Automated and Scheduled Maintenance of Digital Library Collections

Wendy Osborn and Steve Fox
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta
T1K 3M4
Canada
Email: wendy.osborn@uleth.ca

Abstract

In this paper, we propose a strategy for the automated and scheduled maintenance of a digital library collection. Existing systems require the user either to add new data manually to a collection, or to have programming knowledge in order to use existing application programming interfaces (APIs) in order to automate scheduled collection updates. We incorporate a scheduling module into the Greenstone digital library software, which allows the user to set up scheduled and automated building of a collection at periodic intervals. This module interacts with the task scheduler on the host platform, such as Linux, Windows and Mac OS X, thereby making it a simple yet powerful tool for scheduled collection maintenance.

1. Introduction

Many applications generate multimedia documents on a daily basis. Different types of multimedia documents that are generated constantly include images, video clips and audio recordings. For example, the municipal police in many cities use a photo-radar program to catch vehicle operators whose vehicles are traveling over the speed limit. This application generates many pictures of vehicle license plates every day. If a collection of license-plate images is organized into a digital library, this collection needs to be updated regularly to incorporate the recently acquired images .

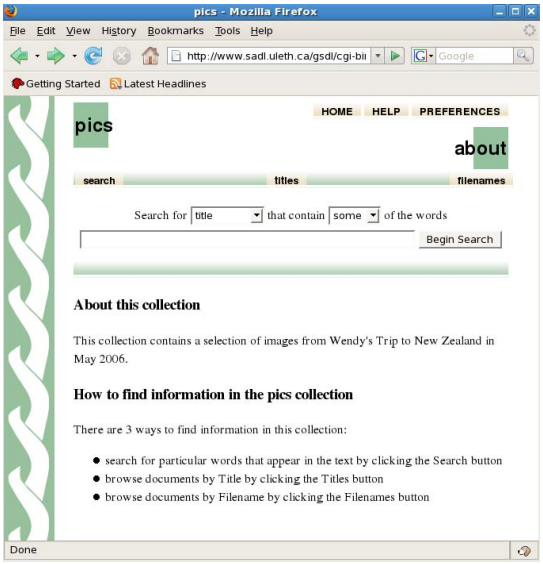
In addition, applications exist that modify documents on a weekly, monthly or yearly basis. For example, many post-secondary institutions maintain a website of information on topics such as program information, admission requirements and information on faculty members. These websites are updated periodically to keep this information current. If a collection of information across several post-secondary institutions is kept in one central repository such as a digital library, then the collection must be updated periodically to maintain consistency with the existing websites.

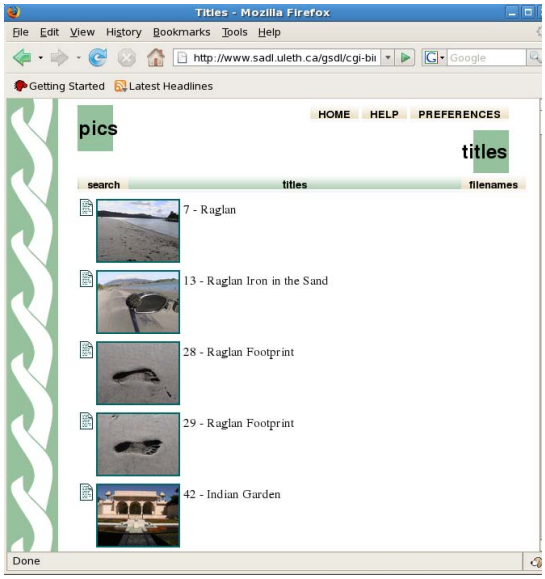
When managing a collection of documents using digital library software, new documents must be added manually to a collection. This is reasonable when documents are added infrequency. However, when documents are added to a collection on a regular basis, such as hourly or daily, an automated and scheduled approach is more desirable. Such an approach will not be time consuming for the user, and will make that time available for other important tasks.

Existing digital library software such as DSpace [5], Fedora [2] and Greenstone [7] require that the images be added manually to the collection. In the case of Fedora, data can be retrieved from a remote location at the time of viewing. However, this location needs to be manually configured. Further, although Fedora and Dspace do provide application programming interfaces (APIs) that could be used to create an automatic updater that schedules tasks, programming knowledge is required for using an API and setting up tools based on it. A tool that is already available for automatic and scheduled updating is more desirable.

