Cross Platform Cross Browser VMS Plug-In

Meet Patel[1] Karan Gandhi[2] Sudhir Bhadauria[3] Vijay Patel[4]
[1,2]M.Tech (VLSI & Embedded Systems)
[3]Department of Electronics and Communication Engineering
[1,2]eiTRA- UVPCE, Ganpat University, India. [3]e-Infochips Pvt. Ltd., Ahmadabad, India. [4]UVPCE, Ganpat University, India.

Abstract— A Plug-in is a piece of software that manages internet content that browser is not designed to process. These usually include patented formats for video, audio, online games, presentations, and more. In this competitive era of technology people want to work from any device like Smartphone, tablet, desktop and laptop with different operating systems. So need of cross platform technology is high. This paper gives an idea of how a plug-in works, its life cycle. Once plug-in gets loaded into memory how to display video/image to any browser window (Cross Browser).

Keywords: Cross Platform[1], Cross Browser[2],Plug-in Life-Cycle[3]

I. INTRODUCTION

A plug-in is a software component that adds a specific feature to an existing software application. When an application supports plug-ins, it enables customization. The common examples are the plug-ins used in web browsers to add new features such as search-engines, virus scanners, or the ability to utilize a new file type such as a new video format. A plug-in declares that it handles certain content types (e.g. "audio/mp3"). When the browser encounters that content type it loads the associated plug-in, sets aside space within the browser context for the plug-in to render and then streams data to it. The plug-in is then responsible for rendering the data. The plug-in runs in-place within the page, as opposed to older browsers that had to launch an external application to handle unknown content types.

A cross browser cross platform VMS plug-in is a video management system plug-in that runs on any browser on any platform. VMS, Video Management System (VMS) is a system which manages access and controls the IP cameras. With GUI one can view camera video through Real Time Streaming Protocol (RTSP) [9] and control camera for video analytics. Another important feature is access this camera from internet via browser. Browser does not have all the functionality to access and control camera we need a plug-in to do this task. So basically a VMS plug-in that runs on any platform (windows, Linux) & on any browser is a cross browser cross platform VMS plug-in.

II. OVERVIEW

Plug-ins are shared libraries that users can install to display content that the application itself can't display natively. For example, the Adobe Reader plug-in lets the user open PDF files directly inside the browser, and the QuickTime and RealPlayer plug-ins are used to play special format videos in a web page. There are two things to build a plug-in. One is on which technology you have to build plug-in and second is what kind of display technology is used to display the content on browser.

A. Plug-in technologies:

Plug-ins are written using NPAPI [4], Google's NaCl [5], Native messaging [6], Mozilla Extensions js-ctypes [7], Web Sockets [8]. NaCl uses a special C/C++ compiler and the Pepper API (PPAPI) to allow the creation of mostly-native code that can run "safely" in your browser. But it has disadvantage that it only works on Google Chrome. Native Messaging allows a Chrome Extension to exchange messages with native applications. It has also disadvantage as Native client it only works with Google chrome. Js-ctypes allows application and extension code to call back and forth to native code written in C. Basically it allows you to call into a DLL or similar from your JavaScript extension. Disadvantage of this technology is it only works on Mozilla products like Firefox/Thunderbird. About Web Sockets, can use to connect to an application running on the local system and communicate with it, thus allowing that application to provide us with services that are otherwise unavailable in the browser, such as native TCP/UDP socket access, hardware access, etc. Cons of this technology are one application would need to be launched somehow manually and it only works on newer web browsers.

So, Netscape Plug-in Application Programming Interface (NPAPI) is a cross-platform plug-in architecture used by many web browsers. It was first developed for Netscape browsers, starting with Netscape Navigator 2.0, but was subsequently implemented by many other browsers. Further implementation is based on NPAPI.

B. Display technology:

For display a video there are several technologies are available for different platform like GStreamer [10], OpenGL [11], SDL [12], GTK (Linux) [13] & GDI (Windows) [14]. GStreamer is a pipeline-based multimedia framework written in the C programming language with the type system based on GObject. Simple Direct Media Layer (SDL) is a cross-platform development library designed to provide low level access to audio, input devices, and graphics hardware via OpenGL and Direct3D.OpenGL (Open Graphics Library) is a cross-language, multi- platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. GTK+ (GIMP Toolkit, sometimes incorrectly referred to as the GNOME Toolkit) is a cross-platform widget toolkit for creating graphical user interfaces. The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers.

