http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (6): 1058-1062, 2006
ISSN 1812-5638

Flash Support on Embedded Web Browsers

Kenguka Mohamed Kenguka and Atindimile Suleiman Kumchaya
Department of Electronics and Information Engineering,
Huazhong University of Science and Technology, 430074 Wuhan, Hubei, People's Republic of China

Abstract: This study aimed to resolve one of problems which face most of current embedded web browsers for example the web browser provided by MINIGUI which does not support Flash. The present study developed an embedded web browser the add flash on flash plug in interface. The designs based on Mozilla and Konqueror reference architecture, since these applications constitute a good sample of current web browser technology and are quite distinct from others as they are all based on different rendering engines and GUI toolkits. During the development work flash support was enabled in the browser by adding functions, module and data types then modify some parts of the source code. After development the browser was tested several time and changes were made, final test performed and the browser supported flash player without any problem. Since flash objects are huge and the browser builds DOM and RENDER tree, it sacrifices its browsing speed. On the other hand new technologies are immerging into web page every day and web pages are updating from time to time. Facing these facts, the developed browser may meet some difficulties therefore the study expect, it not to display well some blocks in a web page.

Key words: Embedded system, embedded web browser, flash engine, Kenguka

INTRODUCTION

Back in November 2000, when the Konqueror/Embedded (Hausmann, 2001) project started, basically only two browsers were available for embedded Linux environments. The first one was ViewML by Censoft, it was and still is, based on the old khtmlw-rendering engine from KDE version 1.X. At the time of its release ViewML was certainly the most advanced and smallest free browser available. Censoft ported the rendering engine to the FLTK toolkit, added a custom HTTP client implementation and wrote a small graphical interface around it. Today it is still one of the smallest browsers, but it lacks support for popular current web standards. The other browser was Opera, a commercial product from Opera Software. Opera Software also based their Linux and embedded Linux-based browsers on Qt (Griffith, 2001). While Opera is quite fast, it is not open source. Therefore, the lack of availability of a free and standards-compliant web browser for embedded systems was the main motivation behind the Konqueror/Embedded Project.

Since the current embedded web browsers are not good enough due to the fact that they do not support web pages with JAVASCRIPT or FLASH and they used to crash very often when processing web pages with Flash objects (Massa, 2005). So it is very clear that there is a need of better design so that it could implement the necessary functions of an embedded browser which were missing.

The present study propose a design which can be used for implementing and enabling Flash player on embedded web browsers, Konqueror embedded browser in Linux gives a very good example for the design (Hausmann, 2001). The study work on flash player support on embedded web browser due to the fact that it makes web pages brilliant and help to reduce web pages area because more than one advertisement can be placed on the same are then be displayed as flash designer wants. This study together with flash engine implementation proposal it also avoid crashes on the web pages which might be caused by flash support.

SYSTEM OVERVIEW

During development of modern embedded web browser conducted at Internet Technology Research and Development Center of Huanzhong University of Science and Technology, Flash, JavaScript, Cookies and HTML 4 support was enabled on the browser which was

Corresponding Author: Kenguka Mohamed Kenguka, Department of Electronics and Information Engineering, Huazhong University of Science and Technology, 430074 Wuhan, Hubei, People's Republic of China

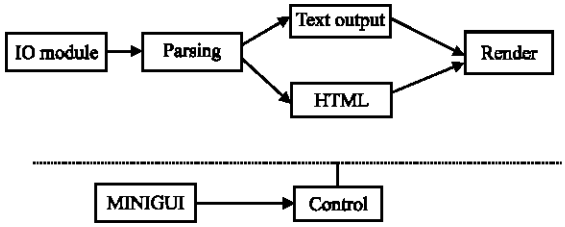

Fig. 1: Web browser function block

developed between March 2005 and January 2006. This study focuses on flash player support on the developed browser, here I present the general overview of the browser and point out location of Flash Engine.

Figure 1 shows the general structure of a web browser module. The web browser module is embedded into Linux, which runs at the platform of MiniGui in Arm-Linux (Aggranat, 2003). The research and design work on browser are based on the KONQUEROR browser in Linux or the monqueror browser carried by MiniGui (Shuyi and Feynman, 2003-2005). Linux divides the web browser into two main parts which are user interfaces and browser kernel (Nutt, 2002). User interfaces are composed of two different modules named MINIGUI (Feynman, 2005) and Control module. The browser kernel is composed of IO module, parsing module and Render module. IO module uses http to get the original web page data and cache it, parsing module tokenizes the HTML page and parse into text output and HTML output, render module displays the whole web page by calling the related modules provided by Linux. Parsing module is enabled to support FLASH. Render module can work without much change if the web browser is stable enough and user interface should add some icons to call kernel (Stenback *et al*., 2003).

