UpStage Front-end Analysis

# Foreword

To look through the whole UpStage implementation, UpStage software roughly has four major components:

* Automated build features are written in Python executables (Python Script, as an alternative of Bash Shell script).
* Python Twisted supported back-en d (include all requests handling features)
* Application website written in HTML, JavaScript, JQuery, and Python Web.
* Application front-end written in ActionScript3

The main focus of this research is to prepare an in-depth analysis for the UpStage front-end technology in order to bring up a complete picture of the role for ActionScript in the current UpStage software.

# How Stages Created and Accessed

Before entering stage, UpStage users must create stage explicitly via UpStage application website. There is no direct connection between creating and entering stage features.

When authorised users (admin or creator privileges) create a stage, they send a request to the server. Then UpStage server shall generate a page to hold the stage, so that the users can navigate to the stage via valid url address.

Procedure:

* front-end script (*html/script/stageedit.js*), *stageChooseSubmit(create)* is invoked when users choose to create a stage.
* front-end script (*html/script/masterpage.js*), *buildRequestByFormName(forname)*
* front-end script (*html/script/masterpage.js*), *buildRequestFromForm(formElement).* Once this bit of code invoked, the method returns an url in string back to *buildRequestByFormName(forname)* method.
* Front-end script, (*html/script/masterpage.js*), fillPage method is invoked. The fillPage() method creates an new element for the stage via DOM.
* Front-end script, (*html/scirpt/masterpage.js*), fillPage method finally navigates users to the stage editing page. User can choose not to edit it by quitting the page, and enter the new stage via stage list page.

Due to the website setting resides on the server side, there will occur concurrency issues once UpStage is scaled up. (stages can be dropped/modified/recreated, while someone is editing the stage.)

From my observation, the work of opening stages on client side is done by AJAX. Front-end code uses *window.XMLHttpRequest* object to access the new page.

In *server/src/html/script*, most Javascript files contain navigation syntax such as:

window.location = ’*path*’;

When the user clicks on a stage from stage list, openStage() method is invoked, and openStage() method redirects the user to the corresponding stage. While the stage is loading, the stage page opens a template from server/config/templates/stage.xhtml, and load all the corresponding data (stage properties) into the template. As a result, the stage shows the same as other users enters the stage. This is done by use HTML <object> tags to embed a client.swf file from UpStage server build/html/swf/client.swf

There is an example on w3school website: <http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_object>

Due to loading can take some time, previous teams added a block of code to handle page on load function. Once the stage resources are loaded completely. The loading block is set to hide, so that the main application will appear in the browser tab.  
  
Another point worth noticing for JavaScript code. There are incomplete brackets and missing semicolons. However, JavaScript does not enforce the same strict syntax rules as Java does. As long as there is no logical error, JavaScript will not complain about missing brackets.

# Structure of ActionScript 3 implementation in UpStage V3

The web application of UpStage is a single App object extending MovieClip in ActionScript3 library.

An App object instantiates a Transport object (derived from XMLSocket class) to handle all the data transactions between clients' requests and server responses.

Transport adds a pointer to App instance.  
Transport object creates Sender and Auth instances. One thing worth look into, Transport extends XMLSocket object. Transport object is responsible for all the data transactions. The Sender object in Transport class is in charge of sending events to the server-end. Sender class sends data via XMLSocket’s method *send(Data data)*

------Current Drawing & Other Activities---------------------------------

UpStage Application uses XMLSocket as its main technology to handle data transportation. Below demonstrates the work flow of interactions between front-end and back-end in V3:

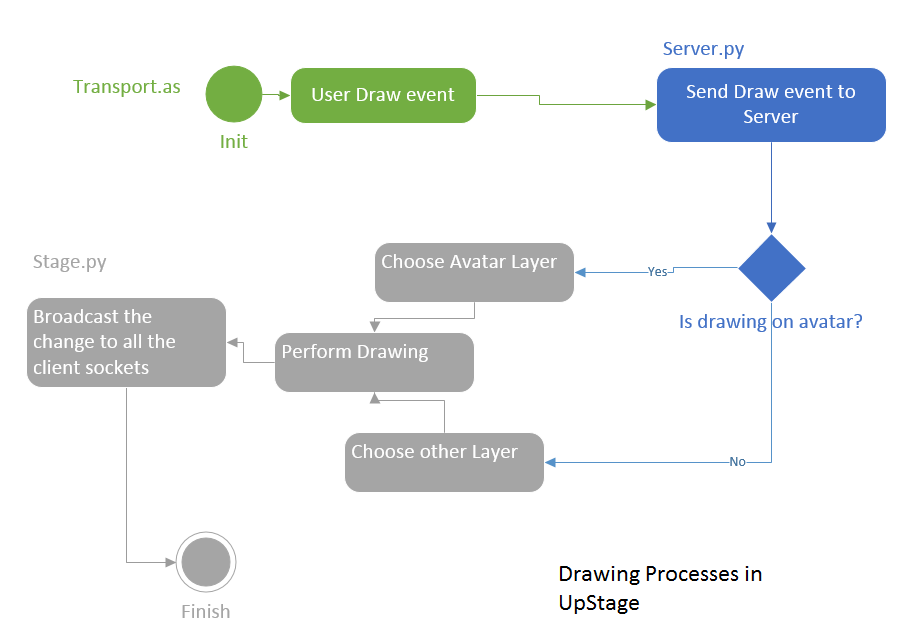
For instance, say we try to draw a dot on stage. The state of server is running and listening for requests. Assumed client has joined a stage, and selected drawing pen.

ActionScript3 handles events by binding items with .onPress/.onClick methods to certain objects.

When a client tries to make a drawing:

* client/src/upstage/Sender.as class sends a “draw” request to server.
* Since Server/src/upstage/server.py has a “draw” handler (listener), client's request is accepted, and passed to a different Python file: Server/src/upstage/stage.py
* Server computes the coordinates of the drawing locations, and broadcasts the drawing information to all the clients
* Once clients receive the drawing broadcast information, client side will perform drawings on the stage. (The drawing feature is enriched by various classes under /clientsrc//upstage/util)
* client/src/upstage/model/ModelDrawing class creates MovieClip and invokes MovieClip's lineTo() method to complete the drawings.

The rest of UpStage event handling functions follow the similar procedure; therefore, the rest functions can be placed into this situation.



-----Application Layer-------------- (may be removed later)

Transport object poses a disadvantage: Since Transport is not responsible for file uploads, the stage may not update itself correspondingly.

How UpStage application gets launched?

In order to use UpStage application, users must:

* Create a stage (App) instance.
* Once a stage instance created, user must upload media and avatars via the application website manually. This bit causes incoherent updates for audience.

Scenario for this:

* Player A enters stage
* Stage Owner/Creator adds new elements to stage
* Player B enters stage

In such order, if there is no force stage refresh when a file upload completion, people will see different elements on the stage.

# Consequences for removing ActionScript part from UpStage

Team must be aware of Flash plugin is deeply embedded in the current UpStage implementation. Even from UpStage server build process, Flash plugin and corresponding dependencies are required.

Removing Flash components shall cause not only losing access to UpStage web application for UpStage users but also demolishing the server installation process on Debian x86 distribution.

# Recommendations

* Replace *client.swf* file with HTML tags.
* Replace XMLSocket with Websocket derivations.
* Replace User List and Chatbox classes with HTML <div>, and apply CSS3 for rendering.

There are more features that future UpStage can do with web technologies. I would recommend a single-paged web application consolidating all the functional features.

# Challenges

* How to assure data update for all the clients with WebSocket rather than XMLSocket from ActionScript.
* Current UpStage front-end allows users to make drawings onto different layers; there is no direct solution to accomplish such feature in HTML5 technologies.

# Solutions

A prototype (the Node.JS prototype) is developed to use WebScoket instead of XMLSocket. The event handling work is replaced with *Socket.io.*

Litten (2010) delivered a solution to tackle multi-layered drawing by using web technologies from HTML5 stack.

# References

Litten. J. (2010) *Using Multiple HTML5 Canvases as Layers.* Retrieved from:

http://html5.litten.com/using-multiple-html5-canvases-as-layers/

# Versions

|  |  |  |
| --- | --- | --- |
| version | Date | Person |
| 0.1 | 06/04/15 | Yue |
| 0.3 | 11/04/15 | Yue |
| 0.4 | 13/04/15 | Yue |
| 0.5 | 14/04/15 | Yue |
| 0.6 | 19/05/15 | Yue |