Gadget Creation for Personal Information Integration on Web Portals

Chia-Hui Chang, Shih-Feng Yang, Che-Min Liou
National Central University, Taiwan
chia@csie.ncu.edu.tw, {tomelf, chemin}@db.csie.ncu.edu.tw

Mohammed Kayed
Beni-Suef Universiy, Egypt
mskayed@yahoo.com

Abstract

Although the ever growing Web contain information to virtually every user's query, it does not guarantee effectively accessing to those information. In many situations, the users still have to do a lot of browsing in order to fuse the information needed. In this paper, we propose the idea of gadget creation such that extracted data can be immediately reused on personal portals by existing presentation components, like map, calendar, table and lists, etc. The underlying technique is an unsupervised web data extraction approach, FivaTech, which has been proposed to wrap data (usually in xml format). Despite the efforts to utilize supervised web data extraction in RSS feed burning like OpenKapow and Dapper, there's no research on incorporating unsupervised extraction method for RSS feeds or gadget creation. The advanced application in gadget creation allow immediate use by users and can be embedded to any web sites, especially Web portals (personal desktop on Web). This paper describes our initiatives in working towards a personal information integration service where light-weight software can be created without programming.

1 Introduction

Although the ever growing Web contain information to virtually every user's query, it does not guarantee effectively accessing to those information. In many situations, the users still have to do a lot of browsing in order to access those information. For example, considering the efforts one has to do everyday: email checking (yahoo, msn, school/work mail), news reading (CNN, USA today, ESPN sports), discussion boards monitoring (e.g. Google forums, Yahoo groups), social communities (Facebook, Myspace), online documents (Google Docs&Spreadsheet, Zoho.com), calendar, investments and banks, etc. It surely takes a lot of time browsing through all these Web sites. As another example, querying the price for airline tickets with different departure date or marking the locations on Google map for the schools in your neighborhood. Both will take multiple and repetitive operations for the users to accomplish the task. In fact, there are a bunch of such special integration applications since web sites can not predict in advance all users' need.

The first example presents the need to monitor Web sites, a particular kind of information integration, while the second one describes various information needs requested by users, a more general information integration case. To keep updated with the newest information, some maintain "favorite folders" to record all the URLs, some use email notifications, while others use multiple starting URLs in their browsers (e.g. Firefox) to check the newest information simultaneously. However, bookmarks do not provide instant update which is desirable for changing information, especially on the Web. Using email notification can also increase the cost of managing emails. Multiple starting URLs may be a solution except that it loads the whole Web page but only a portion of the Web page is targeted. Thus, some researchers have proposed the idea of clipping a portion of information from target web sites to incorporate it into the end user site.

An alternative way to solve this problem is to have many gadgets / widget / portlet / modules (i.e. lightweight software) to retrieve the information from web sites on behalf of users such that any update can be monitored. This idea of integration for personal use has become more and more popular in recent years and attracted many efforts in the development of such platform, e.g. iGoogle, Netvibes, Pageflakes, Protopage, MyAol, MyYahoo, ExciteMix, etc. Such personal portal interface allows users to view their emails, RSS feeds, and discussion forums in one place and disclose the most updated content in a dashboard without visiting all web sites, thus becoming the start page of many people. Moreover, with most of the platforms open and free to add any third party modules, it becomes even popular and powerful since more and more tools are created.

In such personal web portals, users can assemble their favorite gadgets, feeds, social networks, email, videos and blogs on one fully-customizable page. One can also turn any Web page into a dashboard component by clipping any selected area (a segment of HTML source code) as a new widget to their personal dashboard. In fact, the way it allows users to add arbitrary modules makes such platforms perfect for personal information center. Thus, gadgets and personal web portals would be a form and platform where composite services are delivered.

In this paper, we propose the idea to have gadgets performing repetitive operations for users, e.g. marking all addresses found on input Web pages on Google map, or querying the ticket fare for different departure dates (and show the result as a fare calendar), etc. To delegate such repetitive operations, the user can specify the input, output and the data between them for creating gadgets. The first (input) step guides the gadget to fetch HTML pages from the Web, then the second step extract the desired data to be used in the output step. Finally, predefined display modules like lists, tables, calendars and maps can be selected as the presentation interface. Thus, a user not only has a copy of some Web clip from the resource Web site but also assemble the data by some operations imposed by the selected display module.

