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
`{dfhuynh, rcm, karger}@csail.mit.edu`

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 lets 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.

Keywords: mash up, drag and drop, faceted browsing, simultaneous editing, ontology alignment, end-user programming, semantic web, RDF.

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.

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.

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.

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:date`s 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:date`s. 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:date`s 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.

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/.

Figure 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 Figure 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 Figure 2. Three core fields, `label`, `type`, and `origin`, are automatically assigned to all records and their tags are colored gray. 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 automati-
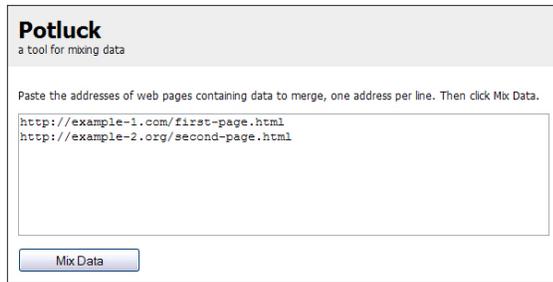
Figure 1. The starting screen of Potluck takes URLs to Exhibit-powered web pages. Clicking Mix Data yields the mixed data in a screen like Figure 2.
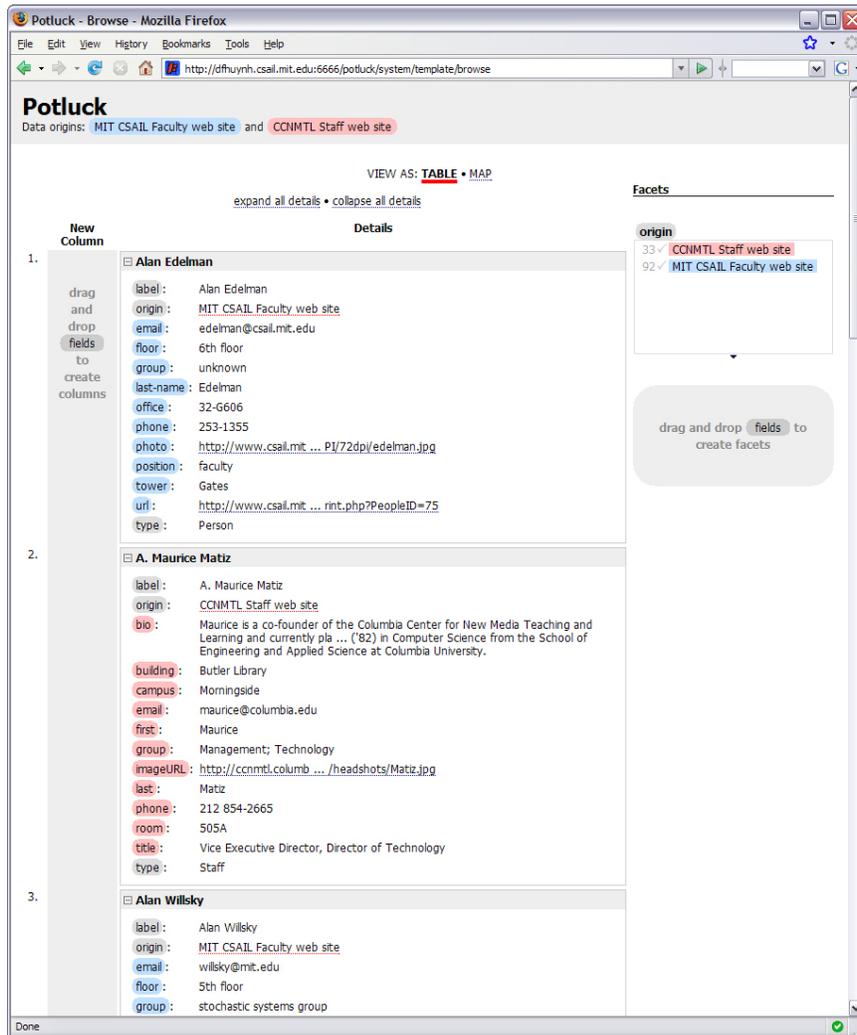
Figure 2. Potluck's user interface shows data that has just been mixed together but not yet processed by the user. Fields are rendered as draggable "field tags," color-coded to indicate their origins. There are two drop target areas for creating columns and facets.

cally merging them together allows the user to make the decision whether or not to merge.