Web browser basic structure: Figure 2 shows the basic structure of an embedded browser (Stenback *et al*., 2003) (Le Hors *et al*., 2004).

Design details: Figure 3 shows the browser's implementation model for research and design in an embedded system. The line composed of yellow blocks shows the flow of data to be processed in browser. And the green blocks are attached to correspond yellow block in order to support some special functions such as JavaScript, Flash and so on (Stenback *et al*., 2003; Le Hors *et al*., 2004).

Implementation and design work were done to the respective parts of Fig. 3.

Major flash engine modifications: During the development process of the project, a lot of modification and changes to Konqueror-Embedded

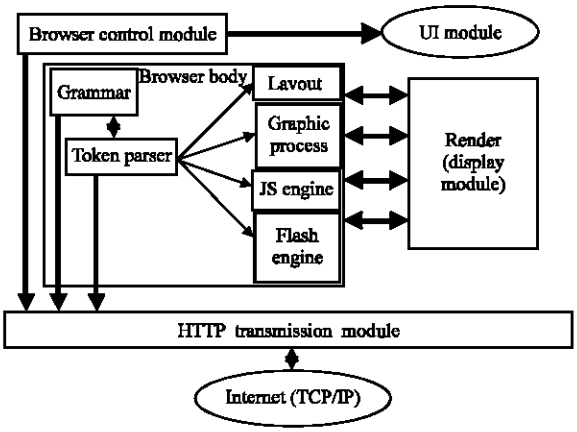

Fig. 2: Basic structure of embedded browser

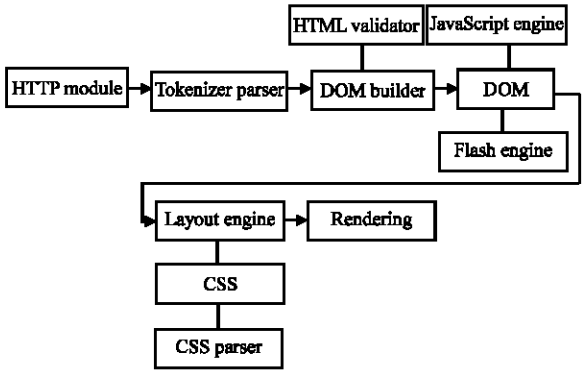

Fig. 3: Implementation model

Browser (Hausmann, 2001) were made, below are some of the changes which were made in order to enable Flash support.

- Class MGDocLoader: Add a member data: MGFlashLoader *m_FlashLoader
- LoadUrlStruct: Add member data about Flash: MGCachedFlash *cachedFlash; RenderPartObject *renderflash;(in MgLoadObject.h)
- enum MGLOADERTYPE: I added a type of Flash.(in MgLoadObjec.h)
- In function getElement(Token *t) of Htmlparser.cpp file, I added the following lines:

  case ID_EMBED:
  n = new HTMLEmbedElementImpl(document);
  break;

FLASH ENGINE

In html, the flash is titled by "EMBED". In order to support flash, the browser should recognize "EMBED" (Griffith, 2001). For the browser to recognize "EMBED",

four classes needed to be implemented: HTMLEmbed ElementImpl, MGFlashLoader, MGCachedFlash and Render Part Object. Here class HTML Embed Element Impl is used to descript Flash; class MG Flash Loader is used to download Flash, class MG Cached Flash is used to cache a Flash and class Render Part Object is used to render a Flash.

Flash engine classes details: The study give detailed explanation of four important classes and their functions together with data involved in it.

1. HTMLEmbedElementImpl

```
class HTMLEmbedElementImpl : public HTMLElementImpl
    {
        public:
    HTMLEmbedElementImpl (DocumentImpl *doc);
    ~HTMLEmbedElementImpl();

    virtual const DOMString nodeName () const;
    virtual ushort id () const;

virtual tagStatus startTag () { return OBJECTStartTag; }
virtual tagStatus endTag () { return OBJECTEndTag; }

    virtual void parseAttribute (AttrImpl *attr);

    virtual void attach (MGHTMLView *w);
    virtual void detach ();
virtual void applyChanges (bool top=true, bool force=true);
virtual void recalcStyle ();

virtual bool mouseEvent (int _x,int _y,int button,MouseEventType type, int _tx, int _ty,DOMString &curUrl,NodeImpl *&innerNode,long &offset);

     QString url;
     QString pluginPage;
     QString serviceType;
     bool hidden;
     QStringList param;
     protected:
     DOMString flashURL;
     MGHTMLView *m_view;
} ;
```

*virtual void parseAttribute(AttrImpl *attr*): This function is used to parse attributes of an object. All html objects should implement this function. For example "class HTMLImageElementImpl".

