConnect, Quelle [10]

LiveConnect wurde mit Netscape Navigator 4 eingeführt. Mit LiveConnect kann man Plugins, Java und JavaScript integrieren. Es ist möglich, aus dem Plugin auf Java-Objekte und umgekehrt, aus dem Java Applet auf Plugin-Funktionen zuzugreifen. Abbildung 5 zeigt, dass man mit JavaScript indirekt auf Plugins zugreifen kann.

LiveConnect benötigt eine von Netscape modifizierte Java Virtual Machine [12], die nur im Netscape Browser integriert ist. Dadurch wird LiveConnect nicht auf Opera oder Internet Explorer unterstützt. Selbst in der neuen Version 6 des Netscape Communicator ist keine modifizierte JVM installiert, was letztendlich langfristig das Ende von LiveConnect bedeutet.

2.5 Plugin Template

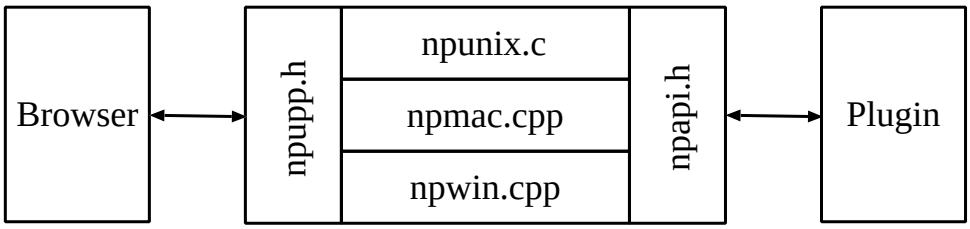

Abbildung 6: Aufteilung Plugin Template

Als Ausgangspunkt der Entwicklung von Plugins bietet das Plugin Software Development Kit ein Plugin Template an, in dem die notwendigen Funktionen bereits vordefiniert sind und vom Plugin-Entwickler nur noch mit "Leben" gefüllt werden müssen. Abbildung 6 stellt die Aufteilung der Dateien des Templates dar. Zur Vereinfachung sind die Dateien zur LiveConnect-Implementation nicht dargestellt.

Da der Plugin-Code plattformabhängig ist, wird von Netscape für jede unterstützte Plattform eine Quelldatei geliefert, die an die Konventionen (z.B. von Funktionsaufrufen) des Betriebssystems angepasst ist. Bei der Erstellung der Plugin-Bibliothek muss die entsprechende Datei verwendet werden, also z.B. npunix.c für Plugins, die unter Unix bzw. Linux kompiliert werden sollen.

Die Header-Datei npupp.h enthält die Funktionsprototypen und die Funktionstabelle, mit deren Hilfe der Browser die Plugin-Funktionen aufruft. Das Initialisieren der Funktionstabelle und Wrapper-Funktionen für die in der Tabelle angegebenen Funktionen sind in den drei Quelldateien npunix.c, npmac.cpp und npwin.cpp definiert.

Für den Plugin-Entwickler ist primär die Header-Datei npapi.h interessant, da hier alle Funktionsprototypen und Strukturen implementiert sind, die im Plugin-Sourcecode verwendet werden können.

Durch Angabe der Compiler-Option XP_UNIX, XP_PC und XP_MAC werden die Quelldateien für die gewünschte Plattform kompiliert. Zusätzlich kann die Compiler-Option PLUGIN_TRACE angegeben werden. Dadurch wird für jeden Funktionsaufruf eine Meldung generiert, die auf die Standardfehlerausgabe (stderr) geleitet wird. So kann man zum Beispiel in 2.3 beschriebene Verhalten bei Aufruf und Schliessen einer HTML-Seite nachvollziehen.

3 Entwicklung eines Rexx-Plugins

Als konzeptionelle Grundlage für das Rexx-Plugin stehen die Implementierungen der Scriptsprachen TCL [16] und Perl [5] als Netscape Plugin zur Verfügung. Beide Varianten basieren darauf, dass der auf dem Client bereits installierte Interpreter aufgerufen wird. Das Script wird als Stream an das Plugin übergeben, in einer temporären Datei gespeichert und als Parameter an den Interpreter übergeben. Der <SCRIPT>-Tag wird für die Entwicklung des Rexx-Plugins nicht betrachtet.

