Seminar Winter Term 2000/2001

Plugin-Architecture of the Browsers
Netscape Communicator,
Microsoft Internet Explorer,
Opera

Michael Fischer

December 2000

Chair for Information Systems and Software Techniques
University of Essen

Contents

Abstract

WWW-Browsers allow the dynamical loading of applications (“plugins”) from third parties, e.g. Adobe, to get PDF-files from the browser and display it in the browsers window. Moreover, there exist serveral plugins for scripting languages like Perl or TCL in source code, as well as detailed documentation, which can be get from the internet. This makes it posible to use client-side scripting in addition to javascript.

Topic of this seminar paper is to weave-in into the plugin architecture of browsers, get to know existing plugins for scripting languages and the design and implementation of plugins for the scripting language Rexx.

1 Basics

Many informations from the internet require special applications in order to be viewed, played, listened or printed. Because of the further deployment of file formats, a browser should not only be able to load informations from the internet and possibly store them on a local memory medium. It should exist a way to handle these informations correctly by a simple mouse click. But it is not possible to implement all available file formats in a browser, on the one hand, because the great number of formats cannot be handled by a single application, on the other hand, because by developing new file formats the browser must permanently be replaced.

Alternativly to the enlargement of the browser, the information could be prepared on the server side and delivered as HTML-page. The browser is only required for input and output. Indeed this way increases both, the load of the network, as well as the load of the server, it is not possible to play video and audio files and the validation of user input requires additional data traffic on the network. There exist several proprietary extensions, from Microsoft as well as from Netscape, which makes it possible to show multimedia data with HTML (e.q. the tag <BGSOUND> [8]). But these are not part of the HTML standard [15].

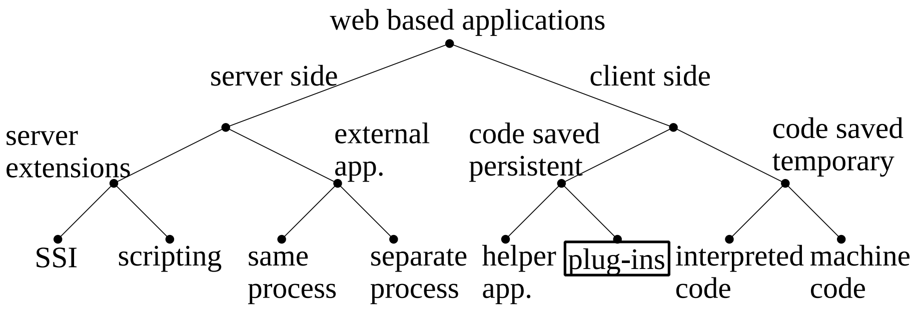

Figure 1: Classification of Web Based Applications [13]

Figure 1 outlines the possiblities of client-side and server-side extensions. The focus of this seminar paper lies on the client-side extension by using plugins. The term “plugin” is not exactly defined. The extract from [14] explains this term as follows:

"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"*

In the following, a short description of several ways is given, which expand the posibilties of viewing informations of the web browsers Netscape Navigator/Communicator , Microsoft Internet Explorer and Opera beyond HTML. In addition to Netscape's "plugins" and Microsoft's "ActiveX", there are also "Helper Applications" and "JavaApplets" presented to explain the term "plugin".

1.1 Helper Applications

Since the first version of Netscape Navigator and Opera, both browsers make it possible to assign a file format to a specific application. By clicking on a URL to a file, the assigned application is started and views the content of this file in accordance to the file format. The application "helps" the browser to process the file format.

Because of the high integration of the Internet Explorer in the operating system Microsoft Windows, there must not done any assignment between file formats and applications. This assignment is stored in the configuration of the operating system, the behavior of the Internet Explorer is the same as of Netscape Navigator.

Advantages of Helper Applications:

- In the beginning of the internet, the great advantage of helper applications was the easier handling of downloaded files by starting the assigned application by a single mouse click. Nowadays this advantage moved to the background because of new and smarter techniques.
- if processing of a file content is required, helper applications are still usefull.

Disadvatnages of Helper Applications :

- The whole file must first be downloaded before it can be processed (i.e.: no streaming of audio- or video-data).
- A special application must be installed for every file format to process the content of the downloaded file. Because there does not exist a viewer application for every file format (like Adobe Acrobat Reader or WordView), the providing of applications needs a high amount of ressources.
- The helper application is started in addition to the web browser. The downloaded information is not viewed in the context of the HTML-page, but in a separate window. This leads to an unclear representation of the information.

