Xenbase Image Scraper Design Document

By: Chris Jarabek (cjjarabe@ucalgary.ca)

Introduction

This document outlines the design and development of the Xenbase Literature Image Scraper. This feature set allows the ability to import and attribute images from literature sources and display them within Xenbase. This feature set builds upon the Xenbase image servlet that was re-factored prior to this project. Please note that this document was written prior to and during the original implementation of this utiltity, some of the implementation details of this document may differ from the final product. **Additionally, this document outlines the design of the utility as it is implemented in Xenbase, not, as it has been implemented for standalone use by external parties.**

Design

In order to understand the design decisions described in the following section, the data structure of a typical journal article must be examined, we will be using a Developmental Dynamics article for this example. From a given article within Xenbase, the "article link" directs the user to a framed Dev. Dyn. webpage containing the article abstract as well as the meta-data related to the article. This page also contains a link to the full article text. The full article page contains the text of the article, as well as thumbnails of the images and their associated captions interspersed throughout the text. The text also contains numerous intra-document links as well and a large amount of markup used for aesthetics and formatting. The full scale versions of the images are not accessed through a standard HTML anchor, but rather through a custom embedded Java script, which causes the full sized images to appear in a new window when the corresponding thumbnail is clicked. The full sized images are also available to users in a "magnified" version (accessible from the new window). Fortunately, both of these images are bound to a standard URL of the following form.

`http://www3.interscience.wiley.com/cgi-bin/fulltext/ZZZ/nfigNNN`

Here, ZZZ is Dev. Dyn. custom article ID and nfigNNN corresponds to the filename of the image where NNN is the number of the image on the page. The pop-up image window also contains the image caption and no other text. This makes the page ideal for scraping the caption text. However, the URL for this page is hidden by the Java script and therefore cannot be accessed directly.

Given that the data structure of the Dev. Dyn. webpage is, at most, shared with other journals hosted by Wiley Interscience (the parent company of Dev. Dyn.), the code for scraping the Dev. Dyn. webpages would have to be unique to this journal. Additionally, it was assumed that other journals which may, in the future have their images extracted, do not share a common data structure with Dev. Dyn. Therefore, one of the first decisions made in the design phase was to use a "plug-in" style approach for the application. The idea is to create a single Java class that

would be responsible for scraping the data off of the Dev. Dyn. page. This class could be plugged into the desired work-flow, by itself, or alongside similar classes built specifically for other journals. Each plug-in module would essentially be a black box having common generic input and output, and the inner workings of the plug-in would have no bearing on the context in which it was used. This means that plug-in modules would be reusable in the future and could be easily copied to make new plug-ins.

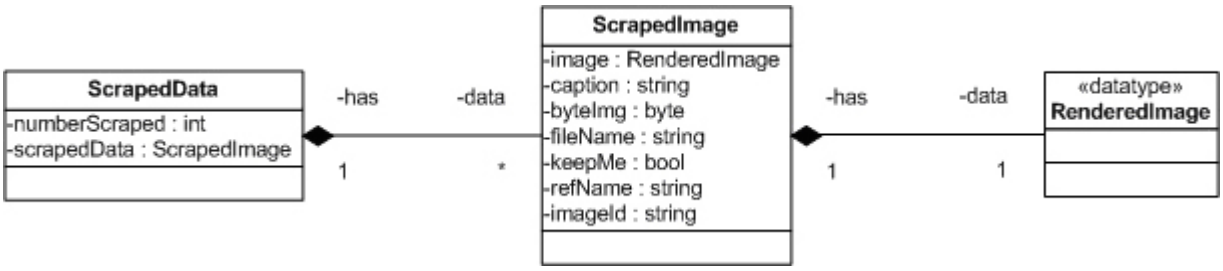

Fig 1. UML Class digram showing the relationships between the JAI RenderedImage objects, the ScrapedImage objects and ScrapedData objects.

The first step of the design phase was to determine the minimum inputs and outputs for a plug-in. The minimum input is a single URL pointing to the full text of a journal that the plug-in is capable of interpreting and processing. The minimum outputs are lists of images and captions. In order to handle the output of the system in a more elegant way, two objects were created called ‘ScrapedData’ and ‘ScrapedImage’ (Fig 1.), which are complex data structures composed of multiple primitive variables. ScrapedImage is an object consisting of numerous variables; but most importantly, a digital representation of the image and the image caption. ScrapedData then contains a list of objects of the type ScrapedImage. Therefore, the plug-in used for scraping a webpage need only take a URL, and return a ScrapedData object. This will provide the main application with all of the images and captions, as well as any supplementary information necessary to complete the importing process. The main application (mentioned above), refers in this case to the J2EE/Struts application frame work used to build Xenbase. To give the user a logical flow of control, a Struts extension called Struts Workflow was used. This extension helps develop wizards which guide the users through the various steps of a process. This extension has already been used elsewhere in Xenbase, specifically, when a user creates a new Xenbase account or lab. Once the workflow is developed, adding additional steps is not time consuming. The workflow for this application was designed with three steps: setting the data source, choosing images / modifying image captions, and a final confirmation step. Upon completion of the workflow, the images are inserted into the database and are correctly attributed to the article from which they came. Leaving the workflow returns the user to the source article whether or not the scraping process was successful (this is described in greater detail in the following section) (fig 2).

