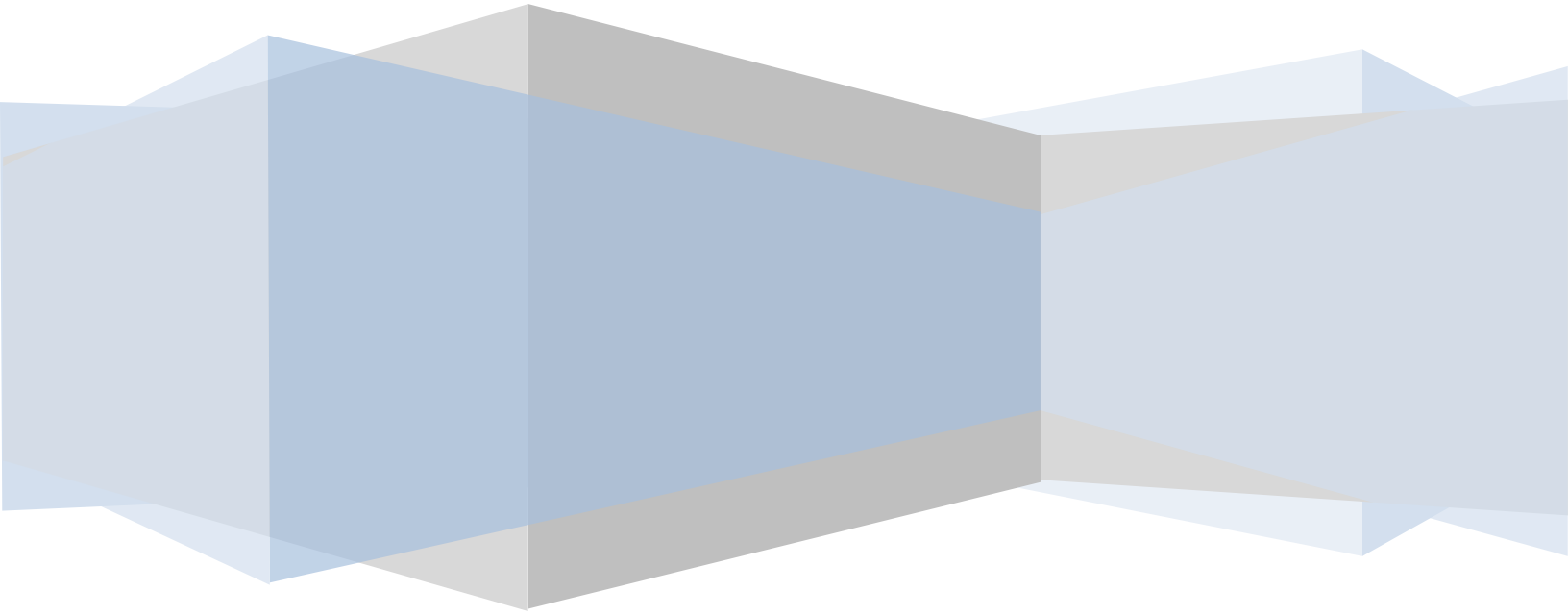
Web Crawler

David Tee

Computer Science Department
Cal Poly, San Luis Obispo

December, 2006



Contents

1. Project Abstract

The World Wide Web is inundated with information, and a lot of companies spend a good deal of their resources on automated extraction of publicly available information from various websites. Crawling the Web and extracting the information is a large part of many major dot com companies such as google.com (search engine), pricegrabber.com (price comparison), gift.com, glimpse.com, etc.

For many companies, being able to extract information from web sites is vital to their business. This project provides a tool that can be used by small companies, and big companies alike to systematically gather data from the web. It provides features not available in alternative crawling solutions.

2. Background Information

My interest in this project started when I was in high school. Back then, I liked creating websites and one of the problems that I ran into was keeping the site up-to-date regularly without putting a lot of man-power behind it. The specific website that I wanted to create was a lyric website. Such a website requires that I build a database that contains the lyrics to all the songs that have ever come out (easily over one million songs). To build this database, I needed a program that can crawl a few public lyrics websites and extract the song information, such as lyric, title, artist(s) and album(s). I did a lot of research looking for a program that could do what I wanted, and came up empty-handed. So, I decided to write one, It would be the first home-grown web crawler that could crawl and extract any information from a website. The data extraction of the crawler would also take into consideration the relationships among web sites that have been crawled (based on the URL path it crawled).

Initially, I wrote a crawler to crawl one lyric website that was pretty easy to crawl (the lyrics can be located by incrementing the id field of the URL link). Then I started modifying the crawler to crawl several different lyrics websites, and the medication to the program for each site's crawling proved time consuming and turned out to be very buggy. So, I modified the crawler to accept some kind of configuration so that I wouldn't have to reprogram it every time.

Another problem when it comes to crawling is that it may take more than days or weeks to completely crawl one website. So, the crawler must support some kind of state to keep track of the urls that it has already crawled and retain the data that it has extracted. That way, when a crawl is interrupted for any reason, the crawler could resume from where it left off.

I also want the crawler to keep track of the state between different crawl sessions. Usually, a website is crawled more than once to keep the data up-to-date, by running the crawl weekly or monthly. The crawler must be able to distinguish between the newly extracted data and data from the previous crawl session.

The most important part of the crawling process is the data extraction, also known as the data scraper. I needed to find a way to extract desired information some files given a particular setting/pattern. The algorithm would be able to parse a file for the desired information given some kind of template that describes the pattern. I spent years developing a flexible configuration for the scraper, using the

technique similar to XML DOM parsing with XPath. The scraper in this crawler is a regular expression based data parser. The scraper parses a file using the *tag library* (tree of *tags*). A *tag* contains a start tag and end tag and other information that describe how to parse the ext for desired data. The parser is described in detail in the Scraper Manual attached.

3. Problem Statement

To my knowledge, there is no off-the-shelf software that performs the task of web crawling and data extraction. The product of this project, the Web Crawler will be the first configurable crawler that can crawl automatically any web sites, extract the data and store the data extracted into a database.

Functional Requirement

The crawler must be able to download a webpage

The crawler must provide a scraper capable of extracting data from a string

The crawler must be able to set delays in between downloading a webpage

The crawler must be able to download URLs simultaneously (downloading a webpage is slow relative to other functions, thus must be threaded to increase performance)

The crawler must be able to maintain the state of crawling

The crawler must be able to save the extracted data to a database and to a text file

The crawler must be able to take a crawl configuration to crawl a different website

The crawler must keep track of statistical information (total URLs downloaded, average speed, etc)

The crawler's crawl process must be able to support start and stop operations

Non-Functional Requirement

The Program must not take up a lot of memory, as well as system resource

The program must able to run for hours without crashing

Data Requirement

There are sets of entries that will be stored in a configuration file which contains instructions for crawling a particular website (crawl project file, file extension: sproj).

The data extracted during the crawl process must be consistent with the data that we wanted to crawl. The relationship between the crawled data should be maintained.

User Interface

It is desired for the Web Crawler to have a graphical user interface that is user-friendly and intuitive. The user interface is documented in the attached documents: The Crawler User Manual and the Scraper User Manual.

(View attached document on Crawler User Manual and Scraper User Manual)

System Requirements

The project is to be developed using Microsoft's .Net framework version 2.0, thus the program will need to run on a machine that has the .Net 2.0 framework installed. This is typically Microsoft's Windows system, but any system that can host .Net 2.0 would be able to run this program.

The system also needs to be installed with SQL Server Express 2005 (1) or Microsoft SQL Server 2000 or later version. This is to support the database access feature of the project.