All display technologies plays different scenario listed above. But what we want technology must work on any platform and browser with NPAPI. So, we choose GTK & GDI for display because GStreamer is heavy in size so we cannot bind it with plug-in application and for SDL & OpenGL we found that technologies are not compatible with web applications. So are like if we are using plug-in in Linux so choose GTK or else for windows choose GDI.

III. IMPLEMENTATION

To develop web browser plug-ins we found several frameworks that hide the details of the web browser interfaces for plug-in development. Such technologies allow developers to focus solely on the features they want to make available on the plug-in, and sometimes they can help creating a plug-in that can be successfully used in web browsers. Using a framework allow us to save the time needed to understand the NPAPI and ActiveX [15] Controls technologies. Thus, it would enable us to develop NPAPI and ActiveX plug-ins with the same source code core for the features we designed to our mechanism. There are four main frameworks that help developers build web browsers plug-ins. The first one that we discarded from this set of frameworks was Nixysa, because it only has support for NPAPI plug-ins, it has poor documentation, and the last release is relatively old (2009). The three remaining libraries are very similar; they all have support for the major web browsers and OSs, and good documentation. Among these options we chose FireBreath [16] since it is restricted to the development of web browser plug-ins.

A. How Plug-ins works

The life cycle of a plug-in as shown in figure 1, unlike that of an application, is completely controlled by the web page that calls it. This section gives you an overview of the way that plug-ins operate in the browser. It looks for plug-in modules in particular places on the system. When the user opens a page that contains embedded data of a media type that invokes a plug-in, the browser responds with the following sequence of actions:

- Check for a plug-in with a matching MIME [17] type.
- Load the plug-in code into memory.
- Initialize the plug-in.
- Create a new instance of the plug-in.

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.

1) Initialization

Sometimes it is necessary to initialize something only once for the entire plug-in process; singleton classes that are shared by every instance, for example. When the plug-in process is created, before your constructor is called the global function global Plug-in Initialize () is run which calls static Initialize () on your plug-in object. When the plug-in process is destroyed, after your destructor is called, the global function global Plug-in Deinitialize () will be called. This function calls static Deinitialize () on your plug-in object.

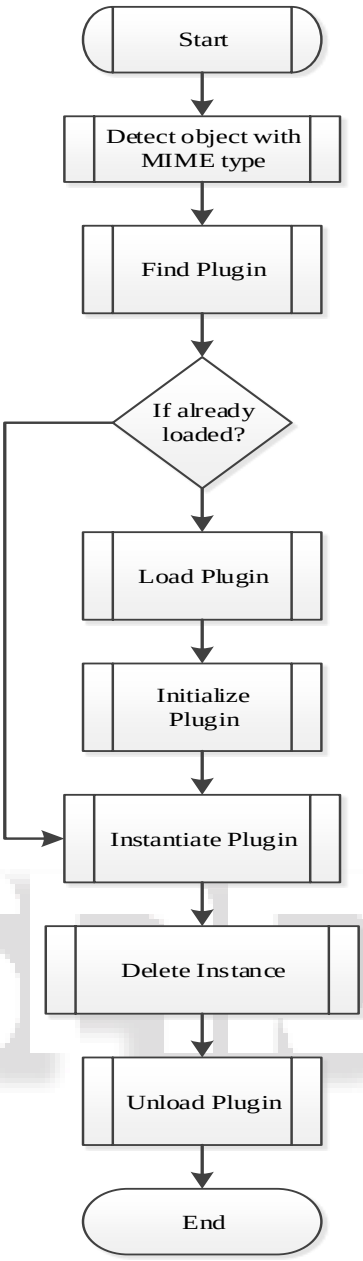

Fig. 1: Lifecycle of Plug-inPLUG-IN START-UP

Order of calls during plug-in Start-up:

- Plug-in Object constructor.
- Plug-in FSPath (File system path to the DLL on windows) is set, if on windows.
- Browser Host object is set on the Plug-in Object (inherited from Plug-in Core).
- The browser gives the plug-in a window (platform dependent) and the Plug-in Object receives Attached Event.
- Fire Breath requests the root JSAPI object from the Plug-in Object by calling getRootJSAPI (). GetRootJSAPI() calls createJSAPI(), which is a function in your plug-in class, to instantiate the root JSAPI object if it hasn't been instantiated already.

At this point your plug-in should be completely initialized and ready to go. The plug-in constructor is called first; anything you do in that constructor must not require a Browser Host or a window, since those are not initialized

until later. For these operations wait for onPluginReady () to be called.

2) Plug-in Shutdown

When the user leaves the page or closes the window, the plug-in shuts down. Order of calls during plug-in shutdown:

- Plug-in Object receives Detached Event – immediately after this the plug-in window will be destroyed, so act accordingly.
- Shutdown () is called on the Plug-in Object.
- Plug-in Object destructor is called.