The rest of the paper is organized as follows: Section 2 describes the system architecture and user interface demonstration. An example of gadget creation is illustrated in section 3. Finally, conclusion and future work are summarized in section 4.

2 System Architecture

We explain the system architecture using Figure 1. An user can create new gadgets for his/her own information need by specifying the input method, output module and desired data to be displayed. The corresponding modules will fetch pages specified by users, extract data embedded in them via unsupervised wrapper technology, and display the extracted data in spreadsheets for selection. Finally, gadgets can be generated based on the fetch plan, extraction rules and selected output module. Once the gadget is created, it can be added to Web portals like iGoogle or any web site by copying the generated code.

For output, there are four predefined display modules: list, table, calendar, and map, where the last two allow users to compare data in time axis and space axis, respectively. These will cover two of the most common scenarios when the users would need help on his/her very own information needs: the first one is monitoring task (e.g. check email, library book loan, currency rate, etc.), the other is comparing task that involves repetitive process to be carried using current Web services. For example, mark a set of address on map, query ticket fare for different departure date with calendar display module, or compare the result for a list of queries in a tabular way, etc.

For input, there are two input methods that users can start

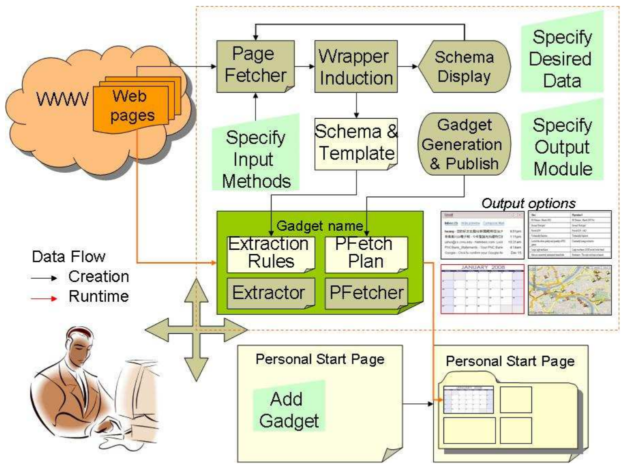

Figure 1. The proposed system architecture for Gadget Creation.

with: a user can specify either a set of static URLs, or a starting URL which contains some query form with associated queries. In practice, more specification is required since the desired data may spread across several pages, especially for monitoring tasks. In the past works (like AsBye or Robomaker), this would require extraction of links to be followed. However, as we will see later, other input pages can be specified easily by selecting URL-type data for fetching.

One of the key difference of this Gadget on Demand (GoD for acronym) system to previous works (such as ASByE [4] and Lixto [1]) is the adoption of unsupervised data extraction technology such that dynamic information in a set of similar pages can be automatically extracted without users' annotation. FiVaTech [3] as a page-level data extraction technique, can deduce the template of the input pages and represent the data as a schema tree. Once the schema and templates are detected, the users are presented with data in several spreadsheets: one spreadsheet for each set and a particular spreadsheet for all nonset data. Based on the spreadsheets, users can select columns to be displayed or for further page fetching and extraction.

In the following sections, we shall focus on how page fetch plan is composed and the preparation of the selected columns for final display. However, before we move on, we shall explain how data from unsupervised extraction system can be visualized to help input page specification.

2.1 Data Extraction and Visualization

For dynamic contents which are generated in response to a submitted query or accessed only through a form, each dynamic Web page is created by embedding a data instance