The following table describes the recommended platform for running the Web Crawler:

Item	Description
Processor	600-megahertz (MHz) Pentium III-compatible or faster processor; 1-gigahertz (GHz) or faster processor
Framework	SQL Server 2005 Express Edition, and.NET Framework 2.0 (SQL Server expression is not required if database is not needed)
Operating System	Windows XP with Service Pack 2 or later Microsoft Windows 2000 Professional with SP4 Microsoft Windows 2000 Server with Service Pack 4 or later Windows Server 2003 Standard, Enterprise, or Datacenter editions with Service Pack 1 or later Windows Server 2003 Web Edition SP1 Windows Small Business Server 2003 with Service Pack 1 or later Vista Home Basic and above (SQL Express SP1 and SQL Express Advanced SP2)
Memory	192 megabytes (MB) of RAM or more; 512 megabytes (MB) or more recommended
Hard Disk	Approximately 500 MB of available hard-disk space for the recommended installation. The program itself requires only about 20 MB, but the database will grow as you crawl more sites
Display	Super VGA (1,024x768) or higher-resolution video adapter and monitor
Other Devices	A mouse or any compatible pointing device

4. Solution

Design

There are three separate components of the crawler: the core library (contains the functionality), the Application UI (user interface codes) and database. The core library will compile to a Windows library file (*.dll) file that can be reused by User Interface implementation. The Application UI will compile to a Windows executable file (*.exe) that can be executed in any Windows system with .Net framework version 2.0 installed. The database is Microsoft SQL Server 2005 (express), and it is needed to save the data to database, or run the crawler in database mode.

Database Scheme

"Figure 1: Data Scheme" is the underlying database scheme that the project depends on. In this ER –Like Diagram, you can see the properties of the crawl solution (crawl project data table). All of the table

fields are self explanatory. The ER diagram is designed using Microsoft Studio's Dataset Object. The naming convention for the data columns are as follows: Any field with the word Date is a Timestamp, any fields that starts with "Is" is a Boolean, any other is either text or integer. Detailed information about specific database field is provided as SQLServerScript.sql and also as ScraperDB.xsd file.

Table Name	Description
Cron	Contains Unix CRON job scheduling information. (The scheduling is not yet integrated with the crawler)
CrawlProject	Contains information about the crawl project (project name, description, etc.)
DataType	Contains the dynamic code that will be loaded and compiled when the crawl solution is loaded. The compiled object will be executed against the extracted Data.
TagLibrary	Contains Tag information for scraping
Project	[Scrape] Project
Crawl Client	Not yet implemented – This table supply information about a crawl client for when the crawler works as a crawl server to distribute the crawling
Url	Store Url information: the url information keeps track of the state of the crawler since it contains crawled, or not yet crawled urls.
UrlStatus	Stores URL status information: Ready, Error, Crawling or Assigned
Proxy	Stores Proxy Information
ProxyStatus	Stores Proxy Status information similar to Url Status. (Good proxy, Bad Proxy, Assigned Proxy)
Project_Url	Contains information about of the [Scrape] project(s) that should be used to extract the data from the downloaded content of the URL
Link_Mapping	Contains information about which [scrape] project(s) should be used to extract the information of the URL extracted.
Project_Sample	This table stores the sample URL and context that is used to test the tag libraries.
Url References	Contains the tree structure of the URLs crawled

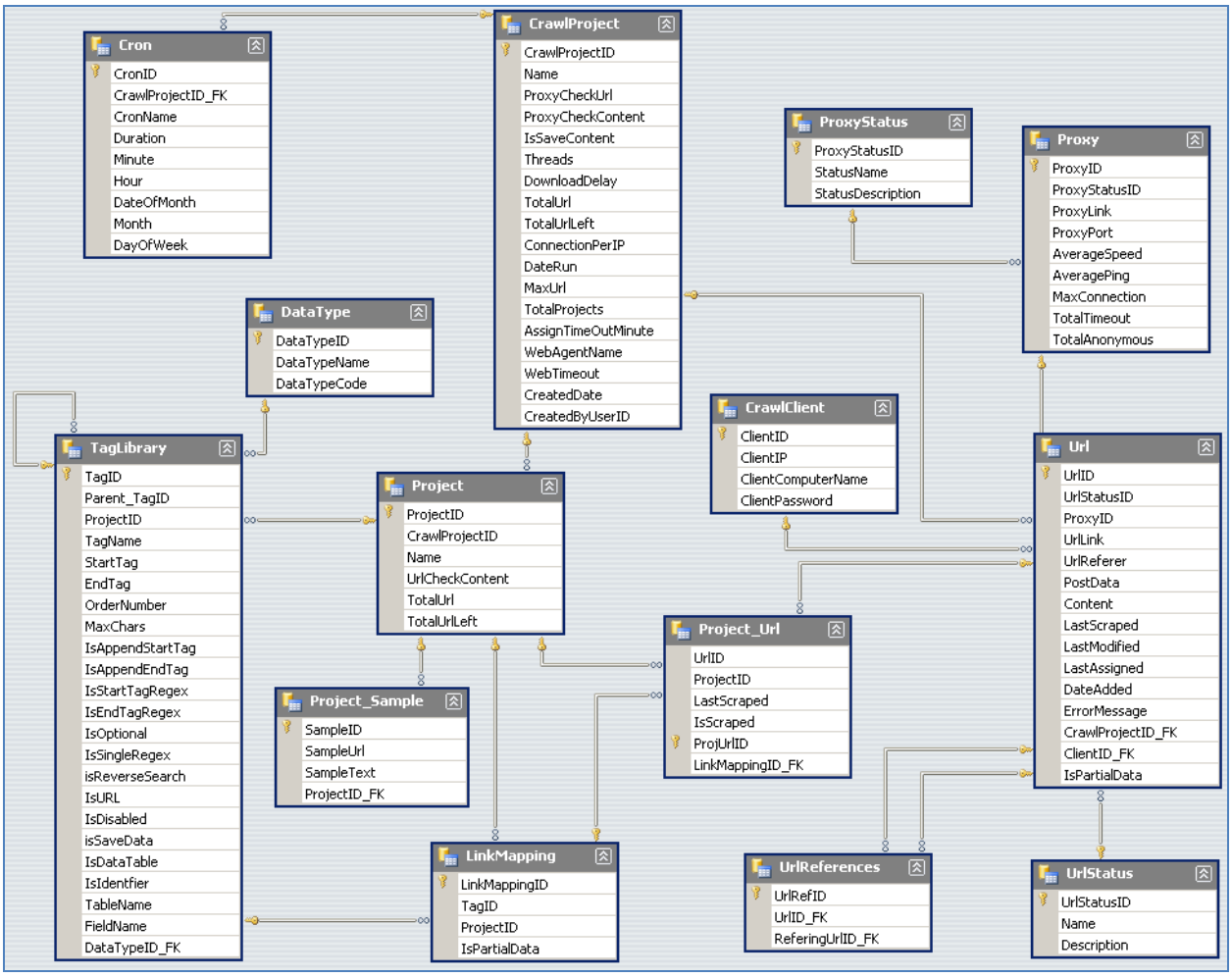


Figure 1: Data Schema

Library

The functionality of the crawler will be separate from the User Interface (ScraperLib Visual Studio Project). Within the library the crawler is divided into several different components that interact with each other during the crawling process. For every table listed in the Data Schema, there's a corresponding class that manages the data, for example, Crawl Project, Crawl Project Manager, Link Mapping Manager, Url Manager, Proxy Manager, Cron Manager, etc. The tables that contain Status information are created as Enumerable Class.

Crawl Manager

This class manages the creation of crawl project in a database environment.

CrawlProject

The crawl project class is the core class of the whole program. Crawler class contains the crawl project configuration, Crawler, Scraper, Downloader, data adapter, URL Manager, Proxy Manager, Schedule Manager, etc. Creating a new crawl solution is just a matter of creating a new crawler class. Loading and

saving a new crawl solution is just a matter of creating a new data adapter class (FilescraperDataAdapter is the default).

Cron [Schedule] Manager

Schedule manager is responsible for managing schedules. This manager will check for validity of a newly added schedule, and contains a function to return the next scheduled date. It took me a while to select what type of scheduling scheme I wanted in the program, but after researching various database schemes for scheduling, I found Cron to be the best. Unfortunately, .Net framework does not provide any kind of Scheduling Library; I decided to implement a Cron based on my understanding of how Cron Works. (Cron Datatable)

[Scrape] Project Manager

This Manager manages a list of [Scrape] Project in a crawl solution.

Url Manager

Url Manager is responsible for managing a list of URL s and their state in the crawl solution. Url Manager will validate the any urls added to the list, and also contains function to get the next URL that should be crawled. (Url Manager can effectively handle the crawling algorithm, depth first or breathe first by returning the next URL to be crawled based on the date added). (Url Data table)

Proxy Manager

Proxy Manager works the same way as URL, except it manages a list of proxy. The Manager will validate the proxy, and has function to return the next available anonymous proxy that the download/network manager can use. (LinkMapping Datatable)

Download/Network Manager

This download/Network Manager is responsible for downloading WebPages. This class will be threaded to allow simultaneous download. The goal of this Manager is to be able to download any URL given. Now, url is a Uniform Resource Locator, which allows different protocols, such as file:///, http://, https://, ftp:// etc, and the Network Manager is divided this way so that more protocols can be added in the future.