1.2 Netscape Plugins

The term ”plugin” is defined by Netscape as follows [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."*

With version 2 of the Netscape Navigator, the plugin technology was introduced. Unlike helper applications, plugins are not stand-alone applications but software modules, which are dynamically loaded on demand and enhance the functionality of the browser. They are build as dynamic link libraries (DLL) for microsoft windows respectivly as shared objects (SO) for Unix/Linux. For the operating systems MacOS and OS/2 there are comparable libraries build.

With version 3 of the Navigator, Netscape extends the plugin technology by LiveConnect. LiveConnect links the plugin technology with Java and JavaScript.

Since version 3 of the Internet Explorer the plugin technology of Netscape is supported by Microsoft with several restrictions. Theses Restricitons are partly described by Microsoft [4].

The browser Opera supports plugins since version 4.02, but only for Microsoft Windows.

Advantages of Netscape Plugins

- In contrast to Java Applets, Plugins are installed once and available until they are deleted.
- Plugins are written in C/C++ and therefore easy to build for several platforms.

Disadvantages of Netscape Plugins

- There exist no security concepts like the sandbox of Java. By using C/C++ the plugin developer is able to use many possiblities to manipulate client-side data.
- The compatibility to other platforms is made difficult by using system specific libraries. The graphical user interface must be developed especially for each operating system.

1.3 Java Applets

Also with Netscape Navigator 2 and Internet Explorer 3 Java-Applets are supported. A Java Applet is a program, that resides as precompiled bytecode on the webserver and is downloaded by the browser. On execution, the bytecode is interpreted by the Java-Virtual-Machine [12], which is integrated in the browser. In contrast to plugins, applets are downloaded and executed by every call of the HTML-page.

The browser Opera supports Java since version 4.02, but only for Microsoft Windows .

Advantages of Java Applets

- security by using the sandbox concept, no possibility to access files or execute external programs
- platform independent
- providing a uniform graphical user interface on all supported operating systems

Disadvantages of Java Applets

- Java bytecode must be interpreted, therefore Java is slower than e.g. Plugins or ActiveX
- class libraries, which are required by a Java Applet but not available locally must be downloaded in addition to the applet on the first access to the HTML-page. This increases the download time.

1.4 ActiveX-Controls

With Version 3 of Internet Explorer, Microsoft introduced the ActiveX technology. ActiveX [6] is based on the Component Object Model [7], an interface technology, which theoretically could be used on every operating system. But COM, and therefore also ActiveX, is only available on Windows operating systems. In Addition, ActiveX works only with the Internet Explorer.

To make ActiveX-Controls available to Netscape's Browser and Opera, there exist serveral Plugins, e.g. from Esker[3],which gets a link to the required ActiveX-control. But the HTML-Code must be modified with JavaScript and the use of ActiveX-Controls is still restricted to Windows operating systems.

Under [11] exists a FAQ-list which explains, why ActiveX-Controls never be supported by Opera. The main reasons are the dependence on the operating system and in the lack of security of ActiveX-Controls.

Similar to plugins a ActiveX-Control has to be downloaded once and is available until it is deleted. An AktiveX-Control is system-wide available and has access to all system ressources. Therefore there exists no security concept like e.g. in JavaApplets. On the other hand, ActiveX-Controls are available for many programming languages for application development (e.g. MS Visual C++, Borland Delphi).

Advantages of ActiveX-Controls

- possiblity to use many operating system specific features
- there exists a great number of available ActiveX-Controls
- high performance on execution

Disadvantages of ActiveX-Controls

- no security concept
- platform-dependent

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

Table 1: Availability of Java Applets, ActiveX and Netscape Plugins on MS Windows and Linux

1.5 Selection of a suitable technique to develope a Rexx-Plugin

Table 1 shows the availability of the former described plugin technologies by using the possible browser- and operating-system combinations. Helper Applications are not listed here because of the lack of needed integration in HTML-pages. An additional criterion is to execute a external program. Because the script delivered by the webserver must be interpreted by the Rexx interpreter, it must be possible to execute this interpreter. Because of the sandbox-technique, it is not possible to execute external programs from Java Applets.

Also ActiveX is not suitable because of its platform-dependence. Indeed it is possible to execute ActiveX-Controls under Netscape Navigator, but there does not exist a way to execute ActiveX-Controls on other operating system than Microsoft Windows.

In the following the plugin technique of Netscape is described. The Foundation is the documentation of “Netscape Developer Plugin-Guide” [10] and the “Netscape Plugin Software Development Kit”[9].

