



Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Potluck: Data mash-up tool for casual users

David F. Huynh*, Robert C. Miller, David R. Karger

MIT Computer Science and Artificial Intelligence Laboratory 32 Vassar St., Cambridge, MA 02139, USA

ARTICLE INFO

Article history:

Received 28 May 2008

Received in revised form 19 August 2008

Accepted 15 September 2008

Available online 1 November 2008

Keywords:

Mash up

Drag and drop

Faceted browsing

Simultaneous editing

Ontology alignment

End-user programming

Semantic web

RDF

ABSTRACT

As more and more reusable structured data appears on the Web, casual users will want to take into their own hands the task of mashing up data rather than wait for mash-up sites to be built that address exactly their individually unique needs. In this paper, we present Potluck, a Web user interface that let's casual users—those without programming skills and data modeling expertise—mash up data themselves.

Potluck is novel in its use of drag and drop for merging fields, its integration and extension of the faceted browsing paradigm for focusing on subsets of data to align, and its application of simultaneous editing for cleaning up data syntactically. Potluck also lets the user construct rich visualizations of data in-place as the user aligns and cleans up the data. This iterative process of integrating the data while constructing useful visualizations is desirable when the user is unfamiliar with the data at the beginning—a common case—and wishes to get immediate value out of the data without having to spend the overhead of completely and perfectly integrating the data first.

A user study on Potluck indicated that it was usable and learnable, and elicited excitement from programmers who, even with their programming skills, previously had great difficulties performing data integration.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

The construction of a Web 2.0 mash-up site is typically done by programmers. In this paper, we introduce Potluck, a tool that lets casual users—non-programmers—make mash-ups by themselves:

- Potluck allows the user to merge fields from different data sources, so that they are treated identically for sorting, filtering, and visualization. Fields are merged using simple drag and drop of field names.
- Potluck provides an efficient means for the user to clean up data syntactically, homogenize data formats, and extract fields syntactically embedded within existing fields, all through the application of simultaneous editing [9].
- Potluck supports faceted browsing [19] to let users explore and identify subsets of data of interest or subsets of data that need alignment and clean up.

In contrast, today's mash-up construction can only be done by programmers using complex technologies as it involves many technical challenges, particularly:

- scraping data from the original sites, where it may be hidden behind complex queries and web templates;
- aligning the original sites' data into a single coherent data model; and
- creating an effective visualization of the merged data.

These challenges are only worth to overcome for mash-ups that will appeal to many people, and they prevent the construction of mash-ups that are personal or narrow in appeal, serving only a few users and giving little return on investment of efforts and resources. For example, a high-school student writing a term report on the knowledge and use of mycology (mushrooms) among Polynesian tribes will be unlikely to find a mash-up site containing data on both mycology as well as Polynesians. She will also unlikely find enough resources (money and programming skills) to get such a site built quickly enough to meet her deadline, if ever built at all. The long tail of mash-up needs is thus left unanswered. As more and more data becomes readily accessible on the Semantic Web, the number of mash-ups potentially useful but never built increases significantly and spells a huge opportunity miss, reducing the value proposition of the Semantic Web itself.

We conducted a user study of Potluck and report the results here, which show that Potluck is a viable mash up solution for casual users and that it even has features desired by programmers.

* Corresponding author. Tel.: +1 617 733 3647.

E-mail addresses: dfhuynh@mit.edu, dfhuynh@csail.mit.edu (D.F. Huynh), rcm@csail.mit.edu (R.C. Miller), karger@csail.mit.edu (D.R. Karger).

2. Scenario

Before describing the user interface of Potluck, we motivate it with a scenario that illustrates various idiosyncrasies of personal mash-up construction. Let us be optimistic that within a decade, the Semantic Web will be prevalent and RDF data will be everywhere. This scenario argues that even in this future world, users will *still* face problems making mash-ups between data sources.

In 2017, a historian named Henry is documenting the first cases of a rare genetic disease called GD726. These first cases occurred in the Valentine family in the 1820s. He wants to include in his final report a genealogical tree of the Valentine family, annotated with the disease's infliction, as well as a comprehensive table of the Valentines' data in an appendix.

Like most historians, Henry is not a programmer but he is experienced in collecting and managing data in his professional work. The proliferation of RDF means that Henry does not need programming skills to scrape HTML himself: all the information needed for his research has been converted into RDF by various independent organizations and individuals, both professionals and enthusiasts. Henry thinks it would be trivial to simply pool the RDF together and call it done.