Link Mapping Manager

This manager manages “Links” that defines which urls the crawler should recognize as the next set of links to crawl from the result of a Scraper. A link connects a URL tag node to a project, all the resulting URL will be downloaded and scraped the project specified in the link.

Crawler

The crawler manages the crawl algorithm; it handles starting a crawl and stopping a crawl process. In database mode, the crawler will download and scrape a set amount of URL, save the data to database, check out more urls from database, and continue that process until there are no urls left to crawl.

Crawl Helper

Crawl helper contains static functions that help the crawler with the crawling process, such as building the data relationship of the crawled dataset with in a crawl session.

Scraper

The scraper class is responsible to handling data scraping based the information provided to it (The web content, URL, and the tag library).

Log

Log is a static class that Managers the output of the logs generated by various code. Debugging, Error, Info, such logs that are generated from various part of the program is outputted to console by the log class. The output includes the time stomp, and the name of the object that generated the log.

Data Adapter

The library must contain functions that can load and save the crawl project from either a file located in the computer or from database. The crawler must also be able to switch between saving the data crawled to file or to database. In order to do that, Adaptor Design pattern is used to simply adapt the file component (if saving to file), or the database component if saving to database. FileScraperDataAdapter contains functions necessary to save the crawl solution and its data to a file located in computer. The SQLDataBaseScraperAdapter contains functions necessary to save to SQL database. Both these classes inherit ScraperDataAdapter, which is an abstract class that contains information about the crawl solution that will be saved. If new functionalities such as saving to an Excel file, or to text file was to be added, I can simply inherits the ScraperDataAdapter.

SQLScarperDataAdapter

This class will use SQL Client library of .Net 2.0 to save the data to database. This class will be written so that a new database system (such as oracle) could be easily implemented to the system.

FileScraperDataAdapter

This class will simply export the dataset to file.

SQLHelper –DDL

This class generates SQL codes necessary to generate a database table based on the dataset.

Data Type Manager

This class manages various data types that a crawler may use.

Data Type

Data type contains the information about the data type, such as the name of the data type, the dynamic code that will be compiled dynamically as the crawl project is loaded, and a reference to the compiled object.

DataRefiner (Solution)

This class serves as an abstract class for any data refiner written by the user. The class contains only one function that must be over-ridden, and it is to format the string.

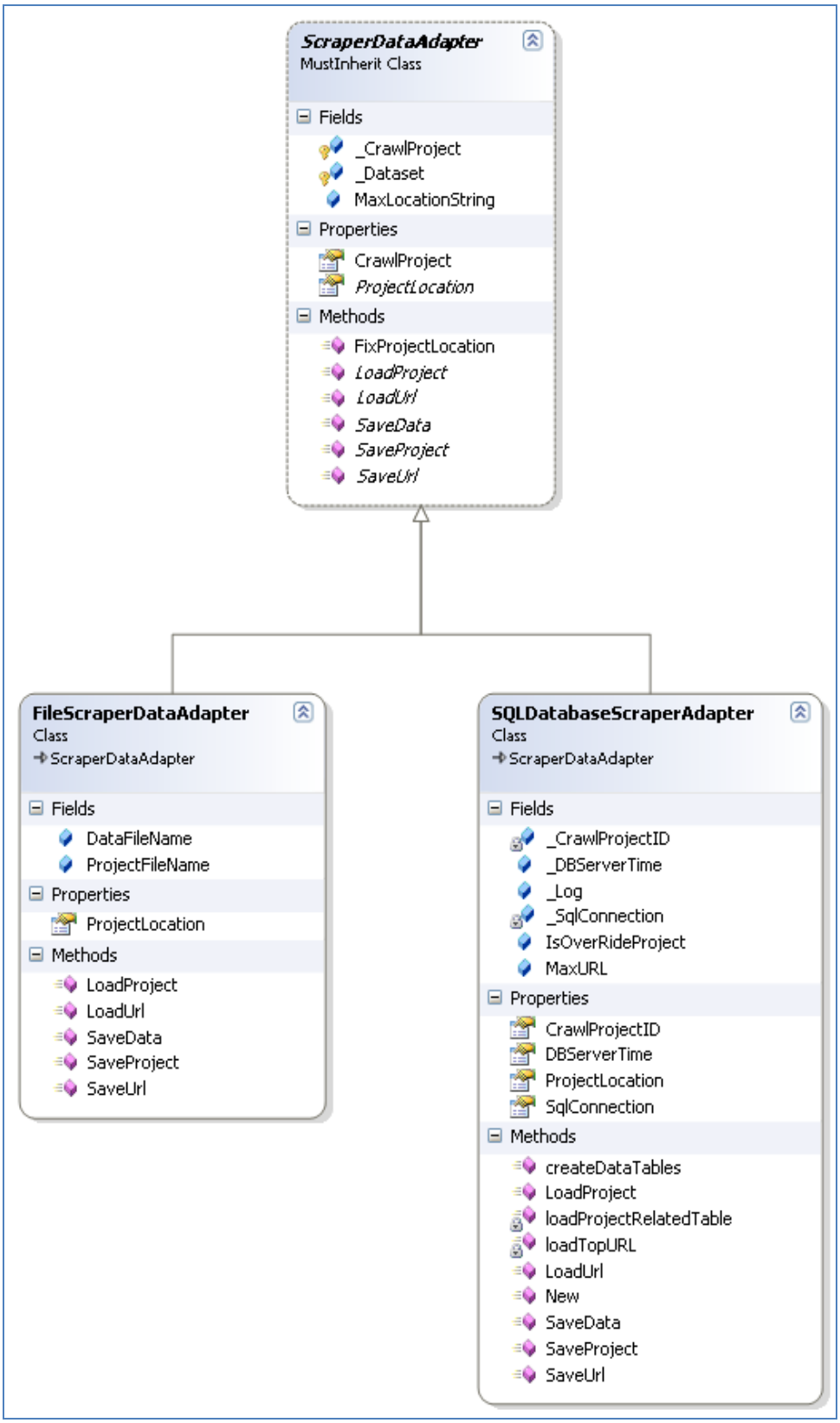


Figure 2: Data Adapter

Component	Description
Extraction component (Scraper)	Extraction component is the heart of the crawler; it accepts definitions for extracting the page's content and scrape/extract the information from the website. (This component is an extension to the Scraper I developed in CSC 200.)
DataTypes	This should be a dynamically programmable component. The job of this component is to further refine the extracted data to a database field type. For example, a website may represent a date in a European time or USA time format and the Datatype component must correctly convert the time.
Network/Downloader	The http downloader accepts an http Uri and downloads the website. This component must be multi-threaded because downloading web pages takes a lot of time relative to the parsing process.
URL Manager	URL manager manages what URL should be downloaded to scrape, and make sure the URL will only be scraped once in the crawl session.
Data Merger	Data merger will merge the appropriate extracted data into a structured data. This component will build data relationship extraction from various web pages.
Database: DDL	This component will generate DDL SQL for creating database tables based on the (scraper) configuration.
Database: DML	This component will update the crawl solution and the extracted data to database. Another job of this component is to identify whether a newly extracted data is new or already exits.

This is a sample of the objects in the Scraper Library:

Application

The application contains the codes for the graphical user interface of the crawler (ScraperApp Visual Studio Project). The application uses the core library for the crawling functionalities.

Rich Text Format (RTF format)

The Rich text box that comes with .Net 2.0 does not provide efficient means of coloring the text. This class helps to generate RTF text based on the text and the color code setting that I want to put on the text. This is essential for efficiently coloring the results of the scraper. (2)

Data Grid View

.Net 2.0's data grid view provides rich features that allow the developer to bind the grid with a data, data table or a list of objects. DataGrid view is used in the project to present the data crawled/scraped to the user.