B. How To get a stream

Our aim is to display video on browser window so some kind of mechanism should get video frame by frame. There are several of ways to get streaming like RTSP, RTP, UDP, MMS and HTTP. The Real Time Streaming Protocol (RTSP) is a network control protocol designed to control streaming media servers. The protocol is used for establishing and controlling media sessions between end points. RTSP client resides on user side to get feed of camera and display it.

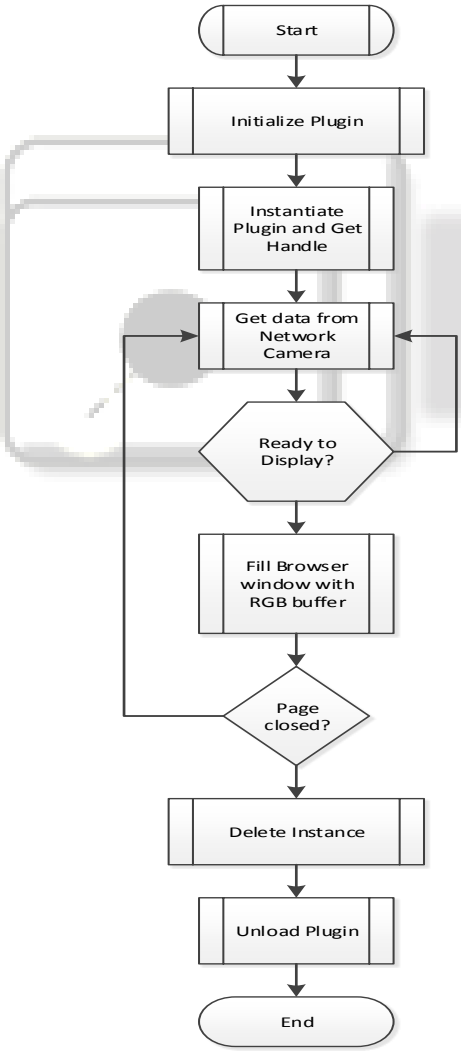

Fig. 2: Algorithm

IP camera supports RTSP protocol and H264 codec [18] so with RTSP client get the feed which is H264 data. We cannot display compressed H264 data direct to browser window. Browser can display RAW data like RGB. To convert H264 data into YUV420p [19] data we have to use FFMPEG. Than convert YUV data into RGB data. So the procedure of display video on browser window is given below,

- Get H264 data with RTSP.
- Convert H264 data into YUV data.
- Convert YUV data into ARGB data buffer.
- Display ARGB data buffer [19] with bowser handle on window.

Same sequence will be executed to display next frame for visualizing video. Coding flow is shown in figure 2. Once we get RGB data buffer we have to draw that buffer into browser window. We have to generate expose event to tell browser that Something we have to draw on browser window. If RGB buffer is not available to use than just skip the drawing part and wait for next drawing event. To generate drawing event in Windows and Linux both are different. In Windows when you call onWindowRefresh () and in Linux OnX11 events will create display event.

IV. Testing and Results

When a user browses to a web page that invokes a plug-in. So for testing purpose we have to create a HTML web-page file with appropriate MIME type listed on it to invoke plug-in. Also video tag to create window in browser in which video will be shown. To check plug-in is working correctly or not we have to open HTML web-page into different web-browsers and check the behaviour and memory usage.

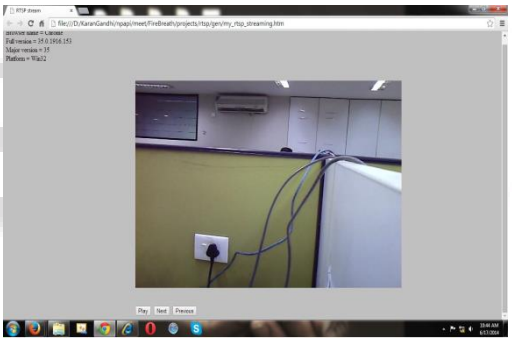

Fig. 3: Streaming in chrome Browser.

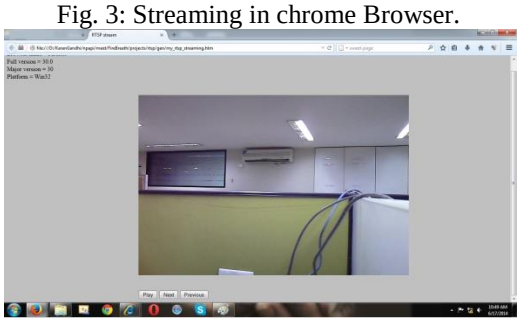

Fig. 4: Streaming in Firefox.