2 Netscape Plugins in Detail

2.1 Implementation in HTML

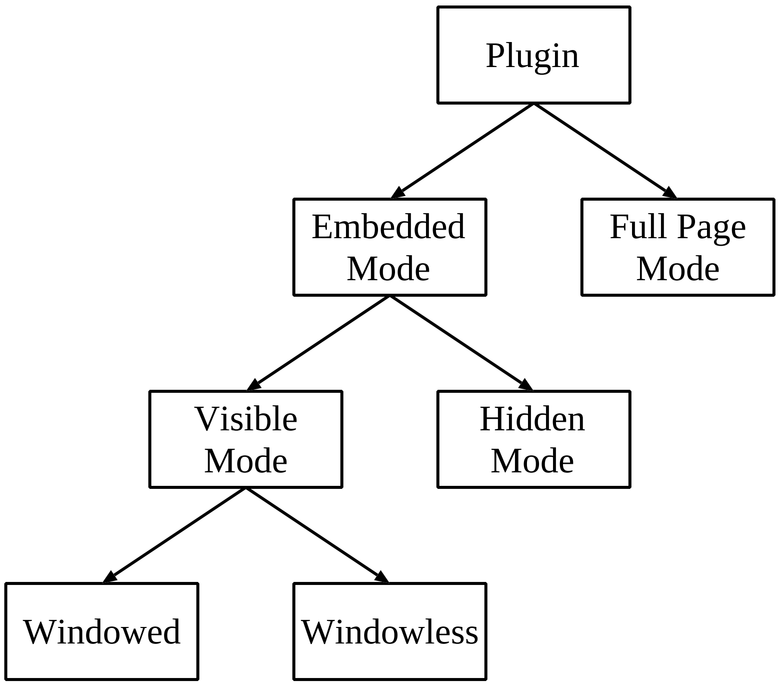

Figure 2: Display Modes of Plugins

Figure 2 shows the different display modes of a plugin. Basically the embedded mode and the full page mode is distinguished.

2.1.1 Full Page Mode

By giving the HTML-Statement

<A HREF="myavi.avi">play myavi</A>

a link with the label "play myavi" is displayed. A click on this link creates a browser window, which is fullfilled with the plugin window. Alternativly the file myavi.avi can be opended by the browsers file-open menu. It is not possible to pass any aditional parameters to the plugin, when working in full page mode.

2.1.2 Embedded Mode

To embbed plugins in HTML-pages there are two tags. The first, EMBED-Tag, is not part of the HTML-specification [15], but is supported by all browsers.

EMBED-Tag The embedded mode is created by the HTML-Tag <EMBED>. The plugin is invoked at the place of the EMBED-definition in the HTML-page, in the size given by the parameters WIDTH and HEIGHT. The option SRC is the URL of the file to display. If a plugin has to be started, which does not need any data (e.g. a plugin to show the current time and date), there must no option SRC be provided, but the MIME-type must be specified via the TYPE-option.

Every plugin has a window for the graphical output. This window is integrated in the HTML-page, the position is specified by the option ALIGN and by the position of EMBED-Tag. If the option HIDDEN is part of the EMBED-tag, no window is displayed. In that case there must no additional information about the width and height of the plugin be given.

Table 2 lists the more important options of the EMBED-tag: Additional to

Option	Description
SRC	URL of the file to display
TYPE	MIME-Type of Plugin
ALIGN	Alignment of Plugin: Left, Right, Top, Bottom
BORDER	Widht of Frame for Plugin-Window
FRAMEBORDER	“NO”, if no border neccessary
HEIGHT	Height of Plugin-Window
WIDTH	Width of Plugin-Window
HIDDEN	Plugin without graphical output
HSPACE	Horicontal space around the plugin
VSPACE	Vertical space around the plugin

Table 2: EMBED-Tag

these options there can be given more options, which are ignored by the browser and passed to the plugin. The HTML-statement

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

loads the plugin, which is assigned to the filetype “.avi” and creates a window with the size 100x100. The option LOOP is not meaningful for the browser and ignored. All four options are passed to the plugin, where they are analyzed.

OBJECT-Tag In contrast to the EMBED-Tag, which only embeds plugins, the OBJECT-tag is able to embed also Applets, ActiveX-controls, images or any other ressources, which are outside of a HTML-page and should be embedded. As a general object reference it prevents from creating new tags for every new ressource, which must be supported by HTML in the future. For example, the introduction of the OBJECT-tag obsolets the APPLET-tag.

OBJECT-Tag Option	Description	EMBED-Tag Option
DATA	URL of the file	SRC
TYPE	MIME-Type of Plugins	TYPE
ALIGN	Alignment of Plugins	ALIGN
HEIGHT	Height of Plugin-Window	HEIGHT
WIDTH	Width of Plugin-Window	WIDTH

Table 3: OBJECT-Tag

Table 3 lists the important options of the OBJECT-tag together with the corresponding options of the EMBED-tag. For passing parameters to the plugin, one must use the PARAM-tag within the OBJECT-tag. The former example of the EMBED-tag is defined as OBJECT-tag as follows:

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

Further documentation is given by [15] and [8].