*virtual void attach(MGHTMLView *w):* This function is used to bind with a view. If we bind with a view, we will use this view to show the object.

*virtual bool mouse Event(int _x,int _y,int button, MouseEventType type, int _tx, int _ty, DOMString and curUrl,NodeImpl *innerNode, long & offset):* This function is used to accept mouse event. For example, if we want to make our Flash accept "double click" event.

DOMStringflashURL: This is Flash's url and it is used as a key word to find flash data from the cache.

2. MGFlashLoader

```
class MGFlashLoader : public MGLoadObject
     {
     public:
    MGFlashLoader ();
    ~MGFlashLoader ();
    bool LoadedMsgFromNet (unsigned char *url, unsigned char *, int length);
    MGCachedFlash *RequestFlash (unsigned char *url, RenderPartObject *render);
    void FromUrlLoaded (unsigned char *url, RenderPartObject **render);
};
```

*void FromUrlLoaded(unsigned char *url, RenderPartObject **render):* This function uses url as keyword to get a object of RenderPartObjec.

*MGCachedFlash *RequestFlash(unsigned char *url, Render Part Object *render):* If url is local file, this function will get Flash from local file, if the Flash of this url has been downloaded, this function will get Flash from temporary file and if the Flash of this url is not downloaded, this function will put a request to network thread.

*bool LoadedMsgFromNet(unsigned char *url, unsigned char *, int length):* used url as keyword to get a flash from cache.

3. MGCachedFlash

This class is used to cache Flash.

```
    class  MGCachedFlash
    {
public:
        MGCachedFlash (unsigned char *url);
        MGCachedFlash (unsigned char *url, char TempName [ ] );
        MGCachedFlash(unsigned char *url, char TempName [ ] , MGPainter *p) ;
        ~MGCachedFlash ();
        void ref () {m_ref++;}
        void deref () {m_ref--; if (m_ref<=0) delete this;}
```

```
        boolLoadDataFromTempFile(charTempName());
        const MGPixmap& pixmap () const;
        MGSize pixmap_size() const;
        void copyFlash(char TempName[]);
    protected:
        friend class RenderPartObject;
        DOM: :DOMString m_URL;
        DOM: :DOMString m_baseURL;
    public:
    //  MgContext xc1, *xc = &xc1;
    //  struct FlashInfo fi;
        long length;
        char *buffer;
        bool m_bIsRequested;
        MGPixmap *bg;
        MGPixmap *p;
    //  FlashHandle flashhandle;
    private:
        int m_ref;
    };
```

long length: Is the size of Flash

char *buffer: Is the data of Flash.

bool LoadDataFromTempFile(char TempName[]): This function is used to copy flash from file "TempName[]" to buffer.

DOM::DOMString m_URL: Is the url of Flash, it is used as a keyword to find Flash.

4. RenderPartObject

From the name of the function we can tell what the function is about to do.

```
class RenderPartObject : public RenderPart
{
    // Q_OBJECT
public:
    Render Part Object ( MGHTMLView *view,
DOM::HTMLElementImpl *o );
    virtual ~Render Part Object ();

virtual const char *renderName () const { return "Render
Part Object"; }

    virtual bool isRendered () const { return true; }
    virtual void calcMinMaxWidth ();
    MGPixmap pixmap () const { return pix; }
    virtual int bidiHeight () const;
    virtual short intrinsicWidth () const;
    virtual int intrinsicHeight () const;
    virtual void setStyle (RenderStyle *style);

    virtual void close ();
virtual void printReplaced(MGPainter *p, int tx, int ty);
    virtual void setWidget ( MGWidget *widget );
    virtual void setSize ( int w, int h );
    virtual void layout ( );
    virtual short width ( )const{return m_width;}
    virtual int height ( ) const{return m_height;}
        void    setFlashUrl(DOM::DOMString    url,
DOM::DOMString baseUrl, MGDocLoader *docLoader);
    //MGPixmap pixmap ( ) const { return pix; }
    void After Data Loaded();
    int getltx ( ){return ltx;}
    int getlty ( ){return lty;}
   int getrbx ( ){return rbx;}
    int getrby ( ){return rby;}

    DOM::HTML ElementImpl *m_obj;
    MGCachedFlash *flash;
    private:
    MGSize pixSize;
    MGPixmap pix;
    MGPixmap resizeCache;
    bool bComplete;
    int ltx;
    int lty;
    int rbx;
    int rby;
    short m_width;
    int m_height;
    };
```

*virtual void printReplaced(MGPainter *p, int tx, int ty):* This function is used to show the flash, when we move the mouse, click the mouse, or press the key, this function will be called.

