Client Extension Mechanisms

(helpers & plugins)

Plug Ins

INTERNET ENGINEERING

- A plug-in is a separate code module that behaves as though it is part of the Browser.
- You use plug-ins that extend Browser with a wide range of interactive and multimedia capabilities, and that handle one or more data (MIME) types.

Demo

INTERNET ENGINEERING

Few Examples

Design Goals

INTERNET ENGINEERING

- To extend the capabilities of Browser by providing inline viewers for types of data not supported by Communicator itself.
- provide an API that is as simple and concise as possible, making it relatively easy to leverage existing native code libraries or convert existing applications to take advantage of the web.
- Plug-ins can use the Java Runtime Interface (JRI) to access Java. Communication with JavaScript, takes place through a LiveConnect connection.

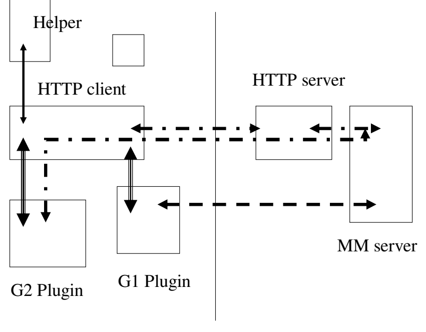
Plug Ins: Design Goals

INTERNET ENGINEERING

- With the Plug-in API, you can create dynamically loaded plug-ins that can:
  - register one or more MIME types
  - draw into a part of a Communicator window
  - receive keyboard and mouse events
  - obtain data from the network using URLs
  - post data to URLs
  - add hyperlinks or hotspots to link to new URLs
  - draw into sections on an HTML page
  - see the plugins: about:plugins

Models for Extension

INTERNET ENGINEERING



When to Design a Plugin?

INTERNET ENGINEERING

- Plug-ins offer solutions to development needs like these:
  - tying existing C++ code Browser platform
  - writing performance-sensitive code
  - taking advantage of specific operating system capabilities
  - integrating native or legacy code bases into Internet/intranet applications
  - using native methods to export low-level functionality that Java does not provide
- Because plug-ins are platform-specific, you must port them to every operating system and processor platform upon which you want to deploy your plug-in.
- Plug-ins are supported in both Netscape Clients and Servers.

Plug Ins and Helper Apps

INTERNET ENGINEERING

- Before Netscape first presented plug-ins in Navigator 2.0, in the first quarter of 1996, users could extend the Navigator with helper applications.
- A Helper application is:
  - a separate, free-standing application started from the Browser.
  - Like a plug-in, the browser starts a helper application when it encounters a MIME type that is mapped to it.
  - Unlike a plug-in, a helper application runs separately from the browser in its own application space and does not interact with the browser or the web.
  - A Browser always searches for a registered plug-in first. If there are no matches for the MIME type, it looks for a helper.

Execution Model

INTERNET ENGINEERING

- When Communicator starts, it checks for plug-in modules:
  - in the plugins directory (Windows)
  - Plug-ins folder (Mac OS) in the same folder or directory as the Communicator application.
  - On Unix, Communicator checks the path set in the environment variable NPX_PLUGIN_PATH.
- When the user opens a page that contains embedded data of a media type that invokes a plug-in, Communicator responds with the following sequence of actions:
  - check for a plug-in with a matching MIME type
  - load the plug-in code into memory
  - initialize the plug-in
  - create a new instance of the plug-in

Execution Model (contd.)

INTERNET ENGINEERING

- Communicator can load multiple instances of the same plug-in on a single page, or in several open windows at the same time.
- When the user leaves the page or closes the window, the plug-in instance is deleted.
- When the last instance of a plug-in is deleted, the plug-in code is unloaded from memory.
- A plug-in consumes no resources other than disk space when it is not loaded.