2.1.3 Visible/Hidden Plugins

The display mode "Visible" is the standard mode for embedded plugins and is turned off by specifying the option HIDDEN in EMBED-tags. Because there is no corresponding option to HIDDEN in the OBJECT-tag, the WIDTH and HEIGHT is set to zero. A hidden plugin has no graphical output window and claims no space in the HTML-page.

2.1.4 Windowed / Windowless Plugins

The display mode "Windowless" was introduced with Netscape Communicator 4. In contrast to windowed plugins, which have an defined output range, windowless plugins can access any drawing context, which is provided by a window-handle. Windowless plugins are not supported by the X-Windows system, therefore the explanation of windowless plugins does not appear here.

2.2 Overview of NPAPI-functions

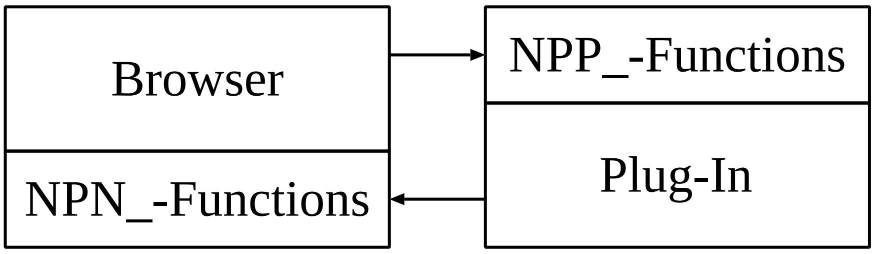

Figure 3: Interface between Browser and Plugin

The interface between browsers and plugins is defined in the Netscape Plugin Application Programming Interface (NPAPI) and divided into two main parts (Figure 3). Functions, which are called from the browser and provided by the plugin are named with the prefix "NPP_". Functions, which are called by the plugin and provided by the browser are named with the prefix "NPN_".

Further, the more important functions are listed. Detailed explanations to the parameters can be read in [10] . A practical example follows in Section 3.

Table 4 lists the more important functions, which are provided by the plugin and called by the browser. These functions are the basic functionality of the plugin and must be implemented, otherwise an error messages occurs by calling the plugin. Table 5 lists the more important functions, which the browser provides for the plugin.

2.3 program run by calling a HTML-page

By starting the browser, all available plugins are registered. Available plugins reside in the subdirectory "plugins" of the browsers program directory. Under Unix a additional directory "$(HOME)/.netscape/plugins" is searched by the Netscape browser. To register the supported MIME-types, two different strategies are used:

Function	Description
NPP_Initialize	Provides global initialization for a plug-in
NPP_Shutdown	Provides global deinitialization for a plug-in
NPP_New	Creates a new instance of a plug-in
NPP_Destroy	Deletes a specific instance of a plug-in
NPP_NewStream	Notifies a plug-in instance of a new data stream
NPP_WriteReady	Determines maximum number of bytes that the plug-in can consume
NPP_Write	Delivers data to a plug-in instance
NPP_DestroyStream	Tells the plug-in that a stream is about to be closed or destroyed
NPP_SetWindow	Tells the plug-in when a window is created, moved, sized, or destroyed
NPP_GetMIMEDescription	MIME-Type of the Plugins
NPP_GetValue	Allows Communicator to query the plug-in for information
NPP_GetJavaClass	Returns the Java class associated with the plug-in
NPP_Print	Requests a platform specific print operation
NPP_URLNotify	Notifies the instance of the completion of a URL request
NPP_StreamAsFile	Provides a local file name for the data from a stream

Table 4: Functions provided by the plugin

Function	Description
NPN_MemAlloc	Allocates memory from Communicator's memory space
NPN_MemFree	Deallocates a block of allocated memory
NPN_GetUrl	Requests Communicator to create a stream for the specified URL
NPN_NewStream	Requests the creation of a new data stream
NPN_Write	Pushes data into a stream produced by the plug-in
NPN_DestroyStream	Closes and deletes a stream

Table 5: Functions provided by the browser

Windows The information about the MIME-type is stored in the DLL. For this, a ressource-file containing version information must be created, which is linked to the DLL. By registering the plugin the browser reads this version information, which contain the MIME-type.

Unix Plugins under Unix must have the NPAPI-functions NPP_GetMIMEDescription and NPP_GetValue, which are called by the browser. NPP_GetMIMEDescription sends back a string like "MIME-Type:File Extension:Description". NPP_GetValue is used to get the name of the Plugin and a description text.