Creating columns and facets. A field tag can be dragged and dropped onto the gray column to the left (Figure 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. Figure 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 its ⊗) removes the column or facet from the display but does not delete the corresponding field's data.

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 the original tags, taking on a pill-shaped form position title . Figure 4 shows several columns and facets of merged fields. Merged field tags can be dragged and dropped just like elemental field tags can in order to create new columns and facets, or to merge into other existing columns and facets.

Creating a merged field does not disturb the elemental fields. Thus, in the scenario, it would be easy to have `gen:mother` and `gen:father` merged together for one purpose while keeping them separate for another purpose, all at the same time. Furthermore, the merging operation is *not* transitive, so that, say, merging fields `mother` and `father` together (to mean `parent`) and then `mother` and `grandmother` together (to mean `female ancestor`) does *not* force all three fields to be merged into `mother/father/grandmother`.

Simultaneous editing. The edit link next to each field value opens up the Simultaneous Editing dialog box where the values of that field can be edited *en masse* (Figure 5). The concept of simultaneous editing originated from LAPIS [9], a text editor that displays several keyboard cursors simultaneously on a text document, generalizes the user's editing actions at one cursor, and applies them to the text at the rest of the cur-
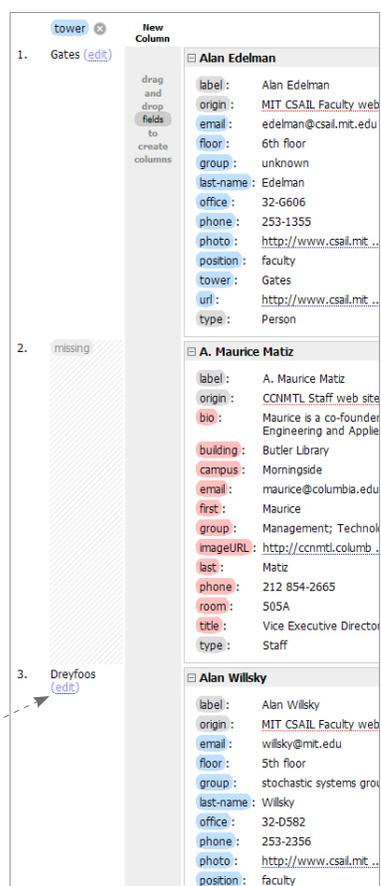

Figure 3. Potluck renders a new column to the left when tower is dropped into the New Column drop target. Since the second record is not from the same origin as the dropped field, its cell in that column shows missing .
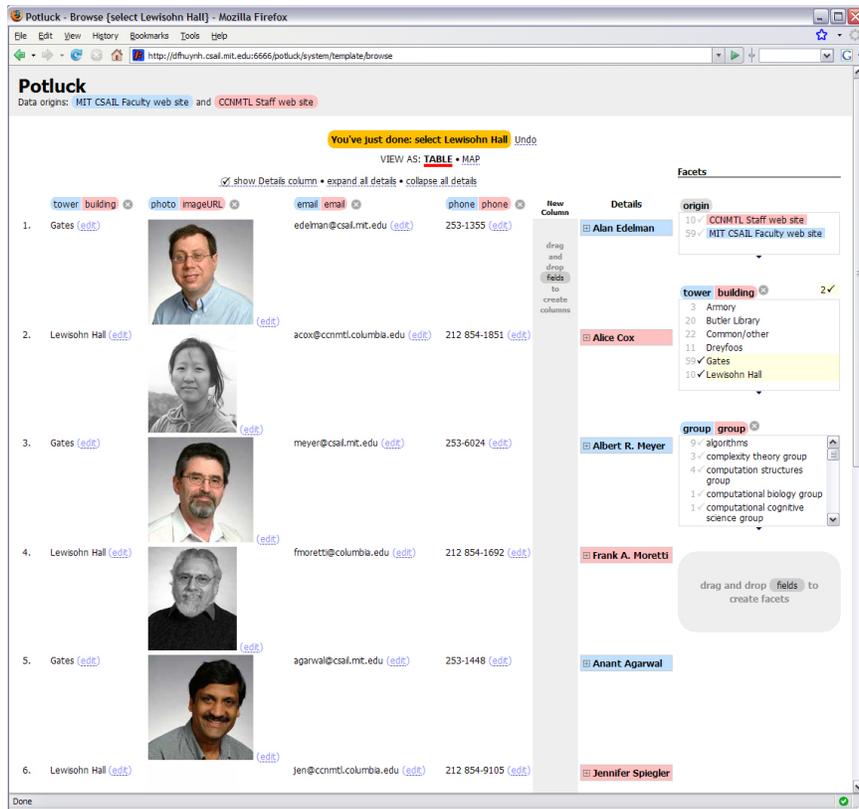
Figure 4. A screen shot of Potluck showing several columns and facets of merged fields. The records' details have been collapsed to make space for the columns.