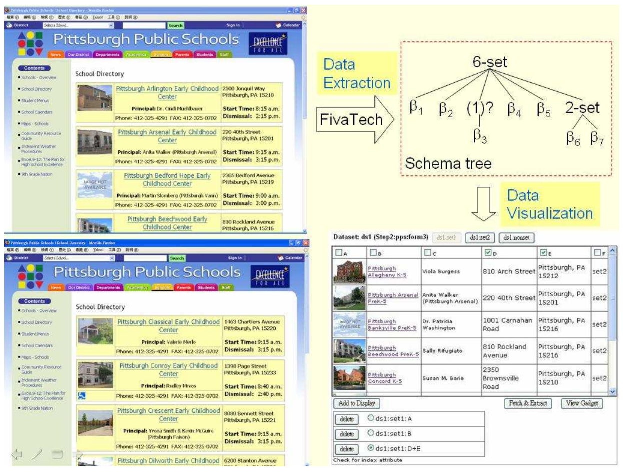
Pittsburgh Public Schools
Data Extraction
FivaTech
6-set
β_1 β_2 (1)? β_4 β_5 2-set
β_3 β_6 β_7
Schema tree
Data Visualization
Dataset: ds1 (Step2:pps:form3)
Add to Display
Fetch & Extract
View Gadget

Figure 2. The schema (top-right) of two input pages (left) from PPS web site and spreadsheet-like data visualization (bottom-right).

x (for the page schema) into a predefined template. As a result, input pages generated from the same program are similar in appearance, making detection of template and schema possible. Generally speaking, templates, as a common model for all pages, occur quite fixed as opposed to data values which vary across pages. Finding that common template usually requires multiple pages (e.g. RoadRunner) or a single page containing multiple records (e.g. IEPAD) as input. There have been many researches on unsupervised data extraction since then, especially for search result records (e.g. DEPTA, ViPER, WSE). Meanwhile, alternative researches on page-level extraction can also be found (e.g. RoadRunner, EXALG, FiVaTech). The comparison of the various approaches can be found in [2]. In this paper, we have adopted FiVaTech [3] as the data extraction module. Given any number of pages as input, the output of FiVaTech is a schema file and one XML file containing the extracted data for the pages.

For the data embedded, we can model its schema by a tree structure where the leaves are basic types and each internal node can be a type constructor of tuple, option, set or disjunction (depending on the number of instances for each instantiation). An example schema, detected from two Web pages of HomePopular (www.homepopular.com), is shown in Figure 2. The schema tells us, each Web page from HomePopular is composed of a 4-tuple including one basic data, two sets (one for sponsored links and the other for search results), and one optional data.

Although tree representation is one way to understand the structured data inside input pages, it is not easy to capture the meaning of each basic types without proper at-

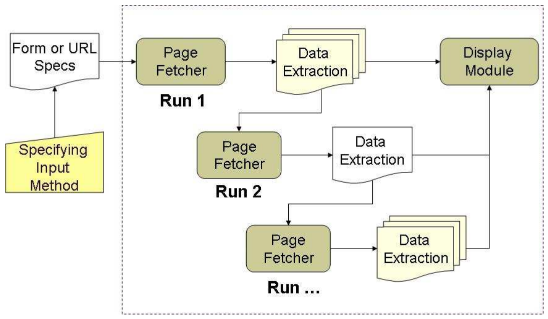

Figure 3. Multiple run execution.

tribute names. Therefore, we choose to display the instances for each basic type by grouping them into several spreadsheets, that is by the set they belong. For example, the structured data in Figure 2 are then presented by three spreadsheets: two for set1 and set2, respectively and one for non-set data (including the basic data and expanded optionals. Note that all tuple, option and disjunction data types can be expanded with the basic types in them producing a flat structure view for each set and noset. For example, the nonset part of the example schema contains a total of three basic types where the last two come from the option type of size 2. As for the case when sets occur inside a set, each set will still be visualized through a separate spreadsheet. Meanwhile, there will be links to subsets under the parent set.

The incorporation of unsupervised data extraction technology holds the vantage of easier maintenance as wrapper can be deduced automatically from input pages without users' annotation. Thus, detection of schema or template change can be conducted smoothly. When such an event is detected, the gadget users can be warned and further schema matching algorithm can be incorporated to fix the gadget.

2.2 Page Fetch Plan Specification

As mentioned above, there are two ways to to specify the input pages: either a set of URLs or a set of queries to a Web form. However, when there are links to be followed inside the input pages, the system needs to know how to extract those links. Thus, data extraction also plays an important part in page fetching. There have been works on combining supervised data extraction technique in page acquisition tool, e.g. Laender et al.'s DEByE (Data extraction by example) environment and ASByE (Agent specification by example) [4] for wrapper generation by example. However, no effort has been reported on how to use unsupervised data extraction in fetch plan composition.