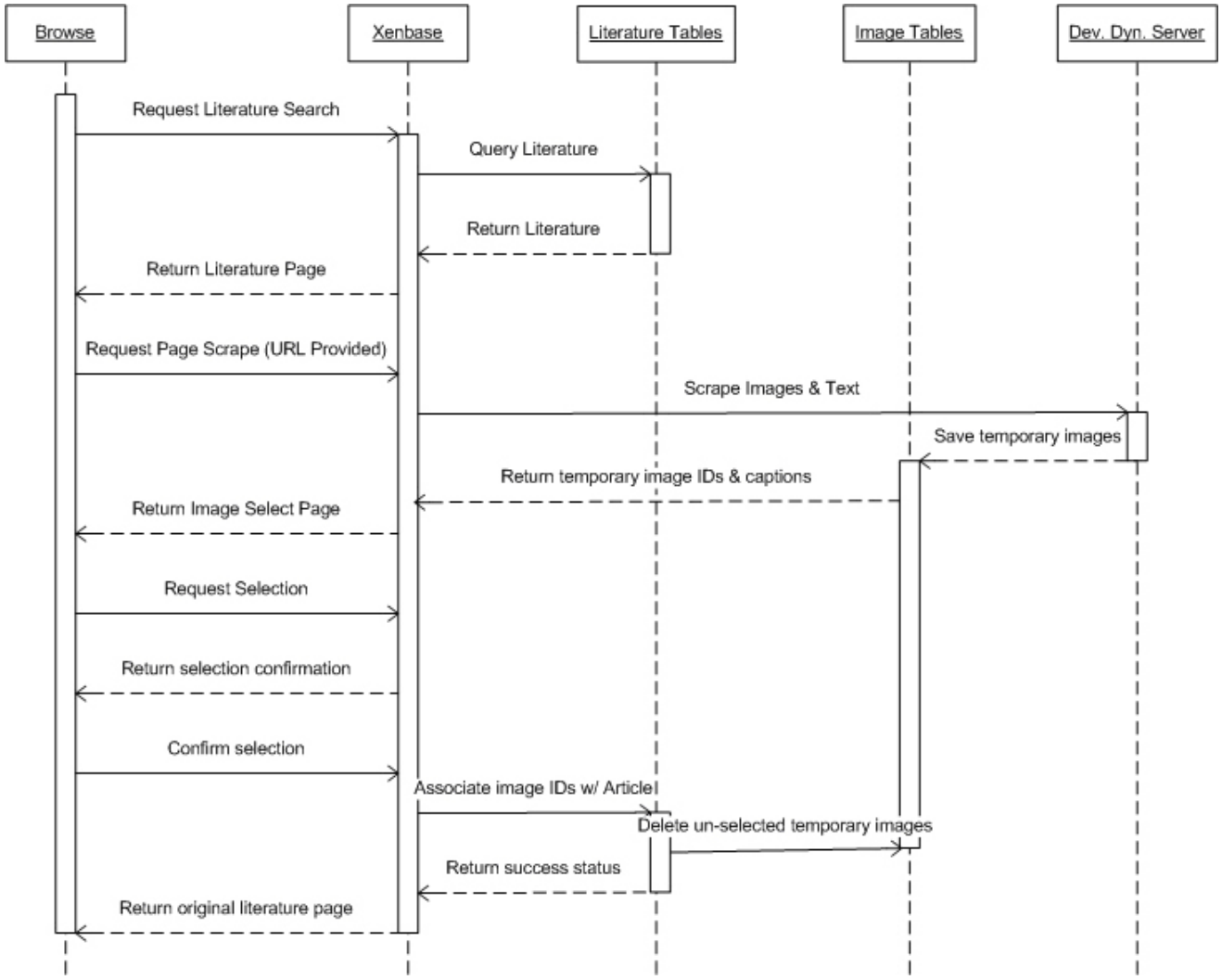

Fig 2. UML Sequence diagram showing order of action between various data entitles when scraping images.