Weiterführende Informationen über die Scriptsprache Rexx erhält man unter [1].

3.1 Anforderung

Es soll ein Plugin auf Basis des Netscape Plugin SDK entwickelt werden. Bei Aufruf eines URL, der auf ein Rexx-Script mit der Dateiendung ".rex" verweist, wird der Inhalt der Datei an den client-seitig installierten Rexx-Interpreter übergeben und von diesem interpretiert. Das Rexx-Script kann HTML-Code erzeugen, der vom Browser angezeigt wird. Dazu muss dem Plugin ein Parameter übergeben werden, der das Ziel des HTML-Codes angibt. Zusätzlich wird dem Plugin ein weiterer Parameter übergeben, der an den Rexx-Interpreter durchgereicht wird. Das Fenster des Plugins wird nicht benötigt, da keine graphische Ausgabe oder Graphical User Interface (GUI) erzeugt wird. Als MIME-Type (Multipurpose Internet Mail Extension) [2] wird "application/x-rexx" definiert. Die Daten werden als Stream empfangen und in einer temporären Datei abgelegt. Zusammenfassend wird das Plugin entsprechend der Beschreibungen aus Kapitel 2.1 folgende Eigenschaften haben:

- Embedded Mode
- Hidden
- stream-orientiert

Der grobe Ablauf des Plugins ist in Abbildung dargestellt.

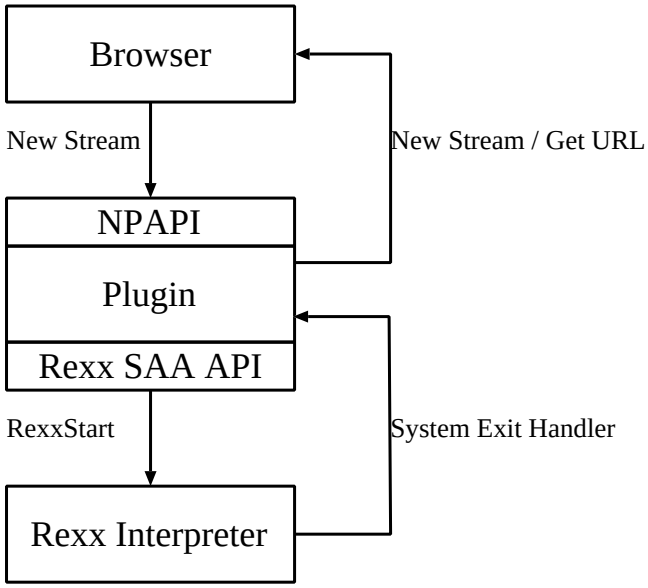

Abbildung 7: Programmablauf des Rexx-Plugins

3.2 Implementierung

Im Folgenden werden nur die "NPP_"-Funktionen vorgestellt, die eine konkrete Aufgabe innerhalb des Plugin-Kerns spielen. Alle anderen in Tabelle 4 aufgeführten Funktionen, die hier nicht angesprochen werden, müssen trotzdem für ein korrektes Plugin zumindest als "leere" Funktion implementiert werden.

Feldname	Datentyp	Beschreibung
Window	NPWindow*	Handle des Pluginfensters
tmpFile	int	Handle der temporären Skriptdatei
tmpFileName	char[L_tmpnam]	Dateiname der temp. Skriptdatei
target	char*	Name des Ziels für HTML-Ausgabe
input	char*	Parameter für Rexx
rexx_stream	NPStream	Zeiger auf den Datenstream zur HTML-Ausgabe
fHTML	int	Handle der HTML-Datei (nur für Windows)
fHTMLFileName	[L_tmpnam]	Dateiname der HTML-Datei (nur für Windows)

Tabelle 6: Instanz-Datenstruktur

3.2.1 NPP_New

Zu jeder neuen Instanz wird Speicher für die Instanzdaten allokiert. Die Struktur der Instanzdaten wird vom Plugin-Entwickler definiert. Die Struktur für das Rexx-Plugin ist in Tabelle 6 beschrieben. Jede Funktion erhält vom Browser einen nicht initialisierten Zeiger instance->pdata. Alle nachfolgend aufgerufenen Funktionen erhalten als Parameter den Zeiger instance als Referenz auf die Plugin-Instanz-Daten.

```
instance->pdata = NPN_MemAlloc(sizeof(PluginInstance));
memset(instance->pdata, 0, sizeof(PluginInstance));
This = (PluginInstance*) instance->pdata;
```

Speicher wird in Plugins immer mit NPN_MemAlloc statt mit malloc allokiert, da der Browser die Speicherverwaltung übernimmt.