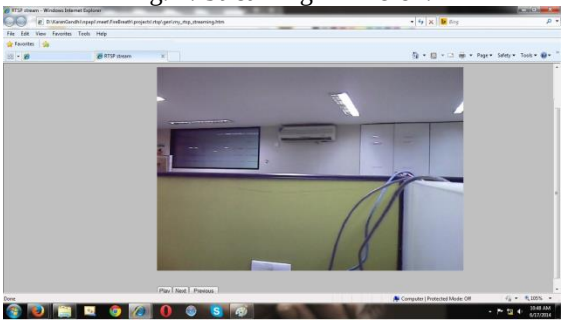

Fig. 5: Streaming in Internet Explorer.

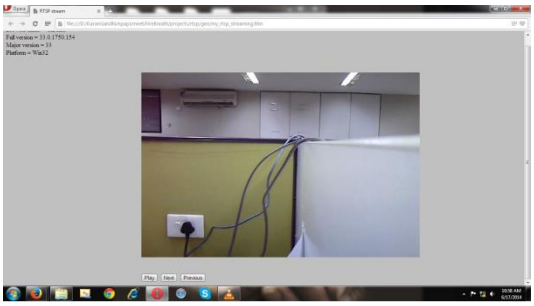

Fig. 6: Streaming in Opera.

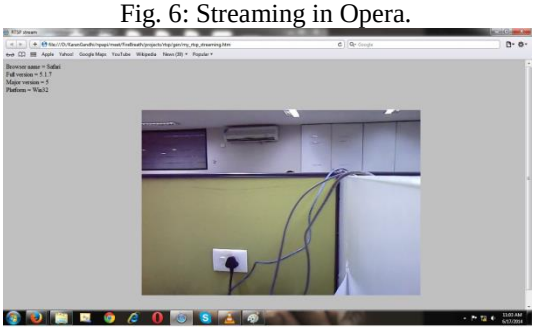

Fig. 7: Streaming in Safari.

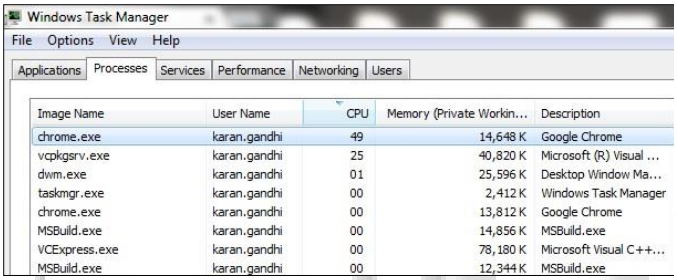

Fig. 8: Memory usage during Streaming.

REFERENCES

[1] "What is Cross Platform" http://en.wikipedia.org/wiki/Cross-platform
[2] "Cross Browser" <http://en.wikipedia.org/wiki/Cross-browser>
[3] "Plug-in Life-Cycle" <http://www.firebreath.org/display/documentation/Plug in+Lifecycle >
[4] "Netscape Plugin Application Programming Interface (NPAPI)" < http://en.wikipedia.org/wiki/NPAPI>
[5] "Native Client" < https://developer.chrome.com/native-client>
[6] "Native messaging" <https://developer.chrome.com/extensions/messaging#native-messaging>
[7] "Mozilla'sjs-ctypes" <https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes>
[8] "WebSocket" < http://en.wikipedia.org/wiki/WebSocket>
[9] "RTSP – Real Time Streaming Protocol" <http://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol>
[10] "GStreamer" < http://en.wikipedia.org/wiki/GStreamer>
[11] "Open Graphics Library - OpenGL" < http://en.wikipedia.org/wiki/OpenGL>
[12] "Simple DirectMedia Layer - SDL" <http://en.wikipedia.org/wiki/Simple_DirectMedia_Layer>
[13] "GTK - GIMP Toolkit" < http://en.wikipedia.org/wiki/GTK+>
[14] "Graphics Device Interface - GDI" <http://en.wikipedia.org/wiki/Graphics_Device_Interface>
[15] "ActiveX " < http://en.wikipedia.org/wiki/ActiveX>
[16] "FireBreath" <http://www.firebreath.org/display/documentation/About+FireBreath> < http://www.firebreath.org>
[17] "Multipurpose Internet Mail Extensions (MIME)" < http://en.wikipedia.org/wiki/MIME>
[18] "H264" < http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC >
[19] "YUV420p" and "ARGB buffer" < http://en.wikipedia.org/wiki/YUV >