sors. Based on the user's mouse clicks, LAPIS guesses how to divide the text document into records (often into lines or paragraphs) and where the cursors should be placed within those records (e.g., after the second word of the third sentence in each paragraph). Whereas LAPIS has to guess what a record is for the purpose of simultaneous editing, Potluck already has the field values conveniently separate. Potluck groups field values into columns by structural similarity, e.g., the phone numbers in the second column all have area code 212. These columns serve to visually separate out values of different forms, call out outliers (such as "Easter Day 1824" in the scenario), and let the user edit different forms differently. The user can click on any field value to give it keyboard focus, and editing changes made to it are applied to other values in the same column in a similar fashion. The multiple cursors in Figure 5 give visual feedback of the simultaneous editing operations in progress.

If a value appears in several records it is shown in only one entry in the dialog box. In the scenario, if the nickname "Bill" appears in three records, the user can
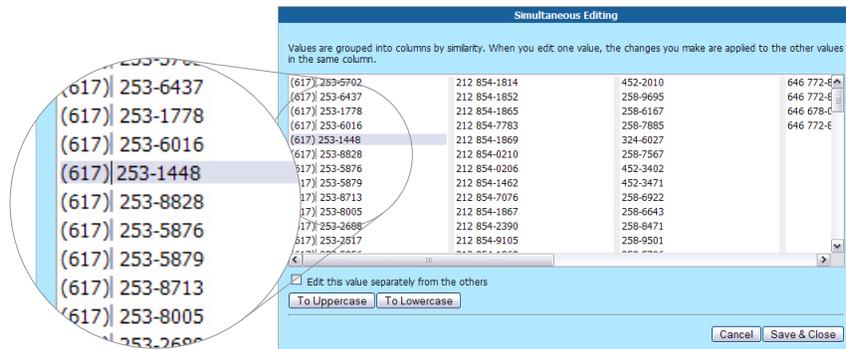
Figure 5. Potluck's Simultaneous Editing dialog box lets the user change several similar values simultaneously by editing any one of them. Multiple keyboard cursors are shown and any editing change to the focused value is immediately reflected in the other values.

click on its single entry in the dialog box, set the checkbox Edit this value separately from the others, and change it to "William" to correct all three records.

Simultaneous editing is useful for correcting inconsistencies between data sets that occur many times, such as prefixing area codes to phone numbers and wrapping existing area codes in parentheses. It is also useful for reformatting a field, such as changing "first-name last-name" into "last-name, first-name", and for making a new field out of an existing field, such as extracting building numbers (32) from within office numbers (32-582).

Faceted browsing [19] is a browsing paradigm in which a set of records can be filtered progressively along several dimensions in any arbitrary order. For example, a set of recipes can be filtered by picking an ingredient first, a cooking method second, and a cuisine finally, or by picking a cuisine first, then an ingredient, and a cooking method finally depending on which order suits the user best. Because the data Potluck handles is often multidimensional, faceted browsing is useful in Potluck as it is designed for exploring multidimensional data in flexible, user-controllable ways. Exploration is needed for identifying and selecting out just the subset of data that is useful as well as for isolating on records that need cleaning up. All faceted browsers so far assume that they are fed data sets that have already been cleaned and made coherent using other tools. In Potluck, we extended faceted browsing for the mash-up task in which data arrives from many sources.

If within a facet there are records for which the corresponding field is missing, the facet explicitly shows a choice for filtering to those records (Figure 6). This visual element, not present in conventional faceted browsing interfaces, also serves to remind the user that, if that field is an elemental field instead of a merged field, the field is not present for records in other data sets.