Als nächstes werden die Parameter ermittelt, die an das Plugin durch den HTML-Tag übergeben wurden. argn[] enthält die Namen, argv[] die Werte und argc die Anzahl der übergebenen Parameter.

```
for (i=0; i<argc; i++) {
    if (strcmp(argn[i], "target") == 0)
        copy_param(&This->target, argv[i]);
  if (strcmp(argn[i], "input") == 0)
        copy_param(&This->input, argv[i]);
}
```

Mit der Funktion tmpnam, die Teil der ANSI-C-Bibliothek ist, wird ein eindeutiger temporärer Dateiname erzeugt. Mit diesem Dateinamen wird eine Datei angelegt, in der das Rexx-Script gespeichert werden soll.

```
tmpnam(This->tmpFileName);
if ((This->tmpFile = creat(This->tmpFileName, S_IWRITE | S_IREAD))
< 0)
```

```
return NPERR_GENERIC_ERROR;
```

3.2.2 NPP_Destroy

Jede Funktion, die auf die Instanz-Daten zugreifen muss, referenziert diese mit der Anweisung

```
This = (PluginInstance*) instance->pdata;
```

Diese Zeile steht daher am Anfang fast jeder Funktion. Im Folgendem bezeichnet "This" immer einen Zeiger auf die Instanz-Daten.

Durch NPP_Destroy fordert der Browser das Plugin an, alle Daten und Dateien, die durch das Plugin erzeugt wurden, wieder zu löschen. NPN_MemFree() ist das äquivalent zu free(). Mit unlink wird die temporäre Datei geschlossen und anschliessend gelöscht.

```
if (This != NULL) {
    unlink(This->tmpFileName);
    NPN_MemFree(This->target);
    NPN_MemFree(This->input);
    NPN_MemFree(instance->pdata);
    instance->pdata = NULL;
}
```

3.2.3 NPP_NewStream

NPP_NewStream signalisiert, dass der Browser einen Datenstrom an das Plugin senden möchte. Man kann den Datenstrom ablehnen, indem als Rückgabe ein Wert ungleich NPP_NO_ERROR angegeben wird.

3.2.4 NPP_WriteReady

Als Rückgabewert erwartet der Browser die maximale Größe des nächsten Datenpakets, das vom Browser an das Plugin gesendet wird. Für das Rexx-Plugin ist STREAMBUFSIZE=4096 definiert. Hier könnte ein Algorithmus eingefügt werden, der bei hohem Datenaufkommen die optimale Paketgröße berechnet.

```
return STREAMBUFSIZE;
```

Nach jedem NPP_WriteReady wird ein Datenpaket mit der Anzahl STREAMBUFSIZE Bytes an die Funktion NPP_Write geschickt. Dieser Vorgang wiederholt sich solange, bis entweder alle Daten gesendet oder das Plugin per NPN_DestroyStream den Datenstrom abgebrochen hat.

3.2.5 NPP_Write

Die in buffer empfangenen Daten werden per write-Anweisung in die temporäre Datei geschrieben. Es werden maximal STREAMBUFSIZE Bytes Daten geschrieben. Die tatsächlich verarbeitete Anzahl Bytes wird als Rückgabewert an den Browser zurückgegeben.

```
return write(This->tmpFile, (char*)buffer, (size_t)(len<=STREAMBUFSIZE
? len : STREAMBUFSIZE));
```

Anhand des Rückgabewerts berechnet der Browser das nächste zu versendende Datenpaket. Falls statt z.B. 4096 nur 2048 Bytes geschrieben wurden, sendet der Browser die nicht verarbeiteten 2048 Bytes erneut.