Implementation

The project is implemented in Visual Basic .Net using Visual studio 2005 and .Net framework version 2.0. The project heavily relies on .Net's Dataset, both strongly typed and not strongly typed Dataset. I used strongly typed datasets whenever I could because it allowed me eliminate careless errors (typo in table name, or data field name) at compile time. (3)

Crawl Project

Crawl project allows the creation of new crawl project. Creating a new instance of crawl project will create all the necessary Managers (such as URL, proxy, Cron, [scrape] project Managers). The Managers will then load up the default properties.

Here's the code for constructing a new Crawl Project given the name of the project.

```
    Public Sub New(ByVal Name As String)
        Me._DS = New ScraperDB
        Me._CrawlProjRow = _DS.CrawlProject.NewCrawlProjectRow

        Me._CrawlProjRow.Name = Name

        Me._CrawlProjRow.ProxyCheckContent = ""
        Me._CrawlProjRow.ProxyCheckUrl = ""

        Me._CrawlProjRow.DateRun = Now
        Me._CrawlProjRow.IsSaveContent = False
        Me._CrawlProjRow.Threads = 1
        Me._CrawlProjRow.DownloadDelay = 0

        Me._CrawlProjRow.TotalUrl = 0
        Me._CrawlProjRow.TotalUrlLeft = 0
        Me._DS.CrawlProject.AddCrawlProjectRow(Me._CrawlProjRow)

        InitManagers()
    End Sub

    Private Sub InitManagers()
        ' Create a project Manager
        ' TODO: Check out builder apttern. (facade)
        Me._ProjectManager = New ProjectManager(Me)

        Me._UrlManager = New UrlManager(Me)
        Me._ProxyManager = New ProxyManager(Me)
        Me._ProjectManager = New ProjectManager(Me)
        Me._DataMapper = New DataMappingManager(Me)
        Me._LinkMapManager = New LinkMappingManager(Me)
        Me._DataTypeManager = New DataTypeManager(Me.Dataset.DataType)
        Me._CronManager = New CronManager(Me)

        ' Build the ScraeDS
        Me.RebuildScrapeDS()        ' Link mapping requires table be build first

        Me._Crawler = New Crawler(Me)
    End Sub
```

ScraperDataAdapter

Scraper data adapter is an abstract class that contains the information to be saved to file or to database. The class contains the crawl project that will be saved to database or file, along with functions to access data (from database or from file system).

```
Public MustInherit Class ScraperDataAdapter
    Protected _Dataset As ScraperDB

    MustOverride Sub LoadProject()' Throws exception
    MustOverride S ub SaveProject()' Throws Exception

    MustOverride Sub SaveData()' Create new if it doesnn't exist

    MustOverride Sub LoadUrl()    ' Load all the url
    MustOverride Sub SaveUrl()    ' Save all the url

    Protected _CrawlProject As CrawlProject
    Public Property CrawlProject() As CrawlProject
        Get
            Return _CrawlProject
        End Get
        Set(ByVal value As CrawlProject)
            _CrawlProject = value
            If value IsNot Nothing Then
```

```
            Me._Dataset = value.Dataset
         End If
      End Set
   End Property

   Public MustOverride ReadOnly Property ProjectLocation()
```

FileScraperDataAdapter

File Scraper Data Adapter class inherits the Scraper Data Adapter Abstract class and handles the saving of dataset to file. The code use .Net Dataset's export xml (WriteXm), and import (ReadXml) to read and write the Dataset to disk.

```
Public Class FileScraperDataAdapter
   Inherits ScraperDataAdapter
   Public ProjectFileName As String
   Public DataFileName As String

   Public Overrides Sub LoadProject()
      Me._Dataset = New ScraperDB

      If Not System.IO.File.Exists(ProjectFileName) Then
         Throw New Exception("File Doesn't Exist")
      End If

      Try
         _Dataset.EnforceConstraints = False
         _Dataset.ReadXml(ProjectFileName)
```

SQLScraperDataAdapter

Unlike the File Scraper Adapter, the SQL Adapter requires a SQL connection command to connect the database. Using this connection, the SQL Adapter will generate necessary *select*, *insert*, and *update* queries.

The code below will load the crawl project from SQL database. The code will load each table in the dataset from database server (the tables listed in the Database scheme). The trick here is the generation of the SQL based on the table, getLoadProjectDataSQL() function.

```
   Public Overrides Sub LoadProject()
      Try
         Dim sql As String = ""
         Me._Dataset = New ScraperDB
         _Dataset.EnforceConstraints = False

         ' Save project data only
         For Each tbl As DataTable In Me._Dataset.Tables
            If tbl Is Me._Dataset.Project_Url Or tbl Is Me._Dataset.Url Or tbl Is
Me._Dataset.UrlReferences Then
               ' Do nothing, lets load it when it's trying to load the url
            Else
               sql = DataManagement.SqlExpress.getLoadProjectDataSQL(tbl, Me._CrawlProjectID)
               Dim da As New SqlClient.SqlDataAdapter(sql, Me._SqlConnection)

               da.Fill(tbl)
            End If
         Next

         Me._Dataset.AcceptChanges()

         ' Crawl Project
         Me._CrawlProject = New CrawlProject(Me._Dataset.CrawlProject(0))        ' Project loaded
         Me._CrawlProjectID = Me._CrawlProject.CrawlProjectRow.CrawlProjectID
         _Dataset.EnforceConstraints = True                    ' No errors

         Me.LoadUrl()                                          ' Load Urls
         Me._CrawlProject.ScraperDataAdaptor = Me              ' Set project's adaptor to this.
      Catch ex As Exception
         Console.WriteLine(ex.Message)
```

```
            Console.WriteLine(ex.StackTrace)
            Throw New Exception("Unhandled error in loading project from database: " & ex.Message)
        End Try
    End Sub

    Public Shared Function getLoadProjectDataSQL(ByVal dt As DataTable, ByVal crawlProjID As
Integer) As String
        Dim ds As ScraperDB = dt.DataSet
        Dim where As New System.Text.StringBuilder()
        Dim tables As New System.Text.StringBuilder()
        Dim sql As New System.Text.StringBuilder()

        Dim parentDT As DataTable = dt
        tables.Append("[" & parentDT.TableName & "]")

        If parentDT Is ds.CrawlProject Then
            where.Append("([CrawlProject].CrawlProjectID = ")
            where.AppendLine(crawlProjID & ")")
        End If

        While parentDT.ParentRelations.Count > 0
            Dim rel As DataRelation = Nothing

            For Each fkRel As Data.DataRelation In parentDT.ParentRelations
                If (fkRel.ParentTable Is ds.Project) Then
                    rel = fkRel
                    Exit For
                ElseIf (parentDT Is ds.Project_Url And fkRel.ParentTable Is ds.Url) Then
                    rel = fkRel
                    Exit For
                ElseIf (fkRel.ParentTable Is ds.CrawlProject) Then
                    rel = fkRel
                    Exit For
                End If
            Next

            If rel Is Nothing Then
                Exit While
            End If

            Dim parentCol As DataColumn = rel.ParentColumns(0)
            Dim childCol As DataColumn = rel.ChildColumns(0)

            where.Append(ControlChars.Tab)
            where.Append(String.Format("([{0}].{1} = ", rel.ParentTable.TableName,
parentCol.ColumnName))
            where.AppendLine(String.Format("[{0}].{1})", rel.ChildTable.TableName,
childCol.ColumnName))

            parentDT = rel.ParentTable
            If parentDT.ParentRelations.Count > 0 Then
                where.Append(" AND ")
            Else
                If parentDT Is ds.CrawlProject Then
                    where.Append(" AND ")
                    where.Append("([CrawlProject].CrawlProjectID = ")
                    where.AppendLine(crawlProjID & ")")
                End If
            End If

            tables.Append(",[" & parentDT.TableName & "]")
        End While

        If where.ToString <> "" Then
            sql.AppendLine(String.Format("SELECT [{0}].* from {1}", dt.TableName, tables.ToString))
            sql.AppendLine(String.Format("{0}WHERE {1};", ControlChars.Tab, where.ToString))
        Else
            sql.AppendLine(String.Format("SELECT [{0}].* from {1};", dt.TableName, tables.ToString))
        End If

        _Log.Debug(sql.ToString)
        Return sql.ToString
    End Function
```

Downloader/Network Manager

The downloader is implemented using Microsoft's .Net Asynchronous Programming Model. (4) The Asynchronous models allow a thread to effectively send event signals/message to another thread without causing race condition with the UI.