We present a solution for automating and scheduling additions and updates to a collection that occur on a regular basis. We extend the Greenstone digital library software to include a scheduling module. This module both automates the construction of a collection, and schedules the construction to occur at specific intervals, such as every hour, day or week. The scheduler works with any existing collection. In addition, the owner of a collection can still perform manual updates when the collection is scheduled for automated building. If the owner needs to remove a document that was mistakenly added during scheduled maintenance, he/she can manually remove the document without affecting the existing scheduled building task. Further, by interacting with the existing task scheduling mechanism on

1-4244-1476-8/07/$25.00 ©2007 IEEE



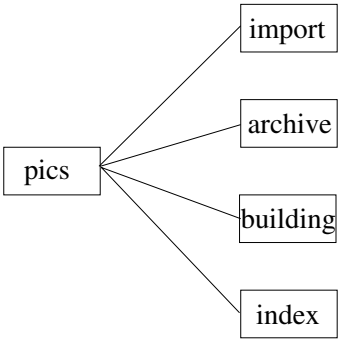


the host system, the scheduling module is kept minimal, but still provides a powerful tool for automatic collection maintenance. Experiments are performed to verify the correctness of the scheduling module in different situations.

The remainder of this paper proceeds as follows. In Section 2, we present some background information on the Greenstone digital library software. In Section 3 we present some background information on cron, which is the task scheduler available on Linux systems, and also available for Windows and Mac OS X. In Section 4 we present our proposed scheduler for automating collection building in Greenstone. In Section 5, we present the results of our performance evaluation on the scheduler. In Section 6, present an example application that the Greenstone scheduler will benefit. Finally, Section 7 concludes the paper and presents some research directions.

2. Greenstone

Greenstone [7] is a suite of software for creating digital library collections and making them available locally or via the Internet. A collection can contain documents of different formats, including images, postscript and PDF files, audio, and formatted and unformatted text. A collection built with Greenstone can be customized in many ways. For example, a collection owner can customize the types of documents that can appear in the collection, the appearance of the interface of the collection, and how the collection will be accessed by other users. In addition, Greenstone is extensible. For example, functionality to support other data formats that are not provided with Greenstone can easily be added to a collection.



There are two types of accessors in Greenstone. The first is an index that provides support for searching. The second is a classifier that support for browsing.

Figures 1 and 2 depict a Greenstone collection called *pics*. Figure 3 depicts the internal representation of the *pics* collection. Here, the four main directories of a collection – import, archive, building, and index – are displayed. The purpose of each is described below:

- *import*. The import directory contains all documents that are to be added to the collection.
- *archive*. The archive directory stores all documents that have been processed from the import folder and added to the collection.
- *building*. The building directory contains all indices and classifiers during the creation process.

- *index*. The index directory contains the indices and classifiers after they are created.

A Greenstone collection is created or modified by following four steps [7]: document addition, document importation, accessor creation, and collection activation. Each step is described in detail below:

1. *document addition*. New documents that will become part of a collection are copied into the import directory for the collection.

2. *document importation*. The documents in the import folder are now processed for the collection by specifying an import command. Different importing options can be specified by the user. For example, documents in the import folder can be added to an existing collection by using a -keepold option. Alternatively, all documents in the import folder - new and existing - can be used to create a new instance of the collection by using a -removeold option. All imported documents are located in the archive directory.

3. *accessor creation*. After the documents in the import folder are processed, indices and classifiers are set up by using a build command. New indices and classifiers can be built by specifying a -removeold option. Alternatively, existing indices and classifiers can be modified by specifying a -keepold or -incremental option. The resulting indices and classifiers are located in the build directory.

4. *collection activation*. Finally, the collection is activated by copying the indices and classifiers from the building directory to the index directory. The collection is now viewable online.

The command for each step can be executed in a terminal or command window. Alternatively, a graphical user interface called the Greenstone Librarian Interface (GLI) is available that incorporates the above steps in a user-friendly manner. In this case, the only steps that are required by the user are to add documents to the import folder via an import panel, and to provide customizations for each command.