While working with multiple data sets at the same time, it can be easy to forget that an elemental field from one data set does not exist in the others. Whenever a facet choice causes all records from an origin to be filtered out completely, that
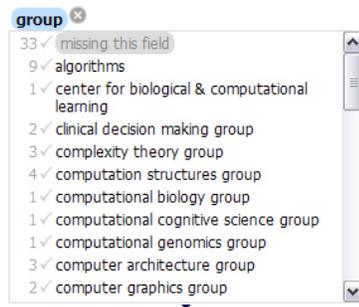
Figure 6. If inside a facet there are records for which the corresponding field is missing, the facet shows `missing this field` as a choice so that the user can get to those records.
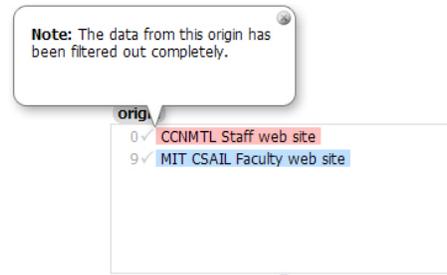
Figure 7. The origin facet does not remove choices for which there are no records. Moreover, it pops up messages to call the user's attention to those filtered out origins.

origin remains in the origin facet and a message is popped up drawing the user's attention to it (Figure 7).

Visualizations. Potluck currently provides two visualizations: a tabular view and a map view. We have discussed the tabular view extensively while the map view can be seen in the screencast accompanying this paper. Any field containing street address or as latitude/longitude pair can be dropped onto the map view to plot the records without doing any programming.

Miscellany. Potluck provides drop down menus on left clicks as alternatives to drag and drop in order to increase the likelihood that the user succeeds at finding some way to accomplish a task. The browser's Back and Forward buttons can be used to redo and undo user actions. Like contemporary highly interactive web interfaces, Potluck also shows the most recently done or undone action and provides a link to undo or redo it.

4 Implementation

Potluck consists of two components: a server-side component implemented as a Java servlet, responsible for retrieving the data within the Exhibit-embedding web pages to mix; and a client-side component implemented in Javascript on top of the Exhibit API [5], responsible for all the user interface interactivity.

Merged fields are implemented as query unions: when the values of a merged field are requested, the values of each elemental field in that merged field are returned in a single result set. No equivalence is added into the data model so that merging operations will not be transitive and so that the original elemental fields can still be used in isolation even after they have been merged.

Simultaneous editing is implemented in Javascript. Each field value is parsed into a sequence of features. Features are runs of digits, of letters, or of white spaces, or individual punctuation marks and symbols. For example, "733-3647" is broken down into three features: the run of digits "733", the symbol "-", and

the run of digits "3647". Field values are then clustered into columns by greedily aligning these sequences of features.

As the user moves the keyboard cursor, makes selections, and edits the text of one value, the cursor positions are generalized to be relative to the features of the field value being edited (e.g., "second character from the beginning of the third last feature"), and then those generalized cursor positions are turned into absolute cursor positions of each of the other field values in the same cluster and used to apply the edit. Secondary cursors are rendered using colored `<span>` elements.

As the clipboard Cut and Paste operations cannot be reliably detected in web browsers, we must support cut-and-paste in the simultaneous editing paradigm using a trick. When some text is inserted, we check if that same piece of text has previously been deleted in one edit action and assume that what has taken place is a cut-and-paste operation. Note that this trick works only for cut-and-paste, not copy-and-paste.

5   Evaluation

We conducted a user study on Potluck to ascertain whether people could learn how to use Potluck as well as to discover usability problems. We also wanted to observe how people use Potluck in an open-ended task using their own judgement about which fields to merge and edit, and how to display them.

5.1 Design and Procedure

This study consists of two tasks: a structured task during which the subjects performed simple steps to familiarize themselves with Potluck, and an unstructured task during which the subjects performed an open ended task based on the skills they had just acquired.

In Task #1, subjects browsed two web pages containing information about 92 people in a lab and 33 people in another lab, and answered questions about these people in ways that required the pages' faceted browsing features (e.g., "how many people are in the Gates tower?"). This warm-up exercise let the subjects learn about the data and about faceted browsing. Then the subjects were asked to use Potluck to mix the data in those pages and to achieve the following goals:

- create a column listing the buildings where people work and make sure the column is filled in with information for people from both labs;
- create a column listing people's phone numbers and edit them to have the form (xxx) xxx-xxxx, using 617 for phone numbers without area code;
- create a column listing people's job titles;
- create a facet of people's job titles, use it to filter for people in directing positions (directors and co-directors), and determine how many such people there are in each lab; and
- create a column of people's last names and sort it in ascending order.

These instructions were *not* worded in low-level details (e.g., click this button) so to allow the subjects the opportunities to learn how to use Potluck's user interface by themselves and to allow us the chance to discover usability problems.

In Task #2, the subjects were asked to use Potluck to mix data from two Exhibit-powered web pages of 40 + 55 publications and then mock up a single web page where hypothetical visitors could conveniently sort and filter through all of those publications as if the data came from a single source. The subjects were left to their own discretion to decide which columns and facets to create, although some examples were given in case the subjects were not familiar with the domain.

5.2 Participants and Apparatus

Six subjects (2 male, 4 female) from a university community were recruited by sending an e-mail message to a mailing list and posting paper ads around our college campus. Four were younger than 30, and two older than 30. They were two students (mechanical engineering and computer science), two researchers (applied math and brain and cognitive science), a lawyer, and an applied math consultant.

We also recruited five subjects (1 male, 4 female) from our campus' libraries, who worked with data in their daily job. Two were in their 20s, one 30s, and two 40s. We wanted to observe if librarians, who have more experience working with data, would use Potluck differently.

There were a total of 11 subjects, referred to as G1 to G6 from the general university population and L1 to L5 from the libraries. All browsed the Web at least a few times a day and used Firefox as one of their primary browsers.

Subjects received $10 each for participating in a 30 – 45 minute study session. All sessions were conducted by one investigator on a single computer (Pentium 4 2.53GHz, 1.00GB) with an 18" LCD flat panel at 1600×1200 resolution in 32-bit color and a Dell two-button mouse with wheel, running Microsoft Windows XP. The study facilitator observed the subjects and took written notes.

5.4 Results

All subjects were able to learn Potluck's user interface with little guidance and to complete the user study's tasks within 45 minutes. We now report the results in more details and point out usability issues to address in the future.

Columns. Nine subjects out of 11 used only drag and drop to create columns. This indicates that the relevant visual cues may be sufficiently strong. One of the other two subjects, G5, used the Create Column menu command at first but adopted drag and drop later. L1 used only the menu command.

G5 and L5 had difficulty understanding that dragging a field tag to create a column automatically filled up the whole column with data wherever the field was available. They continued to drag the same field tag out again and again for each row, paying no attention to the data already shown in the column. We can improve the drag feedback to better indicate that the whole field is being dragged, such as showing ghosted images of several field values near the mouse pointer.

All except one subject merged columns using drag and drop; G2 used the corresponding menu command. G3 and G4 expected the phone fields from both sources in Task #1 to be merged automatically. We can make Potluck suggest such merging if the field names match precisely.

Most subjects merged `position` and `title` together into one column, but one subject also included `group` to more fully qualify `position`. This was because most `title` values were more specific than most `position` values (e.g., "Codirector of Marketing" vs. "professor"). This operation was actually not what the subject intended (as he verbalized): the operation performed a set union of two fields instead of a string concatenation. But as Potluck rendered the `group` value after the `position` value for each record (e.g., "professor, computer architecture"), the visual outcome looked right and the subject was contented. However, sorting on this merged field would produce random orders and a facet created out of this merged field would list the `group` and `position` values separately, not paired together. Potluck should support string concatenation and suggest it as an alternative to merging whenever the two fields involved come from the same source. Note that in the scenario in section 2, concatenation is probably not the desired choice when the `gen:mother` field is dropped onto the `gen:father` field even though both come from the same source.

Facets. All subjects used drag and drop to create facets. Two subjects initially created facets using the corresponding menu command, but they discovered the drag and drop alternative and did not revert to the menu. Merging facets was done solely using drag and drop. We note that the field tags on facets do not offer any menu (an oversight in our implementation); only field tags in the details column