A list of all available plugins can be get by selecting the menu item "help->about plugins" in Netscape Browsers. Opera has a plugin dialog under "file->preferences". Microsoft Internet Explorer does not give any possibility to query the installed plugins. One can write a simple JavaScript program, which uses the methods of the object "navigator". This object is also provided by MS IE. The following example is from [8]:

```
<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>
```

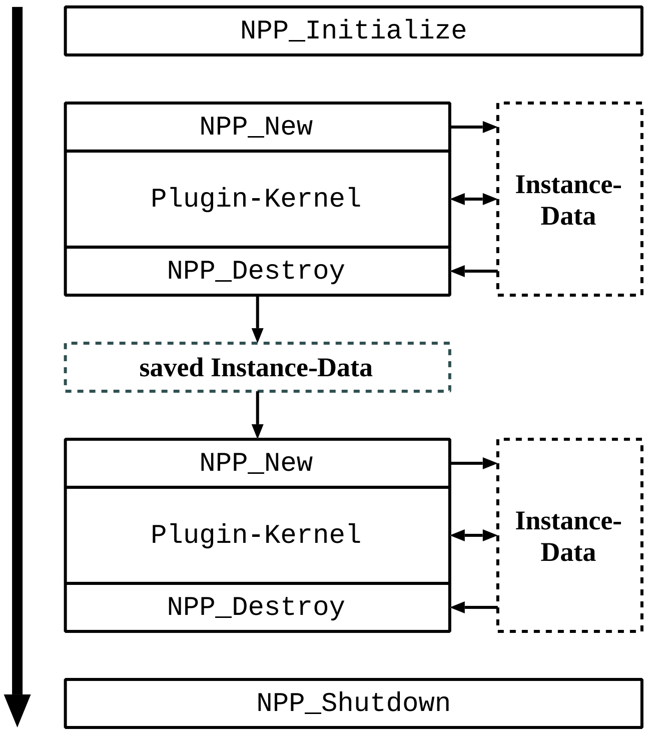

Figure 4: program run of a plugin

Figure 4 shows a coarse scheme of initializing and instanciating a plugin, which is called twice in a HTML-page.

NPP_Initialize is executed at the first call of the HTML-page, NPP_Shutdown is executed by closing the page, after the last instance of the plugin is destroyed.

After NPP_Initialize the function NPP_New is executed for every plugin instance. For example, in the case, that a plugin is called two times on the same page, the function NPP_New is called twice by displaying this page. This is also be done, if a second browser window is invoked.

The instance data is created in NPP_New by each plugin instance and is removed in NPP_Destroy. An instance can pass its instance data to the following instance by allocating memory and passing a pointer to that memory to the browser.

A plugin, which is instanciated by calling the same URL, gets this pointer with the call to NPP_New as parameter and can access the instance data of the former instance.

One part of the plugin kernel is the refreshing of the plugins window by calling NPP_SetWindow and the data exchange with the webserver. By getting data from the webserver, the plugin-function NPP_NewStream is called by the browser. Here can be decided, in which way the data has to be handled. In general there is the stream-oriented and the file-oriented way of processing data.

In stream-oriented mode the functions NPP_WriteReady and NPP_Write are called until all data from the webserver is sent to the plugin and the stream is closed by calling NPP_DestroyStream. With NPP_WriteReady the amount of data, which is passed to NPP_Write, is determined by the plugin. This amount of data is send to the plugin by calling a following NPP_Write. The data stream can be interrupted at any time by the plugin by calling NPN_DestroyStream.

In file-oriented mode, the browser creates a file, which stores all the data sent by the webserver. Then the browser calls NPP_StreamAsFile and gives the name of the local file to the plugin. The processing and deleting of this file must be done in the plugin.

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

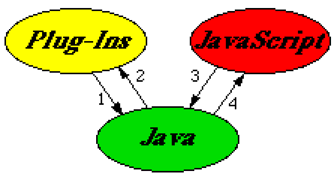

Figure 5: LiveConnect [10]

LiveConnect was introduced with Netscape Navigator 4. LiveConnect integrates plugins, Java and JavaScript. It is possible, to access java objects from a plugin and vice versa, to access plugin functions from a java applet. Figure 5 shows how to indirectly get access from JavaScript to plugins.

LiveConnect needs a modified Java Virtual Machine [12], which is only implemented in Netscape Browsers. Therefore LiveConnect is not supported on Opera or Internet Explorer. And even in the Version 6 of Netscape Communicator is no modified JVM installed, which in the long run leads to the end of LiveConnect.

2.5 Plugin Template

As a starting point to develope plugins, the Plugin Software Development Kit provides a Plugin Template, in which all necessary functions are predefined. The plugin developer has only to fill these functions with some code. Figure 6 shows the division of the different files of the template. For simplification, files of the LiveConnect-implementation are not shown.