3.2.6 NPP_DestroyStream

Nachdem alle Daten an das Plugin gesendet wurden, wird NPP_DestroyStream vom Browser aufgerufen. Die temporäre Datei geschlossen und der Rexx-Interpreter gestartet. Das if-Statement wird im Zusammenhang mit 3.2.9 erklärt.

```
if (stream != This->rexx_stream) {
    close(This->tmpFile);
    return rexx_exec(instance);
}
```

3.2.7 rexx_exec

Um in ein laufendes Rexx-Script einzugreifen, kann man an einigen Stellen sogenannte Hooks (Haken) setzen, die den Programmablauf unterbrechen und die "eingehakte" Funktion aufrufen. Solche Haken werden mit System Exit Handlers realisiert. System Exit Handlers sind Funktionen aus dem Rexx SAA (System Architecture Application) API, der Schnittstelle zwischen Rexx und anderen Programmiersprachen, z.B. C. Weitergehende Informationen können unter [1] abgerufen werden.

Um die Ausgabe des Rexx-Befehls "SAY" abzufangen, wird mit der API-Funktion RexxRegisterExitExe die Funktion hook_RXSIOSAY mit dem gleichlautenden Namen "hook_RXSIOSAY" registriert. Als zusätzlicher Parameter wird der Funktion ein Zeiger auf die Instanz-Daten des Plugins übergeben.

```
UserDataAddr = (long)instance;
sysexit[0].sysexit_code = RXSIO;
sysexit[0].sysexit_name = "hook_RXSIOSAY";
sysexit[1].sysexit_code = RXENDLST;
RexxRegisterExitExe("hook_RXSIOSAY", hook_RXSIOSAY,
(PUCHAR)&UserDataAddr);
```

Anschliessend wird mit der API-Funktion RexxStart der Interpreter aufgerufen. Als Parameter erhält diese Funktion neben dem Namen der temporären Skript-Datei auch den Parameter input, der in NPP_New ermittelt und das Array sysexit, in dem die Namen aller System Exit Handler aufgeführt sind. In diesem Fall ist das nur "hook_RXSIOSAY".

```
arglist[0].strptr = This->input;
arglist[0].strlength = strlen(This->input);
RexxStart(1, arglist, This->tmpFileName, NULL, NULL, RXCOM-
MAND, sysexit, &ReturnCode, &Result);
```

Nachdem das Skript ausgeführt wurde, wird der Hook wieder entfernt.

```
RexxDeregisterExit("hook_RXSIOSAY", NULL);
```

Alternativ könnte man den Hook auch in NPP_New oder NPP_Initialize registrieren.

3.2.8 hook_RXSIOSAY

Bevor der Rexx-Befehl"SAY" vom Interpreter ausgeführt wird, werden eventuell registrierte Hooks aufgerufen und über den Typ des Aufrufs unterrichtet. Der Aufruf wird eindeutig identifiziert durch eine Exit-Number und durch eine Subfunction-Number. Das Rexx-Plugin fängt alle Standard-Ein/Ausgabe-Aufrufe ab (RXSIO) und davon nur den SAY-Befehl (RXSIOSAY). Für alle anderen System Exits wird die Konstante RXEXIT_NOT_HANDLED zurückgegeben. Durch diesen Rückgabewert führt der Interpreter die Ausgabe aus. Falls als Rückgabewert RX_EXIT_HANDLED angegeben wird, überspringt der Interpreter die Ausführung des Befehls.

```
switch (ExitNumber) {
    case RXSIO:
        switch (Subfunction) {
            case RXSIOSAY: return handle_RXSIOSAY(ParmBlock);
            default: return RXEXIT_NOT_HANDLED;
        }
    default: return RXEXIT_NOT_HANDLED;
}
```

3.2.9 handle_RXSIOSAY

Um die Instanzdaten des Plugins zu erhalten, muss Speicher für den Zeiger allokiert werden.

```
UserDataAddr = NPN_MemAlloc(8);
```

Der Speicher wird von der API-Funktion RexxQueryExit mit dem Zeiger belegt, der durch den Aufruf von RexxRegisterExitExe übergeben wurde. Dadurch kann genau wie in den "NPP"_-Funktionen auf die Instanz-Daten mit dem Zeiger This zugegriffen werden.

```
RexxQueryExit("hook_RXSIOSAY", NULL, &Flag, (PUCHAR)UserDataAddr);
instance = (NPP)*UserDataAddr;
This = (PluginInstance*)instance->pdata;
```

An dieser Stelle unterscheidet sich der Code des Plugins für Linux von der Windows-Version. Microsoft Internet Explorer ist nicht in der Lage, die Funktion NPN_NewStream auszuführen. Anscheinend ist die Funktion nicht implementiert.