The downloader also uses the most commonly used data compression utilities: *dflate*, and *gzip*. The advantage of implementing support for this compression is that most web sites transfer data using such compression for fast transfer of text files by compressing them. Websites typically save 60-80% bandwidth off the text transfer (html, css, php, etc) using this approach. Since the crawler will be dealing with text files, this would allow the crawler to save 60-80% on bandwidth, and increase the performance of data download by that much. Implementing *dflate* and *gzip* turns out to be really easy. Just detect the type of the compression from the html response headers, and use the appropriate compression stream.

```
            Dim StreamResponse As Stream = Response.GetResponseStream()

            If _Response.ContentEncoding = "gzip" Then
               StreamResponse = New  Compression.GZipStream(StreamResponse,
CompressionMode.Decompress)
               Console.WriteLine("Downloading gzip")
            ElseIf _Response.ContentEncoding = "deflate" Then
               StreamResponse = New Compression.DeflateStream(StreamResponse,
CompressionMode.Decompress)
               Console.WriteLine("Downloading deflate")
            End If
```

Url Manager

Url Manager is to check the validity of the URL added to the solution as well and linking the Url Added to a [Scraper] Project that will handle the scraping of the content downloaded. Here a sample code of how the URL validates the URL. The code snippet shows the uses of .Net's Uri class to validate the URL string.

```
      If (Not Uri.IsWellFormedUriString(url, UriKind.Absolute) AndAlso linkMapList.Count = 0) Then
         _log.Info(url & " is not a well formed url.")
         Return Nothing
      End If
```

The next URL to crawl depends on the algorithm user prefer, depth-first crawling or breath-first crawling. The code determines the desired algorithm and will return the next URL to crawl. ToScrapeUrlView contains a list of URLs sorted by the date added.

```
   Public ReadOnly Property NextURL() As ScraperDB.UrlRow
      Get
         Dim r As ScraperDB.UrlRow = Nothing

         _log.Debug("Total to scrape: " & _ToScrapeUrlView.Count)
         _log.Debug("Total Url: " & Me._UrlDT.Count)

         If _ToScrapeUrlView.Count > 0 Then
            If IsDeapthFirst Then
               r = _ToScrapeUrlView.Item(_ToScrapeUrlView.Count - 1).Row
            Else
               r = _ToScrapeUrlView.Item(0).Row
            End If
            r.UrlStatusID = UrlManager.UrlStatus.Assigned
         End If

         Return r
      End Get
   End Property
```

Scrape Algorithm

The goal of the Scraper is to scrape the tag library tree given.

1.) Add the content of the URL as a data row entry to data table of root tag node, and start the scrape. Run the Scrape algorithm in a separate thread so that we can set limits on how much time should be allowed to scrape a page. If it exceeds the time given, the thread will be aborted and "Time Out" exception will be thrown.

```
   Public Sub Scrape(ByVal context As String, ByVal url As String, ByVal TagTree As TagTree)
      Me._WebUrl = url
      Me._Text = context
      Me._TagTree = TagTree

      Me._TagTree.ClearData()

      Dim rootNode As TagNode = Me._TagTree.RootNode

      If Me._Text = "" Or Me._Text Is Nothing Then
         Exit Sub
      End If

      ' PID does not matter for root node anyway, could be anything.
      rootNode.NewFieldData(Me._Text, 0, Me._Text.Length - 1)

      Dim starttime As Integer = System.Environment.TickCount
      autoEvent.Reset()
      Dim t As New System.Threading.Thread(AddressOf DoScrape)
      t.Start()

      If Not (autoEvent.WaitOne(Me.ScrapeTimeout, True)) Then
         t.Abort()
         Dim err As String = String.Format("Scrape Timedout at {0} seconds.", (Me.ScrapeTimeout /
1000).ToString("0"))
         Throw New Exception(err)
      End If

      ' Time how long it takes to scrape
      Me._ScrapeTime = New TimeSpan((System.Environment.TickCount - starttime) * 10000)
      _UrlRow = Nothing
   End Sub
```

2.) Combine the pattern of the children nodes into one big regular expression pattern, and apply the pattern to the data of the current tag node. (In the case of root, it will be the content of the URL). The code that does this is located in the Tag Node class because the class contains information about a Tag Node and its children.

```
   Public ReadOnly Property ScraperRegexCapture() As String
      Get
         If Not Me.IsParent Then
            Return _ScraperRegex
         End If

         If _ScraperRegex IsNot Nothing Then
            'Console.WriteLine(_ScraperRegex)
            Return _ScraperRegex
         End If

         ' FOR EACH DATAROW IN DATAVIEW
         Dim regexString As String = ""

         For Each aTagNode As TagNode In Me.TagNodeList
            Dim tempRegex As String = "", minMax As String

            'Set Greedy, lazy or given length
            If aTagNode.Row.EndTag = "" Then
               minMax = "*"               ' Problem with greedy, if i have nextNode, we don't want
greedy, we want lazy
            ElseIf (aTagNode.Row.MaxChars = 0) Then
               minMax = "*?"
            Else
               minMax = "{0," & aTagNode.Row.MaxChars & "}?"
```

```
            End If

            ' IF REGEX, then
            Dim tag1 As String = aTagNode.Row.StartTag
            If (aTagNode.Row.IsStartTagRegex = False AndAlso Not aTagNode.Row.IsSingleRegex) Then
               tag1 = Regex.Escape(tag1)
            End If

            ' IF REGEX, then
            Dim tag2 As String = aTagNode.Row.EndTag, body As String
            If (aTagNode.Row.IsEndTagRegex = False) Then
               tag2 = Regex.Escape(tag2)
            End If

            Dim name As String = "<" & aTagNode.ScrapeTagID & ">"

            ' SINGLE REGEX MODE
            If (aTagNode.Row.IsSingleRegex) Then
               tempRegex = String.Format("(?{0}{1})", name, tag1)
            Else
               tag1 = String.Format("(?{0}{1})", "<" & aTagNode.ScrapeTagID & "s>", tag1)
               tag2 = String.Format("(?{0}{1})", "<" & aTagNode.ScrapeTagID & "e>", tag2)
               body = String.Format("(?{0}{1}{2})", name, "[\w\W\s]", minMax)
               tempRegex = String.Format("{0}{1}{2}", tag1, body, tag2)
            End If

            If aTagNode.Row.EndTag = "" AndAlso aTagNode IsNot aTagNode.ParentTagNode.LastNode AndAlso
Not aTagNode.Row.IsSingleRegex Then
               body = String.Format("(?{0}{1})", name, "[\w\W\s]*?")
               regexString += String.Format("{0}{1}", tag1, body)
            ElseIf (aTagNode.Row.IsOptional) Then          ' Add more lazy operator IF IT OPTIONAL!
               regexString += String.Format("([\w\W\s]*?{0}|[\w\W\s]*?)[\w\W\s]*?", tempRegex)
               'regexString += String.Format("(?={0})?", tempRegex)
            ElseIf (aTagNode Is aTagNode.ParentTagNode.LastNode Or aTagNode.Row.IsSingleRegex) Then
               regexString += tempRegex
            Else
               regexString += tempRegex & "[\w\W\s]*?"
            End If
         Next

         Console.WriteLine("Regex: " & regexString)
         _ScraperRegex = regexString
         Return _ScraperRegex
      End Get
   End Property
```

