Adopting A Common Data Model for End-User Web Programming Tools

David F. Huynh
Metaweb Technologies Inc.
631 Howard St. Suite 400
San Francisco, CA 94105, USA
david@metaweb.com

David R. Karger
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139, USA
karger@csail.mit.edu

ABSTRACT

We observe that end-user web programming tools frequently deal with web data and that there is a clear synergy between these tools. Making them interoperate will benefit both users and researchers alike. The first step toward this goal is to adopt a common data model, such as RDF. More research challenges remain to make them interoperate, such as fleshing out a common API and a unified user interface mechanism for the user to orchestrate the tools to perform complex tasks.

Keywords

End user programming, Web, data modeling, screen scraping, RDF.

1. INTRODUCTION

End-user web programming (EUWP), like programming in general, often involves dealing with data. For example, the Creo+Miro system [7] watches the user interact with tidbits of information on a web page (e.g., looking up the nutritional information for an ingredient in a recipe) and generalizes the user's actions so that they can be applied in similar contexts (e.g., looking up nutritional information for all ingredients of any recipe). The Web Summary Framework [6] lets the user extract data from several web pages and web sites and store it all together as research materials for a particular goal (e.g., buying a house).

Not only do these tools have to deal with data, they have to deal with *generic* data found on the Web, ranging from recipes to houses for sale, sharing little if anything in common. In fact, these end-user programming tools are most useful when applied to data in the long tail of the Web precisely because the data in the tail is not popular enough to warrant investments for commercial tools and users have to resort to self-help programming.

1.1 The Case for Interoperability

If we think of each EUWP tool not as a monolithic application but as a set of elementary features, such as the feature for plotting a map, many of which deal with data, there is clearly an opportunity for the user to mix and match the features from different EUWP tools, getting the best of all worlds. Moreover, whereas EUWP tools are intended for the user to take matter into her own hands when web sites don't let her do what she wants, the ability to mix and match the EUWP tools' features lets she take matter into her own hands once more when each standalone EUWP tool doesn't let her do what she wants, but a combination of these tools does.

Interoperability among these tools is also a boon for researchers. Instead of having to support every feature for making a research project compelling, even if many of these features do not constitute the core contribution of the research, now a researcher can rely on features borrowed from other tools to make her case. In addition, how features from different tools are combined is in itself open for research and can now be explored by users themselves.

1.2 Requirements for Interoperability

There are two requirements for interoperability. First, the tools themselves must be able to programmatically interoperate. For example, they must be able to invoke one another and pass data among themselves.

Second, there must be a user interface mechanism for the user to orchestrate the tools to work together toward some complex goal. For example, to orchestrate Unix command line tools, the user pipes one tool's standard output to another tool's standard input. The operating system's shell provides a command line syntax—a user interface mechanism nevertheless—for the user to perform this orchestration.

In this paper, we focus primarily on the first requirement. The second requirement is a whole topic of research unto itself. We will only discuss it briefly at the end.

1.3 Contribution

To satisfy the first requirement for interoperability, we propose the adoption of a common data model in all EUWP tools. This data model serves as a blackboard through which the tools can share their data. It should be capable of expressing as many different types of data as possible and should not require that these types be known a priori. We suggest RDF as a possible data model to adopt.

1.4 Outline

For the remainder of the paper, we first present an extensive scenario that illustrates how these various EUWP tools might be used together in a complex task. In section 3, we describe the requirements for the common data model and suggest a possible off-the-shelf solution. We provide related work in section 4, discuss future work in section 5, and conclude in section 6.

CHI 2009, April 8–12, 2009, Madrid, Spain.
ACM

2 SCENARIO

There have been many research tools built for end-user programming on the Web, and we wish to illustrate through an extensive scenario here how they might be used together.

Ann and Bob have just moved to Boston and are looking for a new house to buy. There are many factors for them to consider. First, the house must be located where it is convenient for them to get to work either by bike or by public transportation, and for their children to walk to school. Of course the neighborhood must be safe from crimes and should be far from high traffic zones, and there must be essential shops (grocery store, pharmacy) within some distance. The house itself should meet certain criteria: at least four bedrooms, two bathrooms, a nice floor plan, a double garage or street parking, large enough shaded back yard and front yard, good views from the master bedroom's window, built within the last 20 years, etc. Of course, Ann's and Bob's budget might force them to compromise on some of these criteria. Since they don't know the Boston area well, they want to take time to learn and do their research carefully. This might take months.

There are several web sites that list houses for sales, such as realtor.com, but they don't all cover the same listings. To get a complete and accurate picture, Ann and Bob want to integrate the data from as many of those sites as possible. The data must first be extracted from those sites, and that task is best done with a tool like the Web Summary Framework, Marmite [12], or Sifter [9].