As shown in Figure 3, a fetch plan in the GoD system consists of multiple runs. The first run can be specified by a set of URLs or a set of queries, while other runs are determined by URLs extracted from the fetched pages. The system provides "Add URL" button for users to add URL one by one, as well as "Get FORM" button for users to choose a form in the specified URL and add query one by one. Note that our system acts as a proxy between end users and the target Web site that users just entered. Thus, every added query is recorded immediately without waiting the complete response from target Web server, saving the unnecessary waiting time. Another advantage is to avoid the challenging problem of client-side scripts which could change the literal content input by users in an HTML form.

For the remaining runs, the URLs can be specified by selecting part of the extracted data set (similar to select columns of interest for display). Although it seems that the subtask of data extraction is mixed with page fetch module, there is no complicate training procedure (for extraction rule) in between since we utilize unsupervised data extraction technique here. This design allows users to specify the URLs that are embedded in pages without training the extraction rules for desired URLs as in past work.

With the rendering of spreadsheet, the user can then specify the desired columns for output or further fetch and extraction for the following runs.

3 Case Study: PPS Map Gadget

In this section, we use an example to show how repetitive task could be carried by gadget creation. This case study is from Pittsburgh Public Schools (PPS) web site. The school directory in PPS allows users to find school addresses based on type (e.g. Early Childhood Center, K-5, 6-8, High School, All, etc.). To locate the position on the map, users need to copy each address to online map such as Google map and mark it one by one. Such operations may need to be repeated for all schools. Thus, the desired output for this task is a map display module with marked locations for schools extracted from PPS. The following section describes how a non-expert users could interact with the system to make such a widget and the user-interface design principle and alternatives in details.

To automate such a repetitive task, the input pages need to be collected first for further processing. As each page usually shows a limited number of items, the widget needs to know all the HTTP connections to fetch input pages. For the PPS example, we can specify a set of queries (from 1 to 3) for the third form in the starting URL (http://www.pps.k12.pa.us /143110127917194 37/FlexBase/FlexBaseDisplay.asp? DirectoryID=53&DisplayType=C&Field0=&Field7=K-5+ Schools&submit=submit) to fetch all the input pages.Next, the data extraction procedure will be conducted to show the spreadsheet containing the school list. We can then add the columns of interest to display list. Suppose we are also interested in the detail information of each school as pointed by links, we may check the column containing the links for fetch and extraction. This will then trigger the page fetch and data extraction modules for working.

When the final step is completed, a light weight software like gadget for iGoogle (or module for Netvibes) can then be created with associated XML file containing an reference to find rest of the data (extraction rules, fetch plans) at our Web site and instructions on how to process and render the gadget. The generated gadget. Another distribution method is to provide the code for users to copy to their web site as their wish.

4 Conclusions and Future Work

As more and more users are involved in content creation on the Web, the necessity to assemble and reuse existing data/module has become manifest. By choosing personal portal as the platform and incorporating wrapper technology in the creation of gadgets, users can do more integration based on their personal information needs. Although we use iGoogle as our platform, similar modules on other Web portals like Netvibes and Protopages can be produced. By adopting wrapper technology, we are able to integrate more Web sites with existing modules and help users integrate their information.

Acknowledgement

This project is sponsored by National Science Council, Taiwan under grant NSC96-2627-E-008-001.

References

[1] R. Baumbartner, S. Flesca, G.Gottlob. "Visual Web Information with Lixto," Proceedings of the 27th International Conference on Very Large Data Bases, VLDB 2001. Pages: 119 - 128.

[2] C.H. Chang, M. Kayed, M.R. Girgis, K.F. Shaalan. "A Survey of Web Information Extraction Systems," IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 10, 2006. Pages: 1411 - 1428.

[3] M. Kayed, C.H. Chang, M.R. Girgis, K.F. Shaalan, "FiVaTech: Page-Level Web Data Extraction from Template Pages," Workshops on Data Mining in Web2.0 Environment, 2007.

[4] J. P. Lage, A. S. da Silva, P. B. Golgher, A. H. F. Laender. "Automatic generation of agents for collecting hidden Web pages for data extraction," Data & Knowledge Engineering, Vol. 49, No. 2, 2004. Pages: 177-196.