3.) Split the regular expression match results, and insert the data into corresponding data table of the child node. The function below takes a match (one match out of many matches), and split the data (using regular expression's match groups), and insert the data matched into appropriate tag node's data table.

```
Private Sub SaveScrapedData(ByVal aScraperNode As TagNode, ByVal match As Match, ByVal PRow As
ScraperTempDS.FieldDTRow)
      Dim row As DataRow = Nothing

      ' A new data row.
      If aScraperNode.DataDT IsNot Nothing Then
         row = aScraperNode.DataDT.NewRow
         aScraperNode.DataDT.Rows.Add(row)         ' This will generate rowID
      End If

      For Each node As TagNode In aScraperNode.TagNodeList
         Dim body As Group = match.Groups(node.ScrapeTagID)                    ' Data Group
         Dim startTag As Group = match.Groups(node.ScrapeTagID & "s")          ' Start Group
         Dim endTag As Group = match.Groups(node.ScrapeTagID & "e")            ' End Group

         Dim data As String
         Dim startIndex, dataIndex, endIndex, endLength As Integer

         ' For Text Coloring - Store the location of the place found Matched
         startIndex = PRow.DataTagIndex + startTag.Index
         If startIndex = 0 Then
            startIndex = match.Index
         End If

         ' ReverseSearch
```

```
         If (node.Row.isReverseSearch) Then
            data = startTag.Value & body.Value & endTag.Value
            Dim m As Match = Regex.Match(data, node.ScraperReverseRegexCapture,
RegexOptions.RightToLeft)

            If m.Success Then
               body = m.Groups(node.ScrapeTagID)
               startTag = m.Groups(node.ScrapeTagID & "s")
               endTag = m.Groups(node.ScrapeTagID & "e")
               startIndex = startTag.Index + startIndex
            End If
         End If

         dataIndex = startIndex + startTag.Length
         endIndex = dataIndex + body.Length
         endLength = endTag.Length

         data = body.Value
         If node.Row.IsAppendStartTag Then
            data = startTag.Value & data
            dataIndex = startIndex
         End If

         If node.Row.IsAppendEndTag Then
            data = data & endTag.Value
         End If

         If (node.Row.IsURL) Then
            data = getAbsoluteURL(data, Me._WebUrl)
         End If

         ' Dynamic code edit: TBD

         ' Save other Data into the table
         'Dim pid As Integer = PRow.PID
         Dim fieldRow As ScraperTempDS.FieldDTRow

         If aScraperNode.Row.IsDataTable AndAlso row IsNot Nothing Then
            fieldRow = node.NewFieldData(data, dataIndex, endIndex)                ' Set parent ID
            If row(DataTableUtil.ID) IsNot Nothing Then fieldRow.PID = row(DataTableUtil.ID)

            If aScraperNode.ParentDataTableTagNode IsNot Nothing Then
               Dim colName As String = aScraperNode.ParentDataTableTagNode.DataDT.TableName &
DataTableUtil.POSTFIX
               row(colName) = PRow.PID  ' May need to try catch error ->

               fieldRow.PID = row(colName)
            End If
         Else
            fieldRow = node.NewFieldNoLinkData(data, dataIndex, endIndex)          ' Do not use Auto
Generated ID
         End If

         fieldRow.StartTagIndex = startIndex
         fieldRow.EndTagLength = endLength

         ' If this node is saving data, then row must not be nothing!
         If node.Row.isSaveData Then
            ' Fix the data using a datarefiner
            If Me._CrawlProject IsNot Nothing And node.Row.DataTypeRow IsNot Nothing Then
               Dim dRefiner As DataRefinerAsm.DataRefiner =
Me._CrawlProject.DataTypeManager.getDataType(node.Row.DataTypeRow).DataObject
               row(node.Row.TagName) = dRefiner.Refine(data)
            Else
               row(node.Row.TagName) = data
            End If

            ' Generate checksum
            CrawlHelper.GenerateChecksum(row)
         End If
      Next

      If row IsNot Nothing Then
         row(DataTableUtil.URL) = Me._WebUrl

         ' Add Url RowID
         If Me._UrlRow IsNot Nothing Then
            row(DataTableUtil.URLIDFK) = Me._UrlRow.UrlID
         End If
      End If
   End Sub
```

4.) If the tag node is of Url Node type then convert the data extracted into absolute URL links. The function below takes the relative URL, and the URL used to download the page. The function will return valid absolute URL. The EscapeUriString function of the Uri class will escape invalid URL characters, such as / to %05, etc.

```
   Public Shared Function getAbsoluteURL(ByVal relUrl As String, ByVal mainUrl As String) As String
      Try

         Dim baseUri As Uri = New Uri(mainUrl)
         Dim absoluteUri As Uri = New Uri(baseUri, Uri.EscapeUriString(relUrl))
         absoluteUri = New Uri(absoluteUri, Uri.EscapeDataString(absoluteUri.AbsolutePath))

         Return absoluteUri.ToString()
      Catch ex As Exception
         Return relUrl
      End Try
   End Function
```

5.) Repeat Step 2-4 to the child nodes that is of parent node type. (The child node has children)

Crawl Algorithm

The goal of the Crawler is scrape one page, and use the data extracted by the scraper appropriately. The code for crawling is contained the Crawler.vb file.

1.) Gets the next Url from the Url Manager (Url Manager will return the last added url that has not been crawled – if Depth first, or the first added url that has not been crawled – if Breath first)
2.) Download the content of the url using Network Manager
3.) Scrape the content downloaded using the Scraper
4.) Based on the links from Link Manager, add the extracted links to the Url Manager with their designated [scraper] project to Url Manager.
5.) Build relationship on the extracted data based on the Link Manager, and the Project_Dataset data table. (This process requires that another temporary data base to keep track of primary keys).

```
Private Sub UpdateReferences(ByVal linkMap As LinkMap, ByVal urlRow As ScraperDB.UrlRow)
      Dim proj As Project = linkMap.Project

      Dim datasetDT As DataTable =
Me._CrawlProject.Dataset.Tables(DataTableUtil.URL_DATASET_TABLENAME(Me._CrawlProject.CrawlProjectRow.C
rawlProjectID))
      Dim dv As New DataView(datasetDT)
      dv.RowFilter = Me._CrawlProject.Dataset.Url.TableName & DataTableUtil.POSTFIX & " = " &
urlRow.UrlID

      Dim dsRow As DataRow
      If dv.Count > 0 Then
         dsRow = dv(0).Row
      Else
         Return      ' No relations exists
      End If

      For Each node As TagNode In proj.TagTree.DataTagNodes
         If (node.ParentDataTableTagNode Is Nothing) Then         ' If the table in the project is a
root table then

            If node.DataDT.Rows.Count = 0 Then
               _log.Warn("No data scraped in the project")
            Else
               ' Get Url ID and find the column
               If linkMap.IsPartialData Then
                  copyFieldRelation(node, dsRow)
               Else
                  _log.Info("Copying data references.")
                  CopyTableRelations(node, dsRow)
               End If
```

```
            End If
        End If
    Next
End Sub
```

6.) If Database access: if the set amount of urls is crawled, then save the project and project data to database. Then reload the Url List.

7.) If File system: Repeat from step 1 until there is no urls to get from Url Manager

```
    Private Sub NewStartAll()
        If (Me._CrawlProject.UrlManager.TotalScraped + Me._CrawlProject.UrlManager.TotalAssigned
>= Me.MaxUrl Or _
        Me.mMyWebclient.CurrentTotalThreads = 0) And _
        TypeOf (Me._CrawlProject.ScraperDataAdaptor) Is SQLDatabaseScraperAdapter Then

        ' If all the assigned url is finished, and project is saveable
        If Me._CrawlProject.UrlManager.TotalAssigned = 0 And Me._CrawlProject.UrlManager.IsSaveable
Then
            ' If sql type of crawling, then we want to save data when there's no URL left
            saveData(True)
        Else
            ' If all the partials are assigned, dun't assign any more.
            Dim urlDT As ScraperDB.UrlDataTable = Me._CrawlProject.Dataset.Url
            Dim partialDV As New DataView(urlDT)

            ' Make sure there are no partial url in progress
            partialDV.RowFilter = urlDT.IsPartialDataColumn.ColumnName & " = True AND " & _
              urlDT.UrlStatusIDColumn.ColumnName & " = " & UrlManager.UrlStatus.Ready

            If partialDV.Count = 0 Then
                Exit Sub
            End If
        End If
    End If

    Me._CrawlStatus = CrawlStatus.Crawling
    For i As Integer = Me.mMyWebclient.CurrentTotalThreads To
Me._CrawlProject.CrawlProjectRow.Threads - 1
        ' Get the next Url to crawl
        Dim urlRow As ScraperDB.UrlRow = Me._CrawlProject.UrlManager.NextURL

        If urlRow IsNot Nothing Then  ' Start a new download
            _Log.Debug("Crawling: " & urlRow.UrlID)
            _Log.Debug("Crawling: " & urlRow.UrlLink)

            ' Skip, make room for partial urls to download + scrape
            If Me._CrawlProject.UrlManager.PartialUrlCount > Me.MaxUrl And urlRow.IsPartialData =
False Then
                urlRow.UrlStatusID = UrlManager.UrlStatus.Ready
                Continue For
            End If

            Try
                Dim uri As New Uri(urlRow.UrlLink)
                mMyWebclient.DownloadStringAsync(New Uri(urlRow.UrlLink), urlRow.UrlReferer, Nothing,
New CrawlInfo(urlRow))
            Catch ex As UriFormatException
                urlRow.UrlStatusID = UrlManager.UrlStatus.Error
                urlRow.ErrorMessage = ex.Message
            Catch ex As Exception
                urlRow.UrlStatusID = UrlManager.UrlStatus.Error
                urlRow.ErrorMessage = ex.Message
            End Try
        Else
            Me._CrawlStatus = CrawlStatus.Ready
            RaiseEvent CrawlFinished()
        End If
    Next
End Sub
```