LINUX: Die Ausgabe des Rexx-Programms wird als Stream an den Browser weitergeleitet. Zum Erzeugen eines Streams vom Plugin zum Browser wird die Funktion NPN_NewStream beim ersten Aufruf des Exit-Handlers einmalig aufgerufen. Nach dem ersten Aufruf von NewStream ist This->rexx_stream initialisiert, sodass kein zweiter NPN_NewStream aufgerufen wird. Der Parameter wird This->target gibt das Ziel der Ausgabe an. Das ist entweder der Name eines Frames der aktuellen HTML-Seite oder einer der Parameter _blank/_new für eine neue Seite, _parent oder _self.

```
if (This->rexx_stream == NULL) {
    if (NPN_NewStream(instance, "text/html", This->target,
  &This->rexx_stream) != NPERR_NO_ERROR) {
        NPN_MemFree(UserDataAddr);
        return RXEXIT_HANDLED;
```

```
        }
    }
```

Für alle Aufrufe des Exit-Handlers wird an NPN_Write die Ausgabe des Rexx-Programms übergeben. ParmBlock ist ein Zeiger auf einen Speicherbereich, der Daten für den Exit-Handler bereithält. Im Falle des Exit-Handlers RXSIO:RXSIOSAY ist das ein Zeiger auf eine RXSTRING-Struktur, die die Ausgabe des Rexx-Programms enthält.

```
    NPN_Write(instance, This->rexx_stream,
((EXIT*)ParmBlock)->siosay.rxsio_string.strlength,
((EXIT*)ParmBlock)->siosay.rxsio_string.strptr);
    NPN_MemFree(UserDataAddr);
    return RXEXIT_HANDLED;
```

Mit RXEXIT_HANDLED wird dem Interpreter angezeigt, dass die Ausgabe des SAY-Kommandos erfolgreich abgeschlossen ist.

Der restliche Programmcode stammt aus der Funktion rexx_exec. Nachdem der Exit-Handler durch RexxDeRegister wieder entfernt wurde, wird zum Abschluss der Datenstrom zum Browser durch NPN_DestroyStream beendet.

```
    #ifdef XP_UNIX
    if (This->rexx_stream != NULL)
        NPN_DestroyStream(instance, This->rexx_stream, NPRES_DONE);
#endif /* XP_UNIX */
```

WINDOWS: Da der Internet Explorer keinen Stream akzeptiert, wird eine temporäre Datei erzeugt, die die Ausgabe des SAY-Befehls speichert. Genau wie NPN_NewStream wird auch hier das Kreieren der Datei nur einmal vorgenommen.

```
    if (This->fHTML == 0) {
        tmp = _tempnam(NULL, "rexx");
        strcpy(This->fHTMLFileName, tmp);
        free(tmp);
        if ((This->fHTML = creat(This->fHTMLFileName, S_IWRITE
  | S_IREAD)) < 0) {
            NPN_MemFree(UserDataAddr);
            return RXEXIT_HANDLED;
    }
}
```

Die Ausgabe des SAY-Befehls wird per write in die Datei geschrieben.

```
    write(This->fHTML,
    ((EXIT*)ParmBlock)->siosay.rxsio_string.strptr,
    ((EXIT*)ParmBlock)->siosay.rxsio_string.strlength);
    NPN_MemFree(UserDataAddr);
    return RXEXIT_HANDLED;
```

Und ebenfalls in rexx_exec wird zum Abschluss die Datei geschlossen. Der hauptsächliche Unterschied liegt in der Verwendung der Funktion NPN_GetURL, die den Browser dazu auffordert, den angegebenen URL zu öffnen. In diesem Fall ist das die temporär erzeugte HTML-Seite. Der komplette URL wird in der Variable "url" erzeugt und zusammen mit dem Parameter This->target an den Browser geschickt.

```
    #ifdef XP_PC
    if (This->fHTML != 0) {
        close(This->fHTML);
        sprintf(url, "file://%s", This->fHTMLFileName);
        if (!NPN_GetURL((NPP)instance, url, This->target) ==
      NPERR_NO_ERROR)
        return NPERR_GENERIC_ERROR;
}
    #endif /* XP_PC */
```