3. Cron

Cron [3] is a program for users to schedule tasks that will run automatically at a specified time. A task can be one command, or a script containing several commands that are executed in sequence. Initially, cron was implemented for Unix and Linux platforms, with most systems running Vixie cron [6]. Mac OS X also runs Vixie cron. In addition, versions of cron now exist for Windows platforms, such as Pycron [1]. We summarize the general ideas behind all implementations of cron that are applicable to our work.

```
30  *  *  *  * /collect/pics/gsdl.pl
59 23  *  *  * /usr/bin/cleanup.bash
00  6  *  *  7 /home/someuser/alarm
00  0  1  1  * echo "Happy New Year!"
```

Cron runs continuously in the background of the operating system. Every minute, cron reads several task configuration files. Each such file is called a crontab file. A crontab file contains a record for every task that is scheduled for execution. Cron locates and runs all tasks that are scheduled at the current time.

The format of a crontab record is (min hr dom moy dow $user$ $task$), where min, hr, dom, moy, and dow are the minute, hour, day of month, month of year and day of week, respectively, $user$ is the username that the command will run under, and $task$ is the command or script that is executed at the specified time. A task can be scheduled to run hourly, daily, weekly, monthly, or yearly:

- *hourly*. An hourly scheduled task executes at a specified minute.

- *daily*. A task that is scheduled daily executes at a specific hour, which is required, and a specific minute. If no minute is specified, execution is assumed to take place at the beginning of the specified hour.

- *weekly*. A weekly scheduled task runs on a specified day, which is required, and at a specified hour and minute. If no hour or minute is specified, execution is assumed to take place at midnight.

Unspecified values are replaced with an asterisk.

Systems that use Vixie cron support two types of crontab files – system crontab and user crontab. The system crontab files are primarily for system administration and maintenance tasks. Also, even if all tasks have a specified low-level username, root privileges are required for modifying a system crontab file. User crontab files, on the other hand, can be set up by any user on the system to execute tasks under their own username. Assuming the user has permission to execute the task, no root permissions are required. In addition, Pycron only supports user crontab files. Therefore, the Greenstone scheduler uses a user crontab file.

Figure 4 displays a sample user crontab file. It contains 4 tasks that are scheduled for specific times. The first task, /collect/pics/gsdl.pl, is scheduled to be run at 30 minutes past every hour. The second task, /usr/bin/cleanup.bash, is scheduled for execution daily at 11:59pm. The third task, home/someuser/alarm, is scheduled every Sunday at 6:00am. Finally, the fourth task, which echoes "Happy New Year!", is schedule for execution every January 1st at 12:0am.

```
#!/usr/bin/perl

$ENV{'GSDLHOME'}="/home2/gsdl/gsdl";
$ENV{'GSDLOS'}="linux";
$ENV{'GSDLLANG'}="";
$ENV{'PATH'}="/bin:/usr/local/sbin:/usr/local/bin:/sbin:/usr/sbin:/usr/bin:/usr/X11R6/bin/
:.:/usr/local/gsdl/bin/script:/usr/local/gsdl/bin/linux";
system("import.pl -removeold pics");
system("buildcol.pl -removeold pics");
system("\\rm -r /gsdl/collect/pics/index/*");
system("mv /gsdl/collect/pics/building/*
             /gsdl/collect/pics/index/");
system("chmod -R 755 /gsdl/collect/pics/index/*");
```

```
#!/usr/bin/perl

$ENV{'GSDLHOME'}="C:\\gsdl";
$ENV{'GSDLOS'}="windows";
$ENV{'GSDLLANG'}="en";
$ENV{'PATH'}="C:\\gsdl\\bin\\windows\\perl\\bin;C:\\gsdl\\bin\\windows;C:\\gsdl\\bin
\\script;c:\\program files\\imagemagick-6.3.2-q16;C:\\usr\\bin\\;C:\\Perl\\bin\\;
C:\\WINDOWS\\system32;C:\\WINDOWS;
system("import.pl pics");
system("buildcol.pl pics");
system("rd \/S \/Q \"C:\\gsdl\\collect\\pics\\index\"");
system("md \"C:\\gsdl\\collect\\pics\\index\"");
system("xcopy \/E \/Y \"C:\\gsdl\\collect\\pics\\building\\*\" \"C:\\gsdl\\collect\\pics
\\index\\\"");
```

4. Task Scheduling in Greenstone

In this section, we present our design for a scheduler module for Greenstone. It is written in Perl and runs on Linux, Windows and Mac OS X. We chose perl because a perl script can be executed across different platforms without having to recompile it. In addition, we can maintain the the cross-platform requirement of Greenstone.