SQLDDL

I got the idea of generating SQL DDL code given a dataset from reading ADO.Net Cook book by O'Reilly. The function below will generate the SQL code to create the data base table given a .Net Dataset's DataTable. (5)

```
    Public Shared Function getCreateTableSQL(ByVal dt As DataTable) As String
       Dim tablename As String = dt.TableName
       Dim sqlStr As New System.Text.StringBuilder
       sqlStr.AppendLine(String.Format("Create Table {0}", tablename))
       sqlStr.AppendLine("(")

       Dim pkCol As DataColumn = dt.PrimaryKey(0)
       For i As Integer = 0 To dt.Columns.Count - 1
          Dim col As DataColumn = dt.Columns(i)

          Dim isNullString As String = "not null"
          If col.AllowDBNull Then
             isNullString = "null"
          End If

          If col.DataType Is GetType(Integer) Then
             If col Is pkCol Then
                sqlStr.Append(String.Format("[{0}] {1} IDENTITY(0,1) PRIMARY KEY", col.ColumnName,
"int"))
             Else
                sqlStr.Append(String.Format("[{0}] {1} {2}", col.ColumnName, "int", isNullString))
             End If
          End If

          If col.DataType Is GetType(String) Then
             If col.MaxLength = -1 Then
                sqlStr.Append(String.Format("[{0}] {1} {2}", col.ColumnName, "text", isNullString))
             Else
                sqlStr.Append(String.Format("[{0}] varchar({1}) {2}", col.ColumnName, col.MaxLength,
isNullString))
             End If
          End If

          If col.DataType Is GetType(Date) Then
             ' timestamp stores DateAandTime
             sqlStr.Append(String.Format("    [{0}]          {1}              {2}", col.ColumnName,
"datetime", isNullString))
          End If

          If col.DataType Is GetType(Boolean) Then
             sqlStr.Append(String.Format("    [{0}]          {1}              {2}", col.ColumnName,
"bit", isNullString))
          End If

          If i = dt.Columns.Count - 1 Then
             sqlStr.AppendLine()
          Else
             sqlStr.AppendLine(",")
          End If
       Next
       sqlStr.AppendLine(");")

       _Log.Info("SQL Generated: " & sqlStr.ToString)
       Return sqlStr.ToString
```

But a dataset in this program contains more than just one data table, and there are many relationships and constraints for each data table. The program must also be able to generate SQL command for building relationship and constraints as well. The code below generates SQL command for building constraints, and relationships. I got most of the idea and SQL syntax from a book I used in CSC 365 (Oracle 9i). It turns out MS SQL Server and Oracle shares some common syntax when it comes to creating constraints. (6)

```
       Public Shared Function getIndexDropSql(ByVal dt As DataTable)
          Dim sql As New System.Text.StringBuilder()

          For Each rel As DataRelation In dt.ParentRelations
             Dim parentDT As DataTable = rel.ParentTable
             Dim col As DataColumn = rel.ChildColumns(0)
             Dim indexName As String = String.Format("Index_{0}_{1}", parentDT.TableName,
col.ColumnName)

             sql.AppendLine(String.Format("if exists (select 1 from  sysindexes where id =
object_id('{0}') and name = '{1}')", dt.TableName, indexName))
             sql.AppendLine(String.Format("   drop index {0}.{1}", dt.TableName, indexName))
             sql.AppendLine(String.Format(";"))
          Next
```

```
            Return sql.ToString
        End Function

        Private Shared Function getDSRelationFK(ByVal ds As DataSet)
            Dim sql As New System.Text.StringBuilder()

            For Each rel As DataRelation In ds.Relations
                Dim childCol As DataColumn = rel.ChildColumns(0)
                Dim parentCol As DataColumn = rel.ParentColumns(0)

                sql.AppendLine(String.Format("alter table {0}", rel.ChildTable))
                sql.AppendLine(String.Format("    add constraint {0} foreign key ({1})", rel.RelationName,
childCol.ColumnName))
                sql.AppendLine(String.Format("       references {0} ({1})", rel.ParentTable.TableName,
parentCol.ColumnName))
                sql.AppendLine(String.Format("       on delete cascade"))
                sql.AppendLine(String.Format(";"))
            Next

            Return sql.ToString
        End Function

        ''' <summary>
        ''' Get fk constraints
        ''' </summary>
        ''' <param name="dt"></param>
        ''' <returns></returns>
        ''' <remarks></remarks>
        Public Shared Function getTableFK(ByVal dt As DataTable)
            Dim sql As New System.Text.StringBuilder()

            For Each rel As DataRelation In dt.ParentRelations
                Dim childCol As DataColumn = rel.ChildColumns(0)
                Dim parentCol As DataColumn = rel.ParentColumns(0)

                sql.AppendLine(String.Format("alter table {0}", dt.TableName))
                sql.AppendLine(String.Format("    add constraint {0} foreign key ({1})", rel.RelationName,
childCol.ColumnName))
                sql.AppendLine(String.Format("       references {0} ({1})", rel.ParentTable.TableName,
parentCol.ColumnName))

                If rel Is dt.ParentRelations(0) And rel.ParentTable IsNot rel.ChildTable Then
                    sql.AppendLine(String.Format("       on delete cascade"))
                End If

                sql.AppendLine(String.Format(";"))
            Next

            Return sql.ToString
        End Function
```

If the user wanted to remove the data tables from the database, the tables must be dropped in the correct order. This code below determines which data table should be dropped first, and generate the data table drop SQL commands in the correct order.

```
        Public Shared Function getDropCreateLSQL(ByVal crawlProject As CrawlProject) As String
            Dim ds As ScraperDB = crawlProject.Dataset
            Dim sqlDrop As New System.Text.StringBuilder
            Dim sqlCreate As New System.Text.StringBuilder
            Dim dropString As String = ""

            Dim sqlDropIndex As New System.Text.StringBuilder
            Dim sqlCreateIndex As New System.Text.StringBuilder

            Dim sqlCreateConstaint As New System.Text.StringBuilder

            ' we problaby need want to drop the tables in order

            For Each dt As DataTable In ds.Tables
                Console.WriteLine(dt.TableName & ": " & dt.ParentRelations.Count)
            Next

            Dim tblList As List(Of DataTable) = crawlProject.DataMapper.getDataTablesUpdateOrder(False)
            For Each tbl As DataTable In tblList
                If DataTableUtil.IsGeneratedTable(tbl) Then
                    dropString = SQLDDL.getDropTableSql(tbl) & dropString    ' Drop in the right order.
                    sqlCreate.AppendLine(SQLDDL.getCreateTableSQL(tbl))
```

```
                sqlDropIndex.AppendLine(SqlExpress.getIndexDropSql(tbl))
                sqlCreateIndex.AppendLine(SqlExpress.getIndexTableSQL(tbl))

                sqlCreateConstaint.AppendLine(SqlExpress.getTableFK(tbl))
            End If
        Next

        sqlDrop.AppendLine(dropString)
        sqlDrop.AppendLine(sqlDropIndex.ToString)
        sqlDrop.AppendLine(sqlCreate.ToString)
        sqlDrop.AppendLine(sqlCreateIndex.ToString)

        'sqlDrop.AppendLine(SqlExpress.getDSRelationFK(ds))
        sqlDrop.AppendLine(sqlCreateConstaint.ToString)

        _Log.Debug(sqlDrop.ToString)
        Return sqlDrop.ToString
    End Function
```