To keep the data up-to-date, Ann and Bob need to perform data extraction periodically. Each time they have to log into each of those sites, browse to the right page, perform a particular search, and then screen scrape the results. Because some of the sites use AJAX, it's not possible to bookmark their searches; instead they have to follow a sequence of UI actions. To avoid all of this tedious, repetitive work, Ann and Bob use CoScripter [11] to automate the whole process. CoScripter stores Ann's and Bob's search criteria and enters those criteria into each site. As Ann and Bob get to know the housing market better, their criteria change, and it's convenient to have to specify their criteria in one place only, namely in CoScripter.

As some of these listing sites don't provide sufficiently powerful browsing and searching features, Ann and Bob still need to filter the extracted data down further to just those listings that match all of their criteria. This is best done with the faceted browsing features offered by Piggy Bank [8].

There are also various other types of data to include in the research, such as data about schools, shops, crimes, public transportation routes and stops, biker-friendly streets, etc. This data can be scraped once since it does not change much. However, since it consists of quite a variety of types of data, it requires a very flexible data model. Moreover, because there are many sites to scrape, Ann's and Bob's scraping efforts might not produce properly aligned data sets on the first try. Ann and Bob might even split the scraping work between them and try to merge their results later. So they will need to realign the data sets after scraping, using a tool like Potluck [10].

Given that all the data needed is properly extracted, aligned, and kept up-to-date, Ann and Bob still need to visualize it in ways to help them learn about the housing market as well as compare their options and ultimately make their decisions. Note that not all criteria can be specified objectively (e.g., "large enough shaded yards" and "nice floor plan" are subjective criteria), so there is no way that the computer can figure out the best choice automatically. Humans are still needed in the loop, and they are best facilitated with visualizations of the data that help them make their decisions. The Web Summary Framework, Marmite, Vispedia [5], Piggy Bank, and Potluck all support custom visualizations. The Web Summary Framework is particularly designed for composing an information dashboard from research materials from the Web.

Certain visualizations might require appending additional data to what was collected so far. For example, to get a map view, Ann and Bob need to geocode all the addresses associated with the homes they found. This requires processing the previously extracted data to produce arguments that can be submitted to a geocoding service. For example, the addresses gotten from scraping might not contain the city name as they are all in Boston, so the city name must be appended to them before geocoding.

The whole process of scraping for new data and constructing visualizations can be orchestrated using CoScripter. And when Ann and Bob change one of their criteria, it's just a matter of updating that criterion with CoScripter and then letting CoScripter invoke the other tools to extract data and generate new visualizations.

To understand the housing market in Boston, Ann and Bob also read local only newspapers on the subject, paying close attention to the ongoing market crash. These news articles don't contain structured data, but they provide valuable insights that Ann and Bob want to refer to often. A tool like Piggy Bank allows them to tag these articles with keywords and save their links into the same database as all other bits and pieces of research materials.

Finally, having done so much work for their research, Ann and Bob want to help other people who are also buying houses by sharing all of their end-user programming scripts. Of course their scripts might not cover all conceivable needs of everyone else, but given interoperability among the tools, other people can easily use their own tools to modify the whole workflow to suite their purpose.

3. COMMON DATA MODEL

As illustrated above, various EUWP tools might be orchestrated to achieve a complex goal. The first step to enable such interoperability among the tools is a common data model. In this section, we discuss the requirements for such a data model and suggest an off-the-shelf data model to adopt.

3.1 Requirements

The data extracted from the Web or to be entered into the Web often consists of chunks smaller than whole files. For example, a single product search result page contains many product records, and simply storing that page as a single file jumbles the different products together, making it harder to work with them as individual entities. Furthermore, existing browser-based user tools have demonstrated compelling features that leverage the data not as text documents but as structures: records can be sorted and filtered by fields in Piggy Bank and the Web Summary Framework. This requires that structure queries be supported on the data. Thus, the shared data should reside in something that resembles a database more than a file system.

While conventional relational databases support structured queries, they require the schemas of the data be designed once initially and kept constant. However, in order to support browser-based tools built for research, innovative and diverse by their very nature,

the common data model must allow new schemas to be incorporated into the system during runtime.

Finally, data transiting from and to the Web can be web-like in form, containing links between seemingly disparate domains of information. For example, the data about a particular book may link not just to the book's author and its reviews, but, as that book is based on history, to real people, places, artifacts, etc. that it mentions. For another example, the data about a writer may link not just to her books but, if she is also an actress, to the films in which she has performed. If such links cannot be made, then certain information important to the user might be left unrecorded. Thus, the common data model must be web-like, capable of supporting arbitrarily interconnected rather than disjoint schemas.