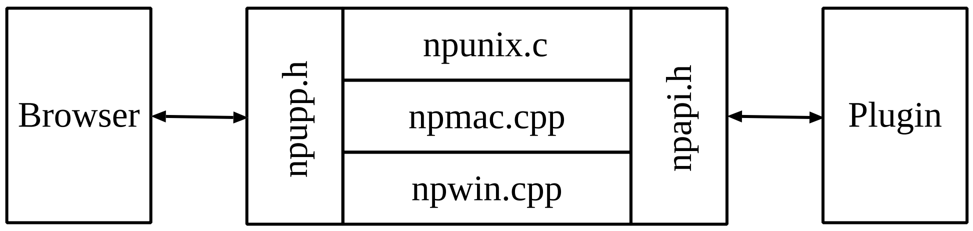

Figure 6: Plugin Template

Because of the platform dependence of plugin code, there are different source files for every supported platform, which are adapted to the conventions of a specific operating system. To create a plugin library, the appropriate file must be used, e.g., npunix.c, to build a shared object for linux/unix.

The header file npupp.h contains the function prototypes and function table, which the browser needs to call the plugin functions. The initialization and the wrapper functions for the functions specified in the table are defined in the three source files npunix.c, npmac.cpp and npwin.cpp.

The most important file for the plugin developer is npapi.h, because here are all function prototypes and structures implemented, which are used to create plugin sourcecode.

By setting the compiler options XP_UNIX, XP_PC or XP_MAC, the appropriate source files for the target platform are compiled. Additional, the compiler option PLUGIN_TRACE kann be defined, to trace every function call to the plugin by generating messages to the standard error queue (stderr). By this way it is possible to test the program run described in 2.3.

3 Developing a Rexx-Plugin

As a conceptual foundation for the rexx plugin there are the implementations of the scripting languages TCL [16] and Perl [5] as Netscape Plugins. Both are based on the fact, that the interpreter is already installed on the client machine and can be executed. The scripting source is passed as stream to the plugin and stored in a temporary file. The filename is delivered as parameter to the interpreter. The <SCRIPT>-tag is not considered in the Rexx-Plugin.

Further informations about the scripting language Rexx are given by [1].

3.1 Requirements

A plugin on the foundation of the Netscape Plugin SDK hast to be developed. On calling an URL, which points to a rexx scripting file with the extension ".rex", the content of this file is delivered to the client-side installed rexx interpreter and is executed. The rexx script creates HTML code, which is displayed by the browser. The plugin becomes a parameter, which is the target of the HTML output. Additional a second parameter is provided, which is delivered to the rexx interpreter. There is no need for a plugin window, because a graphical output is not created. The MIME type (Multipurpose Internet Mail Extension) [2] is defined as "application/x-rexx". The data is received as stream and stored in a local temporary file. In summary, the Plugin will have the following attributes according to the description in chapter 2.1:

- Embedded Mode
- Hidden
- stream-oriented

The coarse run of the plugin is shown in figure 7.

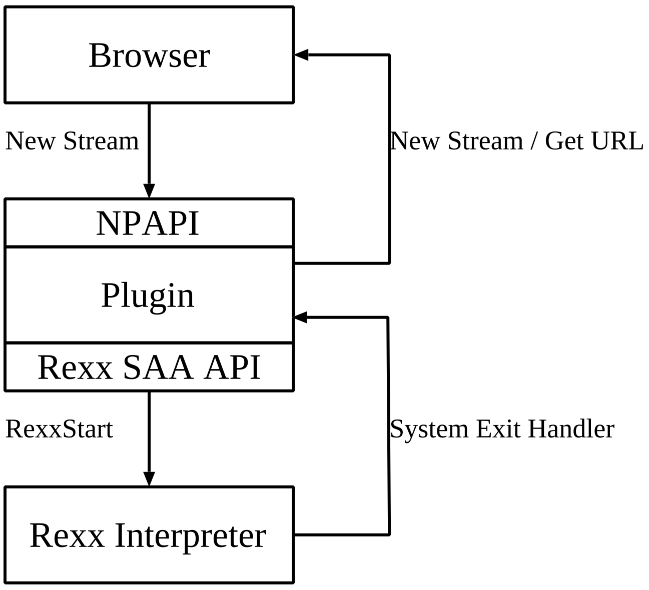

Figure 7: Program run of the Rexx Plugin

3.2 Implementation

In the following are only these "NPP_"-functions introduced, which have a concrete task within the plugin kernel. All other functions listed in table 4, which are not shown here, must even be implemented to build a correct working plugin. These functions reside empty in the plugin source.