Dynamic Data Type

Dynamic Data type allows the user to write a program in VB.net and let the crawler load the program that the user had written. I accomplished this by using Reflection in .Net. Here's a code of how the dynamic code is loaded to memory. The function takes the name of the object, and the code for the object. This dynamic code is limited to a class that implements the abstract Data Refiner class, which is contains one function, Refine().

```
   Private Function compileCode(ByVal className As String, ByVal strCode As String) As DataRefiner
      Dim objCodeCompiler As ICodeCompiler = New VBCodeProvider().CreateCompiler
      Dim objCompilerParameters As New CompilerParameters()

      Dim asmUri As New Uri(System.Reflection.Assembly.GetAssembly(GetType(DataRefiner)).CodeBase())
      Dim asmPath As String = Uri.EscapeUriString(asmUri.AbsolutePath)
      asmPath = asmPath.Replace("/", "\\")
      asmPath = Uri.UnescapeDataString(asmUri.AbsolutePath)
      Console.WriteLine(asmPath)

      objCompilerParameters.ReferencedAssemblies.Add("System.dll")
      objCompilerParameters.ReferencedAssemblies.Add("System.Windows.Forms.dll")
      objCompilerParameters.ReferencedAssemblies.Add(asmPath)

      objCompilerParameters.GenerateInMemory = True

      Dim objCompileResults As CompilerResults =
objCodeCompiler.CompileAssemblyFromSource(objCompilerParameters, strCode)
      If objCompileResults.Errors.HasErrors Then
         Throw New Exception("Error: Line>" & objCompileResults.Errors(0).Line.ToString & ", " &
objCompileResults.Errors(0).ErrorText)
         Return Nothing
      End If

      Dim objAssembly As System.Reflection.Assembly = objCompileResults.CompiledAssembly
      Dim objTheClass As Object = objAssembly.CreateInstance(className)

      Console.WriteLine("Created Data type: " &
System.Reflection.Assembly.GetAssembly(objTheClass.GetType).CodeBase())
      Return objTheClass
   End Function
```

Here is the code for the abstract Refine class:

```
Public MustInherit Class DataRefiner
   Public MustOverride Function Refine(ByVal Text As String) As Object
End Class
```

This is the sample class that inherits Data Refiner and is loaded dynamically as the crawl project is loaded. This class takes a string and returns the first text that matches the pattern for money specified as the match: "\$.*?(\d+\.\d+)".

```
Imports System
Imports Microsoft.VisualBasic
```

```
Imports System.Text.RegularExpressions
Imports DataRefinerAsm

Public Class MoneyDataRefiner
   Inherits DataRefiner

   Public Overrides Function Refine(ByVal Text As String) As Object
      Dim m As Match = Regex.Match(Text, "\$.*?(\d+\.\d+)")

      If m.Success Then
         m = Regex.Match(m.Value, "\d+\.\d+")

         Return Double.Parse(m.Value)
      End If

      Return Nothing
   End Function
End Class
```

Testing

Unit testing is done to test individual component of the project. Visual studio offers a unit testing project that can unit test another project's projects and files.

The testing for the crawler is done by crawling a few sample website with different crawl solution configurations. Running the crawler for about 5 minutes and checking the data extracted. If the data crawled matches the desired data, then the crawler works. Refer to the crawler manual's sample crawl project for testing of the crawler and Scraper Manual's sample projects for testing the functionality of the scraper.

The non functionality requirement is tested by letting the crawler run for a few hours while monitoring the CPU, memory size and page file size (Window's Task Manger).

5. Conclusion

Summary

All of the requirements previously listed have been implemented. The main problem that I ran into was implementing new features that weren't in the original specifications, which got me off-track from the main development. I solved the problem by creating new projects for each of the new features so that they don't distract the main development. As the result, there is a lot of extra code that is at the development or integration stage, such as Cron Manger, Tag Library Wizard, and LCS Delta.

The documentation is far from completion, not all the classes and algorithms are explained. But I think this document contains more than enough details about the project for anther software developer to understand the design and implementation of the project.

Future Plans

There is still much work that is required for the crawler to become an enterprise-level, scalable and durable application. I identify below some features that I think would help the crawler become more valuable.

Crawling Service

The crawler UI will not be implemented to support multiple crawling operations. Instead, a new program will be developed. It will act as a Windows service that runs with Windows.

Web Service

A web application is currently being developed which would allow users to upload a crawler project to a server where it will perform the task of crawling and managing data on the behalf of the user. This way, the user may take advantage of the crawler and would not need to set up a database server.

Artificial Intelligence

I wrote a few codes which acts as a "Scraper Wizard," where a user can enter in a sample data, and sample website, and the scraper will try to generate a tag library that could be used to scrape other similar pages. Using this process, the crawler will be able to generate tag libraries based on the sample data, and atomically crawl more websites.

Ajax/JavaScript Support

The crawler needs to be able to download and extract information from web applications such as Yahoo Mail, Goggle Maps, and Gap.com which heavily rely on JavaScript. In order to do that, the crawler must be able to interpret JavaScript. The only JavaScript interpreter that I found was a side project by Mozilla, called Rihino. (7) But the project is created for the Java environment, so I can't use it within the .Net environment.

Cookie and Post Data support

Url Manager must support URLs with post data settings to be able to request information from some web sites which use forms for navigation (such as back and previous form buttons). The Downloader must be able to request information as *post* data rather than *get* data. The code for handling post data is implemented in the Network Manager as part of the core library, but is not yet integrated with the UI.

Cookies, on the other hand, are a completely different problem. Most websites save *cookie*(s) in the client's computer to keep track of state. The websites may require the browser to send *cookie* every time a page is requested. The crawler must be able to support functionality for receiving and summiting *cookies* to be able to crawl such websites.

Web Brower Simulation

Web browser simulation is probably the ultimate approach to crawl any website out there, even web application. The Web Brower simulation approach will be to implement a web browser and simulate a mouse click as a user clicking on the Web Browser and crawling the website.

6. References

1. SQL Server Express - Easy to Use. *http://msdn.microsoft.com/.* [Online] 31 November 2006. [Cited: November 31 2006.], http://msdn.microsoft.com/vstudio/express/sql/

2. **Volking, Frederick. .** RichTextBoxHS with background highlighting in VB.NET. *http://www.hunterstone.com/.* [Online] Hunter STone, 30 July 2002. [Cited: December 12 2006.], http://www.hunterstone.com/library/RichTextBoxHS.htm

3. **Microsoft, MSDN. .** Data Points: Efficient Coding With Strongly Typed DataSets -- MSDN Magazine, December 2004. *http:///www.msdn.com.* [Online] 1 December 2004. [Cited: December 10 2006.], http://msdn.microsoft.com/msdnmag/issues/04/12/DataPoints/default.aspx

4. **InfoSys. .** Infosys | Microsoft: Asynchronous Programming Model in .NET Framework 2.0 - Part I,,,:. *http://infosysblogs.com/.* [Online] 19 September 2006. [Cited: December 12 2006.], http://infosysblogs.com/microsoft/2006/09/asynchronous_programming_model.html

5. **Hamilton, Bill. .** *ADO.net Cook Book.* s.l. : O'Reilly, 2003. 0-596-00439-7.

6. **Sunderraman, Rajshekhar. .** *Oracle 9i Programming, A Primer.* s.l. : Pearson Eduction, Inc, 2004. 0-321-19498-5.

7. **Mozilla Rihno. .** Mozilla Rihno. *http://www.mozilla.org/.* [Online] 7 December 2006. [Cited: December 7 2006.], http://www.mozilla.org/rhino/

8. **VelocityScape. .** VelocityScape Home. *http://www.velocityscape.com/.* [Online] 31 November 2006. [Cited: November 31 2006.]

9. **DotNetNuke. .** DotNetNuke > Home (DNN 4.3.7):. *http://www.dotnetnuke.com/.* [Online] 12 December 2006. [Cited: December 12 2006.], http://www.dotnetnuke.com/

10. **Microsoft. .** Microsoft .NET Framework Version 2.0 Redistributable Package. *http://www.microsoft.com/.* [Online] 31 Novembder 2006. [Cited: Novembder 31 2006.], http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en#Requirements

7. Appendices

Alternative Solutions

Here's a list of other Alternative crawling solutions that that I found online. There are a few more, the keywords for such software are: scraper, crawler, spider, data miner, data aggregation.

Velocityscape

Velocity Scrape offers a web scraper that can extract information from a list of URL, and store them into text file, or data base. But the scraper doesn't provide site navigation feature and category referencing features. (8)