



Java is a trademark of Sun Microsystems, Inc.

# JavaOne<sup>SM</sup>

## JWebPane Overview

Alexey Ushakov  
Sun Microsystems

# Where do we need it?

- > Email programs or IMs (viewing HTML)
- > Rich clients for content delivery systems (providing reviews for the content)
- > Arbitrary Java or JavaFX applications (adding advert banners)
- > Anywhere you can imagine!

# Project Goal

Lightweight HTML component for Java and JavaFX providing:

- Easy to use API
- Modern HTML support
- Java - JavaScript binding
- DOM access
- Plugins

# Solutions

- > Create HTML renderer from scratch
  - Huge effort
  - Endless ongoing work to match modern HTML
- > Enhance Swing HTML support
  - Significant work to get Swing up to date
  - Need to keep it up to date
- > Use existing HTML rendering engine
  - Mozilla
  - Webkit

# Solutions: Mozilla, Webkit

## > Mozilla

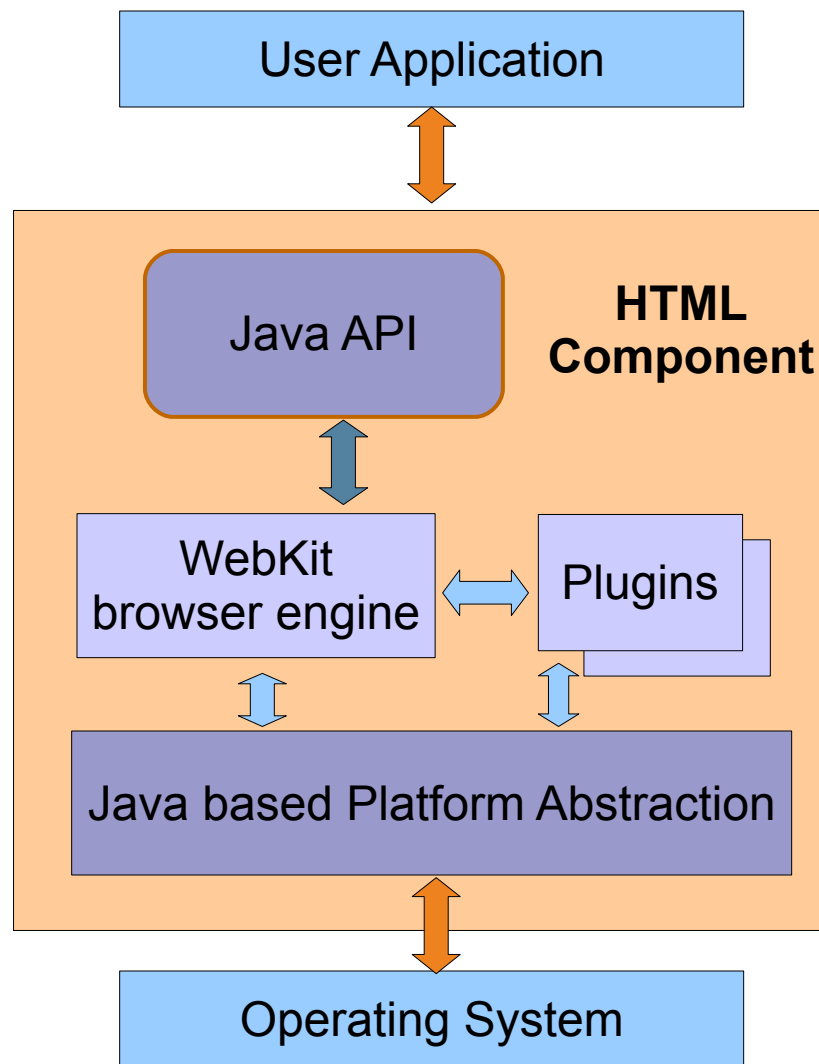
- Up-to-date
- Opensource
- Heavyweight

## > Webkit

- Up-to-date
- Opensource
- Lightweight

# Webkit based Implementation

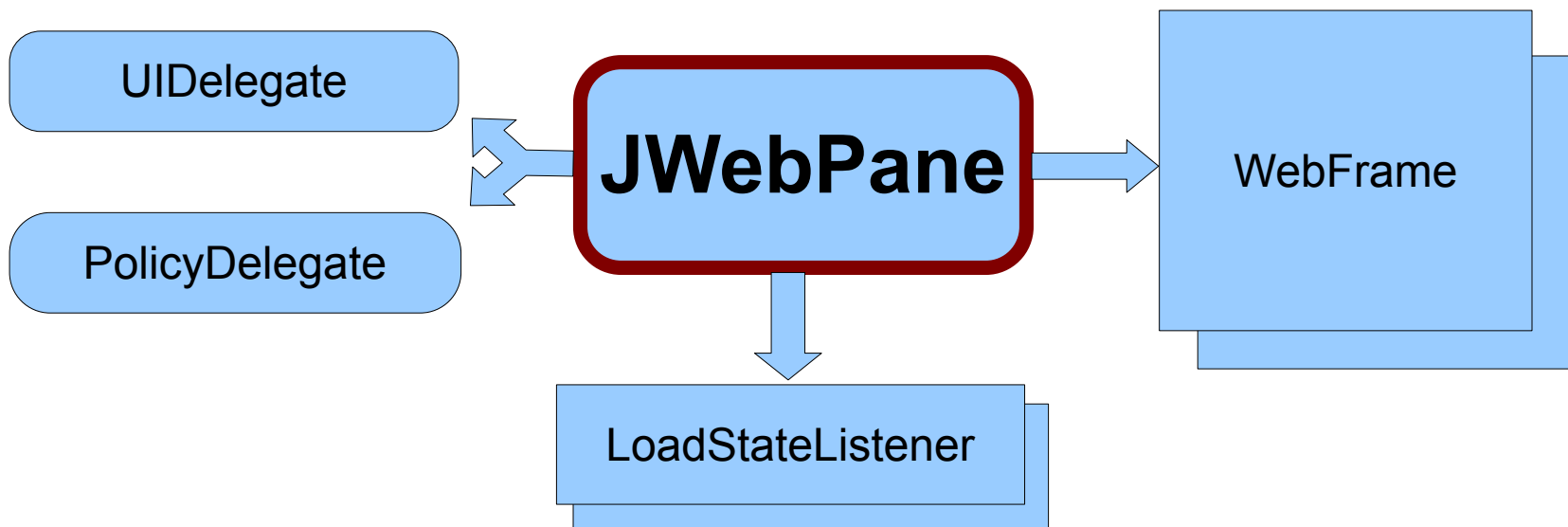
- > WebKit engine for HTML parsing, CSS, JavaScript
- > Java for painting, metrics calculation, events handling, networking, unicode, input methods



# JWebPane plugins

- > Media Plugin
  - Embeds JavaFX Media Component
- > Java Plugin
  - Simple wrapper for Applet environment
  - Same JVM
- > ActiveX Plugin
  - Embeds IE Flash plugin

# API: JWebPane



## API: JWebPane

- > Provides basic web page browsing functionality and basic user interaction, such as navigating links, and submitting HTML forms.
  - Displays one web page at a time
  - Handles scrolling internally (no need for JScrollPane)