*void setFlashUrl(DOM::DOMString url, DOM::DOMString baseUrl, MGDocLoader *docLoader):* This function is used to set the member MGCachedFlash *flash.

EMBEDDED WEB BROWSER TEST RESULTS

A Wireless Logistic Intelligent Terminal (WLIT) and a pc are needed to set up the test environmen (Fig. 4). Pc connects with WLIT's console port, also running sniffer to catch packets for analysis the WLITT should be connected to internet.

Test coverage: Three kinds of categories were covered in the test procedures

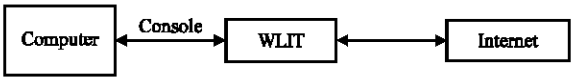

Fig. 4: Browser test environments

Table 1: Browser ability test

Procedures	1.	Run browser in QPE	(√)
	2.	Open the homepage(defined before test)normally	(√)
	3.	Open page http://www.hust.edu.cn and click anywhere you want	(√)
	4.	Open page file://kde/shared/apps/konq-embed/start.htm	(√)
	5.	Open page http://e.pku.edu.cn and input the search key of wireless terminal	(√)
	6.	Enter ftp://cong@cong:192.168.88.147:21 to open the ftp server.	(√)
Expect results		No segmentation fault occurs and the browser can browser the page well.	(√)
Notes and errors			

Table 2: Flash support test

Procedures	1.	Access the site of http://happy. ustonline. et/PFlash/ 5027200442019022 6.swf which contains the flash file.	(√)
	2.	After the whole page is downloaded and parsed. The flash control panel is embedded in the page.	(√)
	3.	Press the start key in the flash widget, then the flash will play in this page.	(√)
	4.	Check whether the sound is enable while playing this flash	(√)
Expect results	1.	When you click start key another task will be launched. And the flash will play in the page.	(√)
	2.	The browser can run well	(√)
Notes and errors			

- Case implementation-This was the main function of browser, such as browser ability and flash support.
- Meeting the requirement.
- Cover the all CLI of browser menu.

Test cases: During this test we tried to run the browser in QPE, open out university web page http://www.hust.edu.cn then clicked on several areas on the page, opened file file://kde/shared/apps/konq-embed/start.htm, opened page http://e.pku.edu.cn and search for key word "Wireless Terminal" and lastly opened ftp file ftp://cong@cong:192.168.88.147:21, all of the above test were successfully conducted (Table 1).

During this test procedure we accessed website http://happy.ustonline.et/PFlash/5027200420190226.swf which contained Flash files, after the whole page being downloaded, parsed and Flash control panel being embedded in the in the page we clicked on start button, checked if sound was enabled while playing Flash. The test was very successful as shown on Table 2.

CONCLUSIONS AND FUTURE WORK

This study aimed to resolve one of problems which face most of current embedded web browsers for example the web browser provided by MINIGUI which does not support JavaScript or Flash and crashes very often. Other embedded web browsers can not support cookies and higher HTML versions. All these embedded web browser's deficiencies disturbed embedded web browser's user to a very big extent.

The designed browser aimed to remove one of the deficiencies and provide embedded web browser's user with a comfortable surfing by enabling Flash player support in web pages.

Since Flash objects are huge and the browser builds DOM and RENDER tree, it sacrifices its browsing speed. On the other hand new technologies are immerging into web page every day and web pages are updating from time to time. Facing these facts, the developed browser may meet some difficulties therefore we expect it to not display well some blocks in a web page. Future work and direction to other embedded web browser designers will focus on resolving deficiencies shown in my designed browser.

REFERENCES

Feynman, 2005. MiniGUI Documents and Dev Packages. http://www.minigui.com.

Griffith, A., 2001. KDE 2/Qt Programming Bible. IDG Books Worldwide, Inc.

Hausmannn, S., 2001. Konqueror/Embedded: An Open-Source Web Browser for Embedded Linux System http://www.linuxjournal.com.

Le Hors, A., P.L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion and S. Byrne, 2004. Document object model (DOM) Level 3 core specification. W3C http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.

Massa, A., 2005. Nine tips for improving embedded web browser design. Comms Design. com.

Shuyi, Z. and Feynman, 2003-2005. MiniGUI documents and dev packages http://www.minigui. com.

Stenback, J., P.L. Hegaret and A.L. Hors, 2003. Document Object Model (DOM) Level 2 HTML Specificatio. http://www.w3.org/TR/DOM-Level-2-HTML.