Implementation

The implementation of the plug-in module was by far the most time consuming aspect of the development process. A Java library was needed with the functionality to establish a persistent HTTP connection to the Wiley Interscience web-server. Due to complexities in the way the Wiley server handles cookie management, the standard library used for this functionality (java.net) was inadequate. Therefore, the library that was used was the apache.commons.httpclient library. This library allows for real-time control of the HTTP connection, and additionally, is capable of performing automatic cookie management. This allowed for a graceful handling of the HTTP hand-shake.

A key problem that exists with this library is that one of the cookies sent by Wiley had a date/time stamp that is incompatible with the ISO standard format upon which the Apache

library was built. The only way to overcome this problem was to import the entirety of the apache.commons.httpclient source code into Xenbase and modify the source to accept the cookie date format that was being sent by the Wiley server. Once this was done, images and text could be successfully scraped from Dev. Dyn. webpage.

Once the contents of a given URL could be scraped, the remainder of the plug-in was developed incorporating all of the necessary application logic. The basic flow of application logic for the plug-in is as follows: given that all of the images have URLs as described in the previous section, the program simply generates URLs for figures (i.e. ../mfig001, ../mfig002). Each URL is scraped and this process continues until a URL is generated which when scraped, produces no data. This is assumed to be all of the images in the article. Once all of the images have been scraped, the textual body of the article is scraped and the captions are parsed out matching on character strings that are unique to each caption. This data is then formatted and packaged in a way that is consistent with the ScrapedData object described above, and this object is then returned to the main application.

The main application, as described above, was originally based on the Struts Workflow. However, this project has been unsupported for some time, and so we implemented a 'pseudo-workflow' that simply uses a linked series of struts actions with no external libraries. When a user is on the Xenbase page corresponding to the article they want to scrape, they are presented with a link to import images. This will cause the user to enter the workflow. Upon entering the workflow, the user is presented with a text box containing the URL of the article in question.

When the user moves to the next step of the workflow, several things happen. The plug-in module is run with the URL provided. Following this, the ScrapedData object is returned and the images are inserted into the database. The images are inserted into the database at this stage for two main reasons. First, by inserting the images into the database the image servlet automatically handles the creation of image thumbnails. Second, when the thumbnail images are viewed in the subsequent steps of the workflow, having the images already in the database allows the images to be managed using the image servlet. This allows image handling in this portion of the application to be consistent with how images are handled throughout the rest of the Xenbase application. The downside to this approach is that it generates wasted overhead by inserting images which the user may not then choose to import (and that will ultimately be deleted). It also means that canceling or breaking out of the workflow requires the system to delete these images out of the database. After the images are inserted into the database, their image IDs are returned to the workflow controller so that the correct Xenbase image filenames can be constructed. The user is then presented with the second step of the workflow which contains the scraped images and their corresponding captions in a modifiable textbox, as well as a checkbox allowing the user to choose whether or not they want the given image to be imported.

Once the user has made the desired changes and selections, they can move to the final step of the workflow which simply presents the user with a list of images they have chosen to import, the user can move back in the workflow and change their selection if desired.

When the user finishes the workflow, images that were not selected are deleted from the database, and the selected images are attributed to the source article using the attribution tables.

Finally, the user is forwarded to the article from which they entered the workflow, with the newly added images attached to it.

Attribution

The Attribution links for the Image scraper are as follows:

When papers are loaded into Xenbase, they are attributed to their publishing organization using the 'Copyrights' attribution. When Images are scraped from an image, they are attributed to the source paper using the 'Published' attribution.

Current Journals

The journals supported by the Xenbase Literature Image Scraper are:

- Mechanisms of Development
- Development
- Developmental Biology
- Developmental Dynamics
- Cell
- Developmental Cell
- Current Biology
- PNAS

Classes

Because some the Journals covered by our scraper are published by the same organization, the scraper-plug-ins are actually unique to the publishing organization they are listed below:

Class:org.xenbase.scraper.AbstractScraper

Use: Used by all child scraper classes

Class:org.xenbase.scraper.CurrBio_DevCell_Cell

Journals: Current Biology, Developmental Cell, Cell

Publisher: Elsevier (published via Science Direct)

Class:org.xenbase.scraper.DevDyn

Journals: Developmental Dynamics

Publisher: Wiley Interscience

Class:org.xenbase.scraper.Development

Journals: Development

Publisher: Company of Biologists

Class:org.xenbase.scraper.MechDev_DevBio

Journals: Mechanisms of Development, Developmental Biology

Publisher: Elsevier (published via Science Direct)

You probably noticed that there are two different scrapers for the Elsevier journals, this is because the Journal pages for MechDev and DevBio are different from the pages of the other Elsevier other journals warranting a different scraper. Admittedly they are pretty similar, but different enough to have separate classes.