3.2 RDF as A Possible Solution

One such data model is RDF [2], designed specifically for exchanging and integrating web data. It is a semantic network model consisting of "nodes and arrows", or entities and relationships, which are identified globally with URIs. Piggy Bank and Sifter are already using RDF and have demonstrated the ease with which any kind of web data can be incorporated into the system.

4 RELATED WORK

The strategy of using a common data model to facilitate cooperation between independently developed tools is a time-proven strategy; file systems and clipboard models are proofs. There have been many efforts to enrich file systems so that they can store more properties per file, e.g., Apple Newton's Soup [3], WinFS [4].

Note that the web browser Firefox has started to allow extensions to instantiate and use its internal SQLite databases. These could be used to hold instances of the shared data model, and communicate them from one component to another. The Mozilla Weave project [1] will improve this support further and let the data migrate seamlessly between different computers through a central data cloud service. However, since Weave is providing a common storage and replication model rather than a common data model, data for one extension remains opaque to another (much as files for one application generally remain opaque to another application). Thus it is not possible for the extensions to cooperate accidentally unless their developers decide to cooperate. By advocating a common data model, we allow unplanned (by the developers) cooperation to happen more often.

5 DISCUSSION

As mentioned in the Introduction, there are two requirements for interoperability: programmatic interoperability and user interface mechanism for orchestrating interoperation. Adopting a common data model is only the first step toward programmatic interoperability. It is only the first step because, in order for such a tool as CoScripter to automate a long process involving several other tools as described in our scenario, there must also be a common API for invoking these tools. In fact, even in the absence of CoScripter, one tool might want to directly invoke another.

The requirement for a user interface mechanism for orchestrating interoperation is another challenge to address. Two possible solutions come to mind. First, a structured data copy-and-paste framework can be provided so that the user can copy data from one tool to another. Second, a central store can be provided for all data. Each subset of data within this store can be referenced by a name or a query. In order to pass a blob of data from one tool to another, the user can name that blob of data, or ask the first tool for a query that describes that data, and then enter the name or the query into the second tool.

While those two possible solutions might be sufficient for some cases, they can be cumbersome or unusable when the interaction between the tools is more coupled. Consider using Sifter first for adding faceted browsing to a search result set, and then deciding subsequently to use the Web Summary Framework to compose a summary template for some of the search results. Sifter's philosophy is to support direct manipulation to avoid requiring the user to label the fields. If the only way for the Web Summary Framework to get data from Sifter is through the central store or a one-time copy-and-paste operation, and data is the only thing passed between the tools, then Sifter must force the user to label the fields explicitly first because otherwise, without the presentation template from the original site that Sifter has extracted, the fields can easily become meaningless and confused in the Web Summary Framework. For example, the "sale price" and "used price" fields are hard to distinguish if they are not explicitly labeled.

6 CONCLUSION

In this paper, we observe that EUWP tools frequently deal with web data and that there is a clear synergy between the tools. Making them interoperate will benefit both users and researchers alike. The first step toward this goal is to adopt a common data model, such as RDF. More research challenges remain to make them interoperate, such as fleshing out a common API and a unified user interface mechanism for the user to orchestrate the tools to perform complex tasks.

REFERENCES

[1] Mozilla Labs » Weave. http://labs.mozilla.com/projects/weave/.

[2] Resource Description Framework (RDF) / W3C Semantic Web Activity. http://www.w3.org/RDF/.

[3] Soup (Apple Newton), Wikipedia. http://en.wikipedia.org/wiki/Soup_(Apple_Newton).

[4] WinFS, Wikipedia. http://en.wikipedia.org/wiki/WinFS.

[5] Chan, B., L. Wu, J. Talbot, M. Cammarano, and P. Hanrahan. Vispedia: Interactive Visual Exploration of Wikipedia Data via Search-Based Integration. IEEE InfoViz 2008.

[6] Dontcheva, M., S.M. Drucker, G. Wade, D. Salesin, M.F. Cohen. Summarizing Personal Web Browsing Sessions. UIST 2006.

[7] Faaborg, A. and H. Lieberman. A Goal-Oriented Web Browser. SIGCHI 2006.

[8] Huynh, D.F., S. Mazzocchi, and D.R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. ISWC 2005.

[9] Huynh, D.F., R.C. Miller, and D.R. Karger. Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities. UIST 2006.

[10] Huynh, D.F., R.C. Miller, and D.R. Karger. Potluck: Data Mash-Up Tool for Casual Users. ISWC 2007.

[11] Leshed, G., E. Haber, T. Matthews, T. Lau. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise. SIGCHI 2008.
[12] Wong, J. and J. Hong. Making Mashups with Marmite: Towards End-User Programming for the Web. SIGCHI 2007.