# API: JWebPane

```
JFrame f = new JFrame("Browser");
```

```
JWebPane browser = new JWebPane();
```

```
f.add(browser);
```

```
browser.load(someURL);
```

## API: WebFrame

- > WebFrame is a nonvisual object that identifies an HTML <frame>, <iframe>, or <frameset> element
  - One WebFrame per HTML frame displayed in a JWebPane
  - WebFrames have hierarchical structure
  - Disposed after loading new pages

## API: WebFrame

```
WebFrame wf = browser.getRootWebFrame();  
  
for (WebFrame f : wf.getFrames()) {  
    log.fine(f.getURL() + ":" + f.getTitle());  
}
```

# API: PolicyDelegate

- > Single object associated with JWebPane for implementing a browser policy by allowing or rejecting sensitive operations such as:
  - loading web pages
  - opening new browser windows
  - running scripts on pages

## API. PolicyDelegate

```
class MyDelegate implements PolicyDelegate {  
  
    public boolean permitAction(PolicyRequest r) {  
        return r.getType() != ENABLE_SCRIPTS;  
    }  
  
}
```

```
browser = new JWebPane(new MyDelegate(), ...);
```

## API: UIDelegate

- > GUI object associated with a JWebPane and provides some basic UI for it.
- > Provides a set of GUI-related callbacks invoked by associated JWebPane to customize its appearance
  - Displaying message/input boxes
  - Creating new browser view for the given URL
  - Setting status bar

## API: UIDelegate

```
class MyUIDelegate extends DefaultUIDelegate {  
    @Override  
    public JWebPane createView(URL url) {  
        JWebPane view = new JWebPane();  
        view.load(url);  
        return view;  
    }  
}
```

```
browser = new JWebPane(..., new MyUIDelegate());
```