Fieldname	Data Type	Description
Window	NPWindow*	Handle of the plugin window
tmpFile	int	Handle of the temporary scripting file
tmpFileName	char[L_tmpnam]	Name of the temporary scripting file
target	char*	Name of the target for HTML output
input	char*	Parameter for Rexx
rexx_stream	NPStream	Pointer to the HTML data stream
fHTML	int	Handle of HTML-File (only Windows)
fHTMLFileName	[L_tmpnam]	Name of HTML-File (only Windows)

Table 6: Instance Data Structure

3.2.1 NPP_New

For every instance, memory is allocated, that holds the instance data. The structure of the instance data is defined by the plugin developer. The structure of the rexx plugin is described in table 6. The NPP_New function gets an uninitialized pointer instance->pdata. All in the following called functions get a pointer as parameter to the instance data.

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

Memory for plugins is always allocated by using the function NPN_MemAlloc instead of malloc, because the browser does its own memory management.

The next step is to analyze the parameters, which were deliverd by the HTML tag. argn[] contains the names, argv[] the values and argc the number of parameters.

```
    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]);
}
```

The function tmpnam, which is part of the ANSI-C-library, creates an unique filename. With this filename a file is created, in which the rexx script is stored.

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

3.2.2 NPP_Destroy

Each function, which must have access to the instance data, makes a reference to these by executing the statement

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

Therefore, this line is at the beginning of almost every function. In the following, the variable "This" is allways a pointer to the instance data.

By calling NPP_Destroy, the browser requests the plugin to erase all data and files, which were created by the plugin instance. NPN_MemFree() is equivalent to free(). unlink closes and erases the temporary file.

```
    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 signals, that the browser wants to send a data stream to the plugin. One can refuse the data stream by returning a value not equal NPP_NO_ERROR.

3.2.4 NPP_WriteReady

As the return value the browser expects the maximum size of data the plugin can handle at this time. For the Rexx-Plugin this amount is defined as STREAMBUFSIZE=4096. It is possible to insert a algorithm, which calculates the optimal amount of data to handle.

```
return STREAMBUFSIZE;
```

After every call to NPP_WriteReady a data paket is sent to the function NPP_Write with a maximum number of STREAMBUFSIZE bytes. This process is repeated as often as data is available or the plugin calls NPN_DestroyStream to interrupt the data stream.

3.2.5 NPP_Write

The data received in buffer are written to the temporary file by the write-statement. A maximum amount of STREAMBUFSIZE bytes is written. The actual processed amount of data is given back to the browser as return value.

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

With this return value the browser calculates the next data paket to send. If, for example, instead of 4096 bytes only 2048 bytes are written, the browser again sends first the 2048 bytes, which were not processed by the plugin.

3.2.6 NPP_DestroyStream

After all data has been sent to the plugin, the function NPP_DestroyStream is called by the browser. The temporary file has to be closed and the Rexx interpreter is started. The if-Statement is going to be explained later in context with 3.2.9.

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

3.2.7 rexx_exec

To control a running Rexx script, one can set so called "hooks", which interrupt the program run and call "hooked" function. Such hooks are realized by System Exit Handlers. System Exit Handlers are functions from the Rexx SAA (System Architecture Application) API, a interface between Rexx and other programming languages, e.g. C. Further information can be get at [1].

To catch the output of the Rexx-command "SAY", the function hook_RXSIOSAY has to be registered by the API-function RexxRegisterExitExe with the identical name "hook_RXSIOSAY". As a additional parameter, a pointer to the instance data of the plugin is delivered.

```
    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);
```

After that, the Rexx interpreter is called by using the API-function RexxStart. As Paramter, this function gets the name of the temporary scripting file and an input parameter, which was determined in NPP_New, and the array sysexit, which holds the names of all System Exit Handlers. In this case it is only "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);
```

After the script is executed, the hook is going to be deleted.

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

Alternativly, the hook could be registered in NPP_New oder NPP_Initialize .

