Matt Arriola

**ScrapeDemo**

Software design document

Date: 7/20/2015

# Table of contents

1. Introduction
   1. Purpose
   2. Scope
2. System overview
3. System architecture
   1. Architectural design
   2. Decomposition
      1. Abstract scraper class
         1. Internal design
         2. Interface to derived classes
      2. Derived scraper class
         1. Interface to parent scraper class
         2. Interface to client program
         3. Steps
         4. Error handling
      3. Scraper manager
      4. Scraper designer

# 1. Introduction

## 1.1 Purpose

This is a demo of what a solution for AutoNation’s screen scraping system might look like. The main problem with the current system is that manufacturers change their web site designs from time to time, breaking the screen scrapers AutoNation uses to retrieve information about parts. There isn't much that can be done about this, except to provide a framework that makes it easier to automate the process and swap scrapers in and out when needed.

## 1.2 Scope

My solution aims to ameliorate the problem by providing an abstract class for writing screen scrapers that provides a structured way to automate the process. By automating as much of the screen scraping as possible obviates the need to write large amounts of code just to navigate the target site, and should make it simpler to design new scrapers.

# 2. System overview

A scraper defines the steps of the scraping process, which correspond to pages to load as the scraper navigates the web site. Each step is executed in order, although it is possible to move to other steps arbitrarily. Page elements needed by the scraper can be made available to it by declaring them in an XML file that is loaded by the scraper when it is instantiated. This file is generated by a designer tool included in the demo.

# 3. System architecture

## 3.1 Architectural design

The demo consists of five parts.

* The client program
* The abstract Scraper class
* The specialized derived scraper classes
* The ScraperManager
* The scraper designer

The client program makes use of specialized screen scraper objects to retrieve information from certain web sites. The specialized screen scrapers derive from an abstract Scraper class which provides common functionality and are compiled to separate assemblies. These assemblies are loaded by a ScraperManager object, which provides scraper instances to the client program. The scraper designer tool is used to select elements that contain information needed by the scraper code.

# 

**Figure 1.** The class hierarchy of the demo solution.

## 3.2 Decomposition

### 3.2.1 Abstract scraper class

The Scraper class is the parent class for all scrapers. It defines common behavior for all scrapers that should make it easier for developers to design new scrapers when old ones are broken by site design changes.

#### 3.2.1.1 Internal design

The Scraper class defines a very simple state machine. The details of this are mostly hidden from the derived class and don’t necessarily need to be used.

A list of functions that define the steps of the screen scraping process is maintained. They are executed in the order added. Perhaps if more complex logic is required, this could be changed to an associative array to allow it to perform different actions based on some result from the web site.

#### 3.2.1.2 Interface to derived classes

The absolute minimum a derived class needs to do is to define a set of methods to be called as the scraper navigates a manufacturer web site and attach them to the state machine in the order they’ll be visited with the AddScraperStep method. This method returns that step’s state machine step as an integer; should the scraper have to revisit a step, it can do so by passing that value to SetState.

The Scraper class exposes a WebBrowser object to its derived class through a read-only property. Navigation is done through its Navigate method, and the DOM is available through its Document property. The Scraper class attaches its own event handler to the browser’s DocumentCompleted event so that the state machine is automatically advanced as new pages are loaded and the derived class moves on to the next step.

An error handler is exposed through the ScraperError event, which takes a delegate to a function which takes an error string, and optionally, an uncaught exception thrown by the scraper. If the client doesn’t attach its own error handler, the error string is also available through the ErrorString property, and the exception is in ScraperException.

As mentioned earlier, the derived class can use the SetState method to move arbitrarily to another step if necessary. Also, in the event of an error condition, e.g. if the site design changes, it can use the SetErrorState method to halt execution of the scraper. This might be a useful place to add some code to the abstract Scraper class to report a failure to the system administrator, allowing work to begin as soon as possible on investigating the cause of the failure and writing a new scraper.

Web site script errors are suppressed by default, but they may be desired during development and debugging, in which case the derived class constructor may indicate they be enabled to the parent class constructor.

### 3.2.2 Derived scraper class

Screen scraping functionality is provided by creating a class that derives from the abstract Scraper class. Derived classes should live in the Scraper namespace, as this is where the ScraperManager expects them to be.

#### 3.2.2.1 Interface to the parent Scraper class

All scrapers must take at a minimum one string parameter in their constructor. This parameter contains a path to the scraper definition file and should be passed to the parent constructor.

#### 3.2.2.2 Interface to client program

All scrapers must define their own Scrape method as an interface to the client program. This method should perform any setup work necessary before calling the parent class’s Start method to direct the state machine to execute the derived class’s first step. The Start method can only be called once; additional calls will throw an InvalidOperationException.

#### 3.2.2.3 Steps

A step is a method for processing a single document. To register a step, a delegate to it passed to the AddScraperStep method. The delegate defines a method that takes no arguments and returns true for the scraper to advance to the next step. If the step requires reloading the page with a new query, it can return false to redo the current step.

#### 3.2.2.4 Error handling

A step may arrive at an error condition in a few different ways. The step may also choose to set the error state itself when HTML elements it expects are not present, indicating a site change or an error condition on the web site itself.

### 3.2.3 The scraper manager

The ScraperManager class is responsible for locating and loading external assemblies that contain usable Scraper classes. It does this with a scraper definition XML file that looks like so:

<?xml version="1.0" encoding="utf-8"?>

<externalScrapers root="..\..\..\">

<assembly path="RedditKarmaScraper\bin\Debug\RedditKarmaScraper.dll">

<class name="RedditKarmaScraper"/>

</assembly>

<assembly path="PsecuAccountScraper\bin\Debug\PsecuAccountScraper.dll" className="PsecuAccountScraper"/>

</externalScrapers>

The root <externalScrapers> element defines a root path where assemblies are located and a list of <assembly> elements pointing to them. Each <assembly> element defines one or more classes that are exported by that assembly.

Once the ScraperManager’s shared instance has been obtained, scraper classes can be instantiated by name through the GetScraper method, for example:

Scraper fordScraper = scraperManager.GetScraper(“FordScraper”)

Should a site design change force the development of an entirely new scraper, a new assembly can be made with the same name and simply dropped in place of the old one, or the scraper definition XML file can be pointed to the new one instead.

### 3.2.4 The scraper designer

The designer window contains an embedded web browser which can be used to navigate the target web site. Individual HTML elements are highlighted as the mouse passes over them, and they can be selected to record attributes that may be used to identify them on a page. This information is organized by step and is saved to an XML file which is loaded by the scraper when it is instantiated. As the scraper executes its steps, the elements defined for that step are bound to the IDs specified in the XML file and can be retrieved by the scraper code from a dictionary.

Should the site design change and break the scraper, the fix could be as simple as running the designer again and selecting the desired information on the page. If the page markup has changed, but the information has remained the same, the scraper definition can be modified to reflect that change while pointing the new elements to the same IDs so that the process is transparent to the scraper code.