The scheduler requires the following parameters from the user as input: 1) the collection to be rebuilt, 2) the full import command that is required to import documents into the collection, 3) the full build command required to construct the indices and classifiers for the collection, and 4) a specification of either an hourly, daily, or weekly build.

For example, if we want the collection $pics$ to be scheduled for construction on a daily basis, the arguments would look something like this:

```
schedule.pl pics "import.pl -removeold pics"
       "buildcol.pl -removeold pics" daily
```

where $pics$ is the name of the collection, "import.pl -removeold pics" is the Greenstone command line argument for importing documents into the collection $pics$, "buildcol.pl -removeold pics" is the Greenstone command line argument for creating the required indices and classifiers for $pics$, and daily indicates that the collection will be scheduled for construction on a daily basis.

Using the arguments provided by the user, the scheduler performs two main tasks: 1) create a script that automates the building of a collection, and 2) create a crontab record that schedules the execution of the script at specified intervals. Each task is described in detail in the next two sections.

4.1. Automation Script Generation

The scheduler works for import and build commands that contain any number of parameters. It contains all instructions that are required for building the specified collection on any of the supported platforms. The commands include those for setting the Greenstone environment variables required for the collection building process, the import command, the build command, and the commands required for

```
00 0 * * * /gsdl/collect/pics/gsdl.pl
```

```
00 0 * * * c:\gsdl\collect\pics\gsdl.pl
```

activating the new indices and classifiers.

Figure 5 shows a sample automation script for Linux that contains all of the instructions for building the collection *pics*. Figure 6 shows an automation script for the same collection that is set up for Windows. Both scripts are created based on the arguments given above in the schedule.pl example. In addition, both are set up in a similar way. The first four instructions set the Greenstone environment variable that are required for the import.pl and buildcol.pl commands. The next two instructions are the import.pl and buildcol.pl commands that are provided to the scheduler. Finally, the remaining instructions handle the copying of indices and classifiers in order to activate the collection. It should be noted that the automation script for Mac OS X is almost identical to the one for Linux.

4.2. Crontab Generation

When scheduling the construction of a collection, the user can specify an hourly, daily or weekly build. Currently, an hourly build is scheduled for the beginning of the hour, a daily build is scheduled for midnight, and a weekly build is scheduled for Sunday morning at midnight. Although monthly and yearly builds can be easily added, we opted to only support the first three so that the code required for the scheduler is kept to a minimum.

First, a crontab record is created to execute the automation script at the specified interval. Figure 7 depicts the daily crontab record for the automation script on Figure 5, while Figure 8 depicts the crontab record corresponding to Figure 6.

After creating the crontab record, it is added to the crontab file if no crontab record exists for the same collection. Otherwise, it will replace an existing crontab record. How this takes place depends on the system:

- *Windows*. Pycron maintains one crontab file only. Therefore, the file is opened, and the new crontab record will either replace the existing record, or is appended to the end of the file. This activates the records.
- *Linux and Mac OS X*. Activating the crontab record requires running a system crontab command with a crontab file as a parameter. This will delete any other scheduled tasks. Therefore, any crontab records are obtained first and placed in a temporary file. Then, the new crontab record is added to the temporary file (or used to replace an existing record for the same collection). Finally, the system crontab command is activated using the temporary file.

Once the scheduler is finished, cron constructs the collection repeatedly at the interval specified by the user.

5. Evaluation

In this section, we discuss the performance of the Greenstone scheduling module. The focus of our experiments is to evaluate the scheduler for correct execution in specific situations. For correct execution, we looked at the following:

- *Crontab*. Does the scheduler generate a correct crontab record? A crontab record is correct if: 1) it is accepted by the crontab program (Linux and Mac OS X only), and it causes cron to execute the automation script at the proper time.
- *Automation Script*. In addition, does the scheduler generate the correct automation script to build a collection. An automation script is generated correctly if the collection it is created for is rebuilt correctly given the parameters that are specified when the script is created.

  We conducted three experiments. The first is an hourly build of one collection. The second is an hourly build of two collections. The third is a daily build of one collection. All three experiments were performed on both Linux and Windows. These tests were not performed extensively on Mac OS X. However, the scheduler was executed many times on this platform to ensure that it does work.