Wenn man als Ziel für die HTML-Seite die Konstante "_self" angibt, wird die temporär erzeugte Seite in dem selben Fenster angezeigt, in dem das Plugin aktiv ist. Die HTML-Seite des Plugins wird also überschrieben, was ein sofortiges Beenden des Plugins zur Folge hat. Nachteil an dieser Methode ist, dass der Browser abstürzt. Wenn als Ziel ein anderer Frame angegeben wird, kann man das Problem umgehen.

3.3 Beispiel

Als Beispiel wird eine HTML-Seite konstruiert, die einen Parameter für das Rexx-Programm aufnehmen kann. Durch Click auf einen Button wird das Rexx-Script gestartet. Die Hauptseite enthält ein Frame (frame1) mit einem Button und einer Textbox sowie dem Plugin und einen leeren Frame (frame2), der als Ziel für die Ausgabe dient. Das Plugin wird durch den Button-Click erneut geladen und mit dem globalen Parameter "global" versorgt. Dieser wird auf der Hauptseite durch folgende Anweisung zwischengespeichert:

```
<script language="JavaScript">
<!–
var global = "test";
//–>
</script>
```

Anschliessend folgt die Definition des Frame-Sets:

```
<frameset rows="40%, 60%">
<frame src="frame1.html" name="frame1">
<frame src="frame2.html" name="frame2">
</frameset>
```

Auf der Seite frame1.html wird eine Funktion angelegt, die durch das OnClick-Ereignis des Buttons aufgerufen wird. Der Inhalt der Textbox wird in der Variable "global" gesichert, da nach der Anweisung "location.reload()" die Seite frame1.html neu aufgebaut wird, was zum Löschen des Textbox-Inhalts führt.

```
<script language="JavaScript">
<!–
function rexx_parameter()
{
    parent.global = document.rpi.rp.value;
    location.reload();
}
//–>
</script>
```

Das Plugin wird durch den OBJECT-Tag eingebettet. Auf die globale Variable wird mit "&{parent.global}; referenziert. Als Ziel der Ausgabe wird "frame2" angegeben.

```
<object data="test.rexx" type="application/x-rexx" height=0 width=0>
    <param name="target" value="frame2">
    <param name="input" value=&{parent.global};>
</object>
```

Hier sind die Textbox und der Button definiert:

```
<form name="rpi">
Rexx Parameter:
<input name="rp" type=text value=&{parent.global}; size=20 max-
length=20>
<input type=button value="exec rexx" onclick="rexx_parameter()">
</form>
```

Bei Betätigung des Buttons wird das folgende Rexx-Script gestartet:

```
say "<HTML><HEAD>"
say "<TITLE>foo bar</TITLE>"
say "</HEAD><BODY>"
say "<H1>rexx script executed at "
say date() " "
say time()
say "</H1>"
if arg() > 0 then do
    say "<p>parameter: "
    say arg(1)
    say "</p>"
end
say "</BODY></HTML> "
```

Das Script erzeugt HTML-Code, der die aktuelle Uhrzeit und Datum ausgibt und darunter den an das Script übergebenen Parameter anzeigt.

Der Einsatz von Javascript könnte entfallen, wenn man im OBJECT-Tag die Option "declare" benutzen könnte. Dadurch wird das Plugin nur deklariert, aber nicht ausgeführt. Zusätzlich zu "declare" muss die Option "id" angelegt werden, die dem Plugin einen eindeutigen Bezeichner zuordnet. Dieser Bezeichner wird im OnClick-Ereignis des Buttons angegeben. Bei Click auf den Button wird das Plugin gestartet. Als Parameter kann so der Inhalt der Textbox direkt an das Plugin übergeben werden und die JavaScript-Bestandteile fallen weg. Leider wird "declare", obwohl in [15] aufgeführt, vom Netscape Communicator nicht unterstützt.