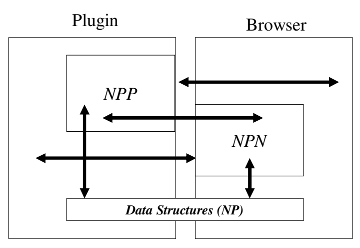
Plug-in Architecture

INTERNET ENGINEERING

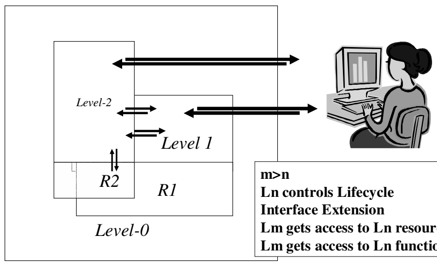
- The Plug-in Application Programming Interface (API) is made up of two groups of functions and a set of shared data structures.
  - Plug-in methods are functions that you implement in the plug-in; Communicator calls these functions. The names of all plug-in functions begin with NPP_, for example, NPP_New.
  - Netscape methods are functions implemented by Communicator; the plug-in calls these functions. The names of all Netscape functions begin with NPN_, for example, NPN_Write.
- Data structures are plug-in-specific types defined for use in the Plug-in API. The names of structures begin with NP, for example, NPWindow.

Plug-in Architecture

INTERNET ENGINEERING



Extensible Software Architecture



m>n
Ln controls Lifecycle
Interface Extension
Lm gets access to Ln resources
Lm gets access to Ln functions

Operations

- How to begin and End Plugins
- Drawing and Event Handling
- Streaming
- Accessing URLs
- Sharing Memory Space Efficiently
- Some Utilities

Initialization

- A set of methods used by the Browser provide the basic processes of initialization, instance creation and destruction, and shutdown.
- Initialization:
  - Communicator calls the Plug-in API function **NPP_Initialize** when the plug-in code is first loaded.
- Instance Creation:
  - Communicator calls the Plug-in API function **NPP_New** when the instance is created.

Destruction

- Instance Destruction:
  - The plug-in instance is deleted when the user leaves the instance page or closes the instance window;
  - Communicator calls the function **NPP_Destroy** to tell the plug-in that the instance is being deleted.
- Shutdown:
  - When the last instance of a plug-in is deleted, the plug-in code is unloaded from memory and Communicator calls the function **NPP_Shutdown**.
  - After shut-down Plug-ins should consume no resources other than disk space.

Handling Memory

- Plug-ins share memory space with the Browser, they can take advantage of any customized memory-allocation scheme the Browser has. It can call the following Browser functions:
  - NPN_MemAlloc method to allocate memory from Netscape Communicator.
  - NPN_MemFree method to free memory allocated with NPN_MemAlloc.
  - NPN_MemFlush method to free memory (Mac OS only) before calling memory-intensive Mac Toolbox calls.
- In addition, the plug-in usually has the option of using its own memory functions.

Drawing and Event Handling

- When it comes to determining the way a plug-in appears in a web page, you (the plug-in designer) and the web page author have many options.
- The content provider determines its 'display mode':
  - embedded in a web page and visible.
  - embedded in a web page but hidden.
  - or displayed in its own separate page.
- Plug-in developer determine whether a plug-in is windowed or windowless:
  - A windowed plug-in
  - A windowless plug-in

Checkout the HTML markers EMBED, & OBJECT tags.

- When a plug-in is loaded, it is drawn into a target area. This target is either the windowed plug-in's native window, or the drawable of a windowless plug-in

..

- The NPWindow structure represents either the native window or a drawable. This structure contains information about coordinate position, size, the state of the plug-in (windowed or windowless), and some platform-specific information

Windowed Plug-in

INTERNET ENGINEERING

- A windowed plug-in draws into its own native window (or portion of a native window) on a web page.
- A windowed plug-in is opaque, hiding the part of the page beneath its display window, and can be invoked in the top HTML layer of a page.
- This type of plug-in determines when it draws itself.