Henry tracks down various birth certificate issuing offices and death certificate issuing offices where the Valentines lived for their RDF data. He notes that some offices use `dc:date` in their data to mean "birth date," some to mean "death date," and some "certificate issuing date." It would be disastrous to consider all the `dc:dates` the same even if the same predicate URI is used.

Henry also tracks down hospital records, which contain `hospital:tod` (short for "time of death"). Hence, `hospital:tod` is equivalent to some of the `dc:dates`. It would be hard to match `hospital:tod` with `dc:date` based on string analysis alone, yet match for some of the cases only.

The records all have geographical location names, but these names are not fully qualified. Those responsible for digitizing them thought that since all locations were within their country, there was no need to include the country name.

As a consequence, Henry needs to append the country name to the many location names in order to map them.

People's names are encoded in two different forms: "first-name last-name" in some data sets and "last-name, first-name" in others. Nick names are also present (e.g., "Bill" instead of "William," and "Vicky" instead of "Victoria").

The hospital records also pose problems. While most of their admittance dates are in ISO 8601 format, a few are of the kind "Easter Day 1824." Such sloppiness has been observed in industrial and institutional databases, and should be expected on the Semantic Web.

Despite all these problems, there is one good thing about the data: Henry can reliably get the mother and father of each Valentine through the `gen:mother` and `gen:father` predicates, which seem to be very widely adopted. This helps Henry construct a genealogical tree visualization.

However, as males and females both have equal chance of passing on GD726, Henry wants to treat `gen:mother` and `gen:father` the same while tracing the disease through the family. Unfortunately, adding an `owl:sameAs` equivalence between those two predicates will break his genealogical tree.

While all parties involved in this scenario acted logically and responsibly, Henry still ends up with a mess of RDF. To fix up the data, Henry must be able to:

- Merge `dc:dates` into *several* groups (the birth dates and the death dates) even though they all use the same predicate URI.

This requires distinguishing the fields by their origins rather than just by their URIs.

- Merge `gen:mother` and `gen:father` together in some situations while keeping them separate in other situations. This precludes the simple approach of adding `owl:sameAs` statements in the data model to implement equivalences.
- Edit the data efficiently to unify its syntax.
- Fix up the data iteratively as he learns more and more about the data.

There is one more challenge that Henry faces which we choose not to address at the moment: that of entity alignment. A person might be referred to by different URIs in different sources, and Henry must make them all equivalent to one another. We leave this task to future work, as we have not found any reasonable user interface solution to it.

3. User interface


We now describe Potluck's user interface, showing how it addresses the problems in the scenario above. The reader is encouraged to view a screencast to understand Potluck's interactivity: <http://simile.mit.edu/potluck/>.

Fig. 1 shows the starting screen of Potluck where the user can paste in several URLs of Exhibit-powered web pages and click *Mix Data*. This results in Fig. 2, which lists data records from the original web pages. The records are interleaved by origins—the pages from which they have been extracted—to ensure that some records of each data set are always visible.

Fields are rendered as *field tags*: `label`, `position`, and `title`. Field tags are color-coded to indicate their origins: blue from one source and pink from another in Fig. 2. Three core fields, `label`, `type`, and `origin`, are automatically assigned to all records and their tags are colored gray. `origin` is generated from each source URL. `label` and `type` are reserved as they are core Exhibit fields and assumed to be used in a uniform manner across different exhibits.

Fields from different origins having the same name are considered different. For example, while `phone` means office phone, `phone` might mean secretary's phone. Or more dangerously, `dc:date` in the scenario (in Section 2) has several distinct meanings. These semantic differences, subtle or significant, might or might not be important to one particular user at one particular moment in time. Keeping the fields apart rather than automatically merging them together allows the user to make the decision whether or not to merge.

3.1. Creating columns and facets

A field tag can be dragged and dropped onto the gray column to the left (Fig. 2) to create a new column listing that field, or onto the gray box to the right to create a facet for filtering by that field. Fig. 3 shows a newly created column. A column or facet can be moved by dragging its field tag and dropping the tag between other columns or facets. Deleting a column or facet (by clicking it ) removes the column or facet from the display but does not delete the corresponding field's data.

3.2. Merging fields

A field tag can be dropped onto an existing column or facet in order to make that column or facet contain data for both the original field and the newly dropped field. Such an operation creates a *merged field*, whose field tag is rendered as a visual juxtaposition of

Download English Version:

<https://daneshyari.com/en/article/557907>

Download Persian Version:

<https://daneshyari.com/article/557907>

[Daneshyari.com](https://daneshyari.com)