4 Fazit

Die Auswahl der Plugin-Technologie in Kapitel1.5 basierte auf der verfügbaren Unterstützung von Plugins auf den unterschiedlichen Plattformen und Browsern. Der Forderung nach Plattformunabhängigkeit kommen Plugins nur in Bezug auf die Architektur des NPAPIs nach, es muss für jede Plattform eine eigene Bibliothek erzeugt werden. Es ist aber durchaus aktzeptabel, für jede Plattform eine einzelne Datei zur Verfügung zu stellen.

Enttäuschend ist die schlechte Unterstützung von Plugins durch Microsoft. Eine Dokumentation über die eingeschränkte Unterstützung von Plugins existiert nicht, man muss bei Problemen in Newsgroups nachforschen und anschliessend, wie aus der Beschreibung des Sourcecodes ersichtlich, entweder einen anderen Lösungsweg einschlagen oder für jede Plattform speziell angepassten Code schreiben.

Es existiert auch für den Browser Opera nur die Anmerkung, dass Plugins unterstützt werden. Es ist zu vermuten, dass die Unterstützung auch für die Linux-Version integriert wird, wenn sich Opera für Linux nicht mehr im Beta-Stadium befindet. Inwieweit Opera für Windows die Plugin-Technologie unterstützt, ist nicht untersucht worden.

Das Erzeugen einer graphischen Ausgabe im Plugin-Fenster ist durch die verschiedenen Archtekturen der GUIs sehr aufwändig. LiveConnect könnte eine Lösung für dieses Problem sein, da man mit Hilfe von LiveConnect graphische Elemente von Java in Plugins integrieren könnte. Allerdings ist auch hier die Unterstützung durch Microsoft nicht vorhanden und selbst unter Netscape Communicator 6 wird LiveConnect nicht mehr funktionieren.

Die Dokumentation der NPAPI wurde seit Januar 1998 nicht mehr aktualisiert. Das kann einerseits bedeuten, dass die Plugin-Technik soweit ausgereift ist, dass sie keiner Verbesserung bedarf. Andererseits ist es auch möglich, das Netscape eine Weiterentwicklung von Plugins zugunsten von anderen Technologien nicht vorsieht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Rexxla, the rexx language associaction. http://www.rexxla.org.

[2] Rfc2045, multipurpose internet mail extensions.

[3] Esker activex 4.1 plugin. http://www.esker.com, 2000.

[4] Microsoft Corp. Microsoft knowledge base search. http://search.support.microsoft.com/kb.

[5] Frank Holtry. The perlplus netscape plug-in. http://home.rmi.net/ fholtry.

[6] Microsoft Corp., http://www.microsoft.com/com/tech/activex.as. *ActiveX Controls.*

[7] Microsoft Corp., http://www.microsoft.com/com. *Component Object Model Specification.*

[8] Stefan Münz. *SELFHTML.* TeamOne, Kistlerhofstr. 111, D-81379 München, 7 edition, April 1998.

[9] Netscape Communications Corporation, http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm. *Plug-in SDK.*

[10] Netscape Communications Corporation, http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm. *Plug-in Guide*, devedge online documentation edition, January 1998.

[11] Opera Software, http://opera.nta.no/security/activex.htm. *ActiveX FAQ.*

[12] Frank Yellin Tim Lindholm. *The Java(TM) Virtual Machine Specification.* Sun Microsystems, Inc., http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html, 1997.

[13] Prof. Volker Turau. Techniken zur realisierung web-basierter anwendungen. *Informatik-Spektrum, Springer Verlag*, 1999.

[14] Billy Barron Mark Bishop Keith Brophy António Miguel Ferreira Edward Hooban Daniel I. Joshi Timothy Koets Bryan Morgan Rob McGregor Zan Oliphant Stig Erik Sando Dave Taylor Rick Tracewell Richard Wainess Greg Wiegand William F. (Bill) Anderson, Robert F. Breedlove. *Web Programming Unleashed*, chapter 1: An Overview of Internet Programming. Sams.net Publishing, Sams.net Publishing, 201 W. 103rd St., Indianapolis, IN46290, first edition, 1996.

[15] World Wide Web Consortium, http://www.w3.org/TR/1999/REC-html401-19991224. *HTML 4.01 Specification.* Chapter 13: Objects, Images, and Applets.

[16] TCL Developer Xchange. Tclplugin. http://dev.scriptics.com/software/plugin.