Windowless Plug-in

INTERNET ENGINEERING

- A windowless plug-in does not require a native window.
- It is drawn in a target called a drawable, which corresponds to either the Communicator window or an off-screen bitmap.
- A drawable can be defined in several ways, depending on the platform.
- Windowless plug-ins can be opaque or transparent.
- A windowless plug-in draws itself only in response to a paint message from Communicator.
- NOTE: Whether a plug-in is windowed or windowless is not meaningful if it is invoked with the HIDDEN attribute.

Drawing in Windowless Plugin

INTERNET ENGINEERING

- Before drawing itself on the page, the plug-in must provide information about itself, set the window or other target in which it draws, arrange for redrawing, and handle events.
- A windowless plug-in can call the following Browser methods to draw itself:
  - NPN_ForceRedraw: Force a paint message for windowless plug-ins.
  - NPN_InvalidateRect: Invalidate an area in a windowless plug-in before repainting or refreshing.
  - NPN_InvalidateRegion: Invalidate an area in a windowless plug-in before repainting or refreshing.

Drawing and Event Handling

INTERNET ENGINEERING

- Communicator calls these Plug-in methods:
  - NPP_SetWindow: Set the window in which a plug-in draws.
  - NPP_HandleEvent: Deliver a platform-specific event. The plug-in must return true if it has handled the event and false if it has not.
  - NPP_Print: Request a platform-specific print operation.
  - NPP_GetValue: Query the plug-in for information.
  - NPP_SetValue: Set Communicator information.
- The plug-in can call these Netscape methods:
  - NPN_GetValue: Get Communicator information.
  - NPN_SetValue: Set plug-in Communicator information.

URLs

INTERNET ENGINEERING

- Uniform resource locator (URL) protocols provide a means for locating and accessing resources that are available on the Internet and on intranets.
- Plug-ins can request and receive the data associated with URLs of any type that the browser can handle, including HTTP, FTP, news, mailto, and gopher.
- Get URL
- Post URL

Getting URLs

INTERNET ENGINEERING

- To retrieve a URL and display it on a specified target page, plugins can use the following functions:
- NPN_GetURL:
  - The plug-in uses the function to ask Communicator to display data retrieved from a URL in a specified target window or frame, or deliver it to the plug-in instance in a new stream. This is the way that plug-ins provide hyperlinks to other documents or retrieve data from the network.
  - NPN_GetURL is typically asynchronous: it returns immediately and only later handles the request, such as displaying the URL or creating the stream for the instance and writing the data.

Get URLs (contd..)

- If the target parameter is set to null, the application creates a new stream and delivers the data to the plug-in instance, through calls to NPP_NewStream, NPP_WriteReady and NPP_Write, and NPP_DestroyStream.

- NPN_GetURLNotify:
  - If Communicator cannot locate the URL and retrieve the data, it does not create a stream for the instance; in this case, the plug-in may want to receive instant notification of the result.

- NPP_URLNotify
  - Communicator Notifies the instance of the completion of a URL request made by NPN_GetURLNotify or NPN_PostURLNotify

INTERNET ENGINEERING

Post URLs

- The plug-in calls NPN_PostURL to post data from a file or buffer to a URL. This function is the counterpart of NPN_GetURL.

- NPN_PostURL writes data from a file or buffer to the URL and either displays the server response in the target window or delivers it to the plug-in.

- For HTTP URLs only, Communicator resolves this method as the HTTP server method POST, which transmits data to the server.

- The result from the server can also be sent to a particular Communicator window or frame for display, or delivered to the plug-in instance in a new stream. Plug-ins can use this capability to post form data to CGI scripts using HTTP or upload files to a remote server using FTP.

INTERNET ENGINEERING

Streams

- Streams are objects that represent URLs and the data they contain, or data sent by a plug-in without an associated URL.

- Although a single stream is associated with one specific instance of a plug-in, a plug-in can have more than one stream object per instance.
- Streams can be either:
  - produced by the Browser and consumed by a plug-in instance
  - produced by an instance and consumed the Browser.