3.2.8 hook_RXSIOSAY

Before the Rexx command "SAY" is executed by the interpreter, all may registered hooks are called with some information about the type of call. The call is clearly defined by a exit number and a subfunction number. The Rexx plugin catches all standard I/O-calls (RXSIO) and from these only the SAY-command (RXSIOSAY). For all other System Exits, the return value RXEXIT_NOT_HANDLED is given back. By this return value, the interpreter continues with the acutal command. If the return value is RX_EXIT_HANDLED, the interpreter skips the execution of this command.

```
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

To hold the plugins instance data, there must be memory allocated for a pointer.

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

The memory is filled by the API-function RexxQueryExit with a pointer, which was delivered by the function RexxRegisterExitExe. So it is possible to get access to the instance data similar to the "NPP_"-functions.

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

At this point, there are differences between the plugin code for Linux and the Windows plugin. It is not possible for the Microsoft Internet Explorer to execute the NPN_NewStream. It seems, that this function is not implemented by Microsoft.

LINUX: The output of the Rexx program is delivered as stream to the browser. To create a stream from a plugin to the browser, the function NPN_NewStream is called once on the first call of the Exit Handler. After the first call of NPN_NewStream, the variable This->rexx_stream is initialized, so no second call to NPN_NewStream is done. The parameter This->target contains the target of the output. This is either the name of a frame of the actual HTML page or one of the constants _blank/_new for a new browser window, _parent or _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;
   }
}
```

For every call of the Exit Handler the output of the Rexx program is given to NPN_Write. ParmBlock is a pointer to data for this Exit Handler. In case of the Exit Handler RXSIO:RXSIOSAY it is a pointer to a RXSTRING-structure, which holds the output of the Rexx program.

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

The return value RXEXIT_HANDLED shows the interpreter, that the SAY-command is successfull executed.

The following code is from the function rexx_exec. After removing the Exit Handler by RexxDeRegisterExe, the data stream to the browser has to be closed by NPN_DestroyStream.

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

WINDOWS: Because the Internet Explorer does not accept streams, a temporary file must be created, which stores the output of the SAY-command. Even like the NPN_NewStream, the creation of this file is done once.

```
    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;
      }
  }
```

The output of the SAY-command is written in the file.

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

And even in rexx_exec the file has to be closed. The main difference is the usage of NPN_GetURL, which requests the browser to open the given URL. In that case, it is the temporary created HTML-page. The complete URL is created in the variable “url” and is delivered to the browser together with the paramter This->target.

```
    #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 */
```

If one specifies “_self” as target, the temporary created page is shown in the same window as the plugin. The great disadvantage of this method is the crash of the browser. If one specifies a different target, this problem is bypassed.

3.3 Example

As example a HTML-page is created, which can get a parameter for the Rexx program. The main page contains a frame with a button and a textbox, the plugin and a second empty frame, which is the target for all output operations of the Rexx program. The plugin can be started by pushing the button on frame 1 and gets the global variable “global”. This variable is declared on the main page by the following statements:

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

This is the definition of the frameset:

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

The page frame1.html contains a function, which is executed by the OnClick-event of the button. The content of the textbox is stored in the variable “global”, because after executing “location.reload()” the page frame1.html is going to be refreshed, which leads to erasing the textbox content.

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

The plugin is embedded by the OBJECT-Tag . The global variable is referenced by “&{parent.global};”. The target for the output is set with “frame2”.

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

Here is the definition of the textbox and button:

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

By pushing the button, the following Rexx script is executed:

```
    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> "
```

This script creates HTML-code, which displays the actual date and time and below the parameter, given by the textbox.

The use of JavaScript would not be neccessary, if the option “declare” of the OBJECT-tag would work. With that, the plugin is only declared and not executed. In Addition to “declare”, the option “id” must be defined to give a unique identifier to the plugin. This identifier is used in the OnClick-event. By pushing the button, the plugin is started. The content of the textbox can directly be delivered to the plugin, so no JavaScript is needed. Unfortunatly the “delcare” option is not supported by Netscape Communicator, even it is a part of the official HTML specification [15] .

4 Conclusion

The selection of the plugin technology in chapter 1.5 was based on the available support of plugins for different platforms and browsers. The demand for platform independence is fullfilled by plugins only in the architecture of the NPAPI. For every platform a different library has to be created. But it is surely acceptable to create a single file for every platform.

The support of plugins by Microsoft is disappointing. A documentation of the restricted support does not exist, one must figure out Problems by e.g. reading newsgroups and after that, as shown in the source code of the Rexx plugin, either find another way of solution or write platform-specific code.

Even for the browser Opera exists only a remark, that plugins are supported. May it is possible, that the plugin support for opera is integrated for the Linux version, if the beta stadium has ended. It was not investigated, how the plugin technology is supported by Opera for Windows.

Because of the different architectures of the graphical user interfaces, a graphical output in the plugin window is very extensive. A solution for that could be Live-Connect, because it is possible to integrate graphical elements of Java in plugins by using LiveConnect. But the support of LiveConnect by Windows does not exist und even Netscape stopps supporting LiveConnect with Netscape Communicator 6.

Since january 1998 the documentation of NPAPI was not refreshed. On the one hand, this could mean, the plugin technology is nearly perfect. On the other hand, it is possible, that Netscape has no further interests in developing the plugin technology because of other, maybe smarter technolgies.

List of Figures

List of Tables

References

[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.