Problem

After a capture session, even though request and response data is present, displaying the same UI (events) that happened during the capture (and as seen by the business user during the capture) is not a straight-forward task. Some of the information such as the user's mouse movements and button clicks are known only to the browser, and are not present in the capture data. Another challenge is to distinguish the usage of AJAX calls which update specific portions of the UI.

Even parsing and interpreting the HTML is not easy because of the extensive use of Javascript frameworks such as JQuery, which create UI (HTML) elements dynamically i.e. these elements will not be present in the html response data, but the presentation functionality is actually present in the included Javascript files.

In spite of the above challenges, there are many advantages in being able to display the UI corresponding to capture data, so that an end-user can visually see the sequence of business events that resulted in the capture data.

Solution

The challenge posed by the Javascript usage can be overcome by using frameworks such as HTMLUnit which internally use a Javascript engine and are capable of generating the UI elements from the Javascript code. The generated HTML can be parsed and used to re-create the capture UI events.

Approach

The approach is to parse the response data and "intelligently" re-create the UI events.

1. Start with the first item in the sequence of request/response pairs of data, which will be the main HTML page. Modify all the URLs in this page such that the URLs corresponding to that server and port are now modified to point to the clockreplay resultdisplayhandler server. Using the browser information from the capture data, use the appropriate browser such as IE, Firefox, Safari, etc. and load the HTML (page) in the corresponding browser.

2. When the previous page automatically attempts to load the image/js/css files for that page, the request will be redirected to the clockreplay resultdisplayhandler server (as per the URL changes made in the previous step). Return the responses from the previously stored response data. At this point of time, the initial page is now fully displayed in the browser.

3. Move to the next request/response in the capture data and "intelligently" parse the previous HTML page to identify the source of this new request. If the new request is another HTML request/response, then we also need to identify whether it is an AJAX call or a complete replacement of the entire page. The latter case can be handled as specified in the first step above. The AJAX call scenario can be handled by simulating the AJAX call from the browser, by adding a Javascript method to the previous HTML page, which will be triggered after a certain timeout, say 5 seconds (after the page loading is completed).

Design

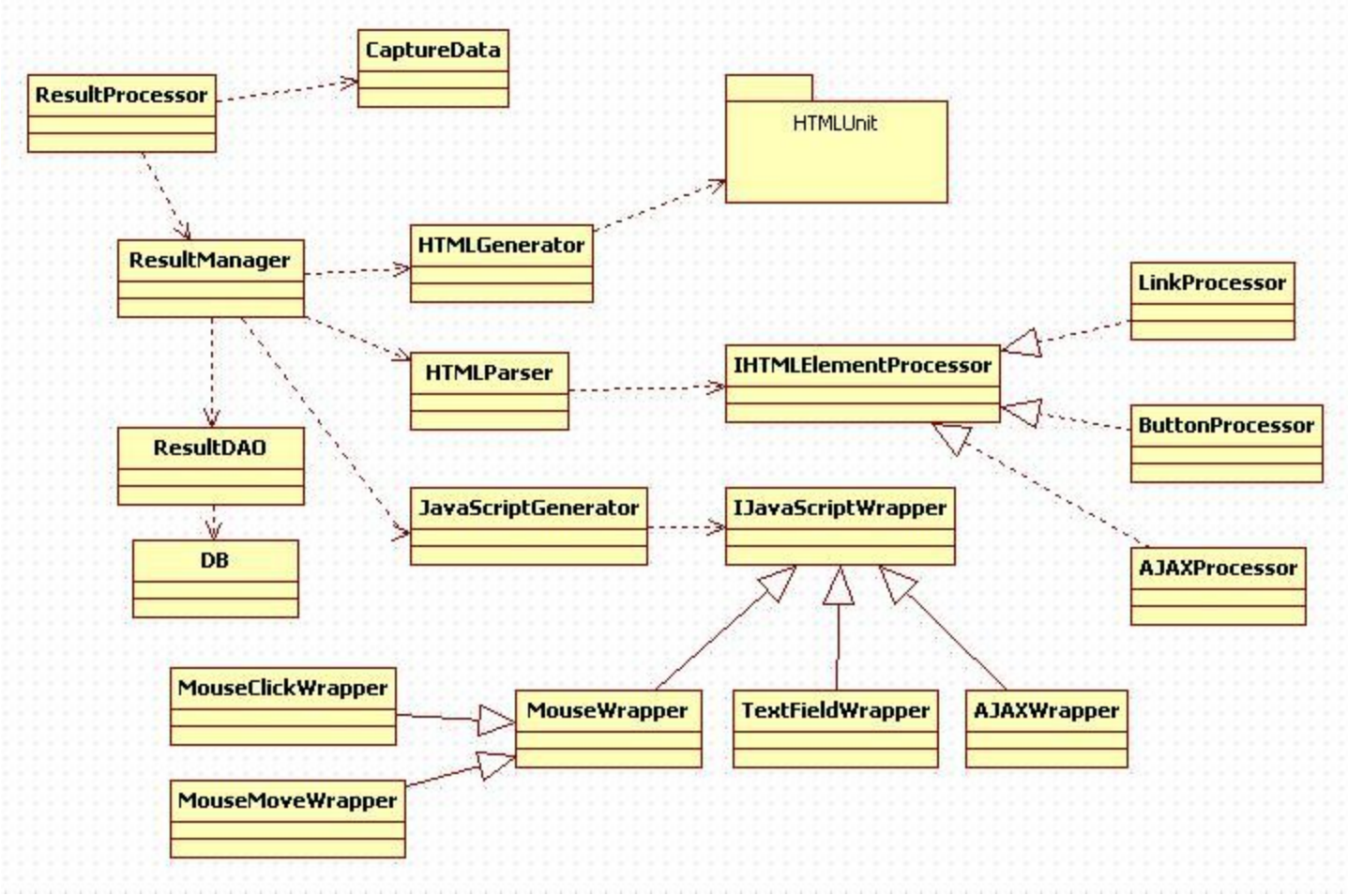
The solution consists of 3 separate pieces:

1. The processing of request/response data and generation of Javascript code which simulates business user actions.

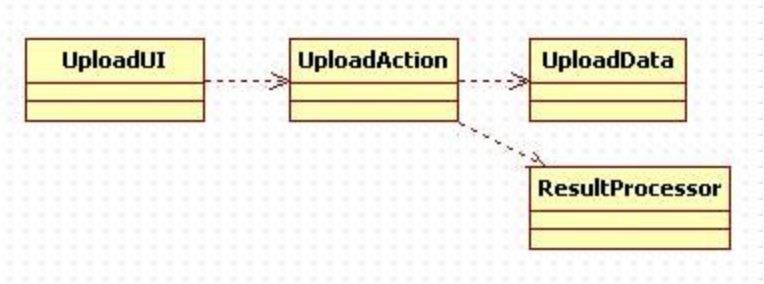
2. The display handling part which will perform the actual display as part of the administration web console.

3. An "upload" feature which allows new capture data to be "uploaded" to the server. This will trigger the processing functionality (item 1 above) which will process the data, prepare it for display, and store it in the DB.

Result Processor Design:



Upload Design:



:

Special Cases

1. Some HTML could be "hidden" (such as Tabs within the page), but still present in the HTML that we parse. Identifying the hidden elements and handling them appropiately requires complex logic to be implemented.

2. Some HTML elements could be generated dynamically via Javascript as a response to a button click or an AJAX call. Handling these cases is also not straigh-forward.