5.1. Hourly Build of One Collection

The focus of this experiment is to ensure that schedule.pl produced a correct crontab record and automation script to re-build a collection of images every hour for 24 hours. To determine that each execution of the automation script was successful, we looked at the following. For Linux, cron was configured to send an email message containing the output of the automation script. The email message for each of the 24 builds contains an output for a successful Greenstone build. For Windows, pycron maintains a log that contains two records for each automation script execution – one record indicating the start of execution, and one record indicating the end of execution and a return code. The terminating records for all 24 builds have a return code of zero,

which indicates a successful execution of the automation script.

In both cases, a visual inspection of the collection was also performed periodically to verify the success of the scheduler.

5.2. Hourly Build of Two Collections

The focus of this experiment is to ensure that if multiple collections are scheduled to be rebuilt at the same time, they are successfully rebuilt. We scheduled the building of two collections on hourly intervals for 24 hours. A perusal of email messages (for Linux) and the log (for Windows) verify that both collections were rebuilt successfully.

5.3. Daily Build of One Collection

The focus of this experiment is twofold. The first is to ensure correct daily execution of an automation script generated by the scheduler. The second is to simulate a situation of a collection that has images added to it on a daily basis. We schedule one collection to be built daily for seven days. The collection starts with 10 images, and 10 images are added daily. In addition to the successful daily build, the collection did reflect the addition of the new images that arrived daily.

6. Scenario

In this section, we present an overview of an application that has the requirement of incremental insertions and updates. Such an application can and will make use of the Greenstone scheduler.

The application in question is a prototype for a post-secondary information resource [4]. This information resource is constructed from existing websites from participating institutions. Information is collected and unified on the different degree and diploma programs offered by all post-secondary institutions across a specific geographical region. This region can be local, provincial, national or even global. The purpose of the information resource is to provide a one-stop-shop for students to browse and search for information on programs that they may consider pursuing at the post-secondary level. Also, placing all post-secondary program information in one location allows students to compare programs across different institutions easily.

An important feature of a post-secondary resource is its ability to be kept current. Therefore, its contents must be updated on a regular basis, whether it is daily, weekly or monthly. Any changes to existing programs at a participating post-secondary institution must be updated in the resource, any new programs must be added, and any programs that not longer exist must be removed. Any changes must take place in a timely manner.

The Greenstone scheduler can perform the insertion, deletion and updates required by a post-secondary information resource. The scheduler works in conjunction with other modules in Greenstone to retrieve and process the necessary information from the World Wide Web. Then, the scheduler generates the appropriate scheduled task to build the collection so that it is kept current with the existing websites.

7. Conclusions

We proposed and implemented a scheduler for Greenstone. It allows users to automate the maintenance of their collections by providing a few simple parameters. The scheduler interacts with a resident task scheduler on the local operating system, which results in a minimal but powerful tool. Several experiments are performed to show the correct execution of the the scheduler for different build times and for different numbers of collections.

Currently, we are working on functionality to remove existing tasks from the crontab file that are no longer required. Other future directions of work include the following.

First, we will modify the scheduler to accept specific time and day values for an hourly, daily, or weekly scheduled build of a collection. Second, we want to extend the Greenstone Librarian Interface (GLI) so that users can choose a scheduled build option, and also specify the time and day of the week for the maintenance of their collection. The GLI already creates the appropriate import and build commands given the options that the user selects, and these can provided as parameters to the scheduler.

References

[1] G. Kalab. Pycron. Website, visited July 2007. `http://www.kalab.com/freeware/pycron/pycron.htm`.

[2] C. Lagoze, S. Payette, E. Shin, and C. Wilper. Fedora: an architecture for complex objects and their relationships. *International Journal on Digital Libraries*, 6(2):124–138, 2006.

[3] E. Nemeth, G. Snyder, and T. R. Hein. *Linux Administration Handbook*. Prentice-Hall, 2007.

[4] W. Osborn, S. Fox, and S. O'Shea. A unified resource for post-secondary program information. Working Paper, 2007.

[5] R. Tansley, M. Bass, and M. Smith. Dspace as an open archival information system: Status and future directions. In *Proceedings of the 10th European Conference on Digital Libraries (ECDL 2006)*, September 2006.

[6] P. Vixie. Vixie cron for FreeBSD. Website, last visited August 2007. `http://www.freebsd.org/cgi/cvsweb.cgi/src/usr.sbin/cron/`.

[7] I. Witten and D. Bainbridge. *How to Build a Digital Library*. Morgan Kaufmann, 2002.