- Each stream has an associated MIME type identifying the format of the data in the stream.

INTERNET ENGINEERING

Streams into Plugins

- Streams produced by Communicator can be either:
  - automatically sent to a plug-in
  - requested by the plug-in.

- Communicator calls the Plug-in methods:
  - NPP_NewStream to create
  - NPP_WriteReady to find out how much data the plug-in can handle
  - NPP_Write push data into the stream
  - NPP_DestroyStream to delete it.

INTERNET ENGINEERING

Modes of Streams

- The plug-in instance selects a transmission mode for streams produced by Communicator.
- Stream data can be pushed by the Browser, pulled by the plug-in, or saved to a local file and passed to the plug-in.

  - Normal mode: Communicator uses the NPP_Write method to "push" stream data to the instance incrementally as it is available.
  - Random-access mode: The plug-in calls the NPN_RequestRead method to "pull" stream data. This mode is more expensive, because the entire stream must be downloaded to a temporary file before use unless the stream comes from a local file or an HTTP server that supports the proposed byte-range extension.
  - File mode: Communicator saves the entire stream to a local file and passes the file path to the plug-in instance through the NPP_StreamAsFile method.

INTERNET ENGINEERING

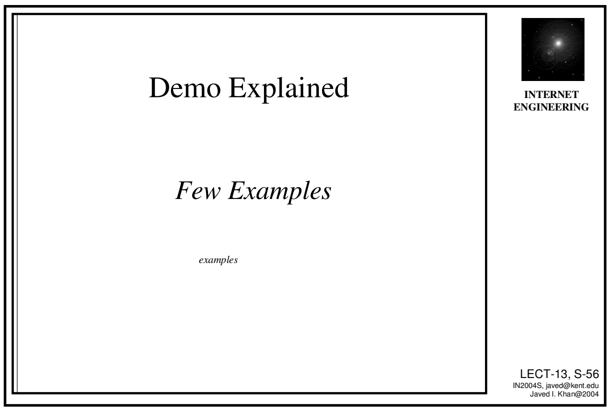
Streams from Plugin to Browser

- Streams sent by the plug-in to Communicator are like normal-mode streams produced by Communicator, but in reverse.

- In normal-mode streams, Communicator calls the plug-in to tell it when a stream is created and to push more data. In contrast, for streams produced by the plug-in, the plug-in calls the Plug-in API methods :

  - NPN_NewStream to create a stream
  - NPN_Write to push data into it
  - NPN_DestroyStream to delete it.

INTERNET ENGINEERING



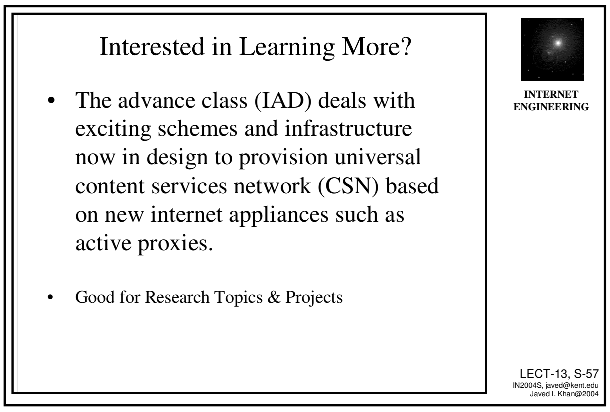
Demo Explained

Few Examples

examples

INTERNET ENGINEERING

LECT-13, S-56
IN2004S, javed@kent.edu
Javed I. Khan@2004



Interested in Learning More?

- The advance class (IAD) deals with exciting schemes and infrastructure now in design to provision universal content services network (CSN) based on new internet appliances such as active proxies.
- Good for Research Topics & Projects

INTERNET ENGINEERING

LECT-13, S-57
IN2004S, javed@kent.edu
Javed I. Khan@2004