## API: LoadStateListener

- > Listener for loading events such as:
  - loadingStarted
  - redirectProcessed
  - loadingFinished **or** (if there is some problem)
    - loadingRejected - PolicyDelegate rejects an operation
    - loadingFailed - loading fails due to an error
    - loadingStopped -loading is stopped with the `JWebPane.stop()` method

## API. LoadStateListener

```
class MyListener implements LoadStateListener {
    public void loadingStarted(LoadStateEvent e) {
        log.fine("Started:" + e.getURL());
    }
    public void loadingFinished(LoadStateEvent e){
        log.fine("Finished:" + e.getFrame().getTitle());
    }
    public void loadingFailed(LoadStateEvent e) {
        log.severe("Failed:" + e.getError());
    }
}
```

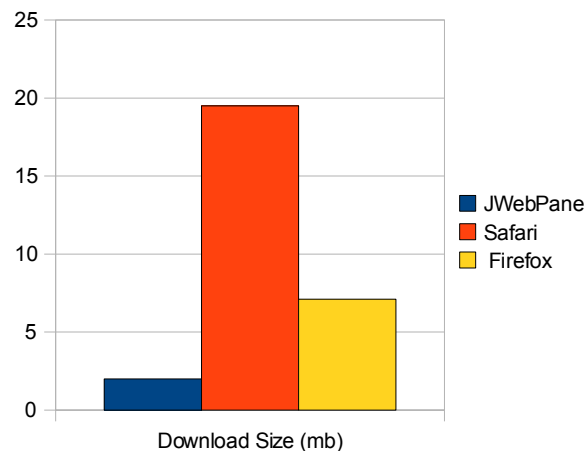
# Deployment

- > JWebPane can be deployed as a JNLP extension
  - Plugins are deployed as separate JNLP extensions
- > Platforms Supported
  - Windows
  - Mac OS X
  - Linux
  - Solaris

## Metrics (Windows XP)

### > Download size

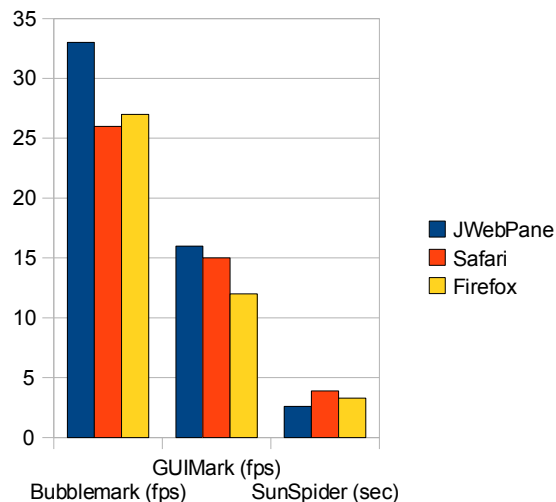
- JWebPane: 2 mb
- Safari 3.1: 19.5 mb
- Firefox 3: 7.1 mb



### > Performance

(JWebPane/Safari/Firefox)

- Bubblemark: 33/26/27 fps
- GUI Mark: 16/15/12 fps
- SunSpider: 2.6/3.9/3.3 sec



# Questions?

## > Blogs

- <http://weblogs.java.net/blog/alex2d>
- <http://weblogs.java.net/blog/ixmal>



# JavaOne<sup>SM</sup>

# Thank You

Alexey Ushakov  
alexey.ushakov@sun.com





## Where do we need it?

- > Email programs or IMs (viewing HTML)
- > Rich clients for content delivery systems (providing reviews for the content)
- > Arbitrary Java or JavaFX applications (adding advert banners)
- > Anywhere you can imagine!

Viewing HTML content: rich text formatting to message clients

Rich clients for content delivery systems ( an application store): to integrate external web resources such as blogs or reviews into the client

## Project Goal

Lightweight HTML component for Java and JavaFX providing:

- Easy to use API
- Modern HTML support
- Java - JavaScript binding
- DOM access
- Plugins

## Solutions

- > Create HTML renderer from scratch
  - Huge effort
  - Endless ongoing work to match modern HTML
- > Enhance Swing HTML support
  - Significant work to get Swing up to date
  - Need to keep it up to date
- > Use existing HTML rendering engine
  - Mozilla
  - Webkit

Currently Swing HTML components support HTML 3.2,  
but we need to handle HTML 4 and 5

## Solutions: Mozilla, Webkit

- > Mozilla
  - Up-to-date
  - Opensource
  - Heavyweight
  
- > Webkit
  - Up-to-date
  - Opensource
  - Lightweight

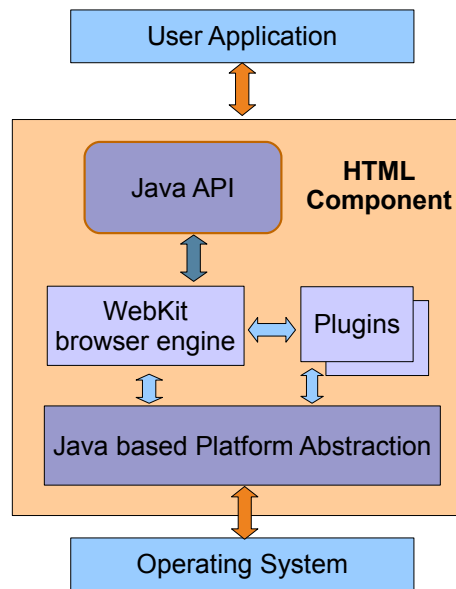
Mozilla has a pretty large open source developer community. It's up-to-date with modern HTML standards. However it doesn't allow us to easily create lightweight browser component.

Webkit also has a large opensource developer community. It's up-to-date, and fortunately for us it can be used for creating lightweight components.

One more benefit from our perspective is that it is extremely portable, as proven by a number of webkit ports for different toolkits and platforms. QT, GDK, windows mobile and symbian to name a few.

## Webkit based Implementation

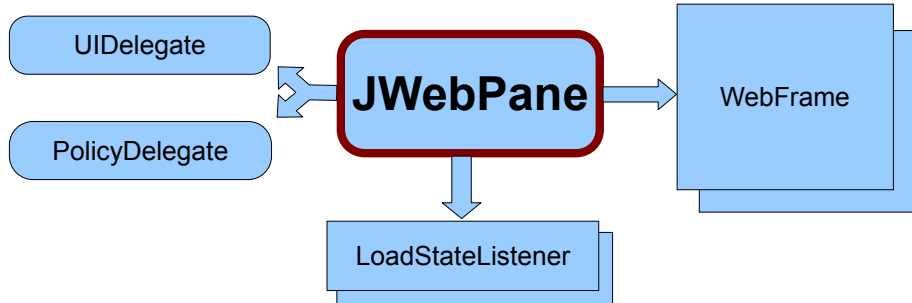
- > WebKit engine for HTML parsing, CSS, JavaScript
- > Java for painting, metrics calculation, events handling, networking, unicode, input methods



## JWebPane plugins

- > Media Plugin
  - Embeds JavaFX Media Component
- > Java Plugin
  - Simple wrapper for Applet environment
  - Same JVM
- > ActiveX Plugin
  - Embeds IE Flash plugin

# API: JWebPane



## API: JWebPane

- > Provides basic web page browsing functionality and basic user interaction, such as navigating links, and submitting HTML forms.
  - Displays one web page at a time
  - Handles scrolling internally (no need for JScrollPane)

JWebPane is a Jcomponent that provides basic browser functionality. It also provides registration of listeners and delegates, rendering and access to a document structure.

## API: JWebPane

```
JFrame f = new JFrame("Browser");  
JWebPane browser = new JWebPane();  
f.add(browser);  
browser.load(someURL);
```

Here is example of minimal application to show a web page.

## API: WebFrame

- > WebFrame is a nonvisual object that identifies an HTML <frame>, <iframe>, or <frameset> element
  - One WebFrame per HTML frame displayed in a JWebPane
  - WebFrames have hierarchical structure
  - Disposed after loading new pages

WebFrames are non visual objects representing corresponding HTML frames of a web page, therefore they are also linked to a tree structure.

WebFrames can be used for retrieving different information about HTML frames: URL, title, content type and source to name a few.

WebFrames have short period of life as they are disposed after loading a new html page into JWebPane

## API: WebFrame

```
WebFrame wf = browser.getRootWebFrame();  
for (WebFrame f : wf.getFrames()) {  
    log.fine(f.getURL() + ":" + f.getTitle());  
}
```



Here is example of using the WebFrame api. We obtain a root frame from a JWebPane browser object, then iterate through all child frames dumping their urls and titles.

## API: PolicyDelegate

- > Single object associated with JWebPane for implementing a browser policy by allowing or rejecting sensitive operations such as:
  - loading web pages
  - opening new browser windows
  - running scripts on pages

The JWebPane api provides this interface for control under certain networking or javascript operations.

Implementations may use arbitrary logic to allow or reject sensitive operations such as loading web pages, opening new browser windows or running scripts on pages.

## API. PolicyDelegate

```
class MyDelegate implements PolicyDelegate {  
    public boolean permitAction(PolicyRequest r) {  
        return r.getType() != ENABLE_SCRIPTS;  
    }  
}  
  
browser = new JWebPane(new MyDelegate(), ...);
```



Here is a policy delegate rejecting execution of on-page JavaScript. As you can see it has just one method that is called if the html page loaded into JWebPane has some javascript code.

PolicyDelegates should be passed to the JWebPane constructor as shown at the bottom of the slide

## API: UIDelegate

- > GUI object associated with a JWebPane and provides some basic UI for it.
- > Provides a set of GUI-related callbacks invoked by associated JWebPane to customize its appearance
  - Displaying message/input boxes
  - Creating new browser view for the given URL
  - Setting status bar

UIDelegate is a UI object associated with JWebPane. It provides UI related callbacks to customize the appearance of JwebPane. For example you can use JoptionPane for JavaScript message windows or you can put all new windows in tabs.

## API: UIDelegate

```
class MyUIDelegate extends DefaultUIDelegate {
    @Override
    public JWebPane createView(URL url) {
        JWebPane view = new JWebPane();
        view.load(url);
        return view;
    }
}

browser = new JWebPane(..., new MyUIDelegate());
```



This custom UIDelegate can be used to create new a web window in some specific way for the client application. For example, in a tab. So, here we override the createView method of the DefaultUIDelegate which is provided with the api.

At the bottom you can see how to custom UIDelegate.

## API: LoadStateListener

- > Listener for loading events such as:
  - loadingStarted
  - redirectProcessed
  - loadingFinished **or** (if there is some problem)
    - loadingRejected - PolicyDelegate rejects an operation
    - loadingFailed - loading fails due to an error
    - loadingStopped -loading is stopped with the JWebPane.stop() method

JWebPane provides a set of callbacks to track different loading events. They are represented in LoadStateListener interface.

## API. LoadStateListener

```
class MyListener implements LoadStateListener {
    public void loadingStarted(LoadStateEvent e) {
        log.fine("Started:" + e.getURL());
    }
    public void loadingFinished(LoadStateEvent e){
        log.fine("Finished:" + e.getFrame().getTitle());
    }
    public void loadingFailed(LoadStateEvent e) {
        log.severe("Failed:" + e.getError());
    }
}
```



Here is a listener for some tracing of the loading events providing some basic information for the web resources like URLs, titles of the pages or error codes

## Deployment

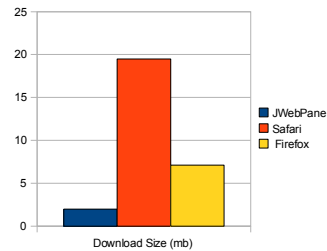
- > JWebPane can be deployed as a JNLP extension
  - Plugins are deployed as separate JNLP extensions
- > Platforms Supported
  - Windows
  - Mac OS X
  - Linux
  - Solaris

JNLP extension for JWebPane hides logic of choosing the right jars and native libraries for your application

## Metrics (Windows XP)

### > Download size

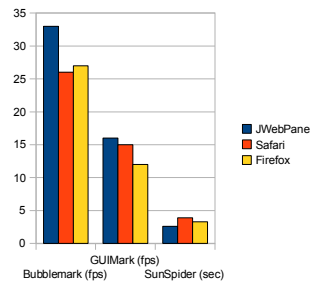
- JWebPane: 2 mb
- Safari 3.1: 19.5 mb
- Firefox 3: 7.1 mb



### > Performance

(JWebPane/Safari/Firefox)

- Bubblemark: 33/26/27 fps
- GUI Mark: 16/15/12 fps
- SunSpider: 2.6/3.9/3.3 sec



Download size of the component is about two megabytes. This is because we leverage Java platform for many things like rendering, networking, unicode.

Also, you can see that it has quite good performance regardless of the fact that all the rendering is performed by primitive call backs from native code to Java2D

## Questions?

### > Blogs

- <http://weblogs.java.net/blog/alex2d>
- <http://weblogs.java.net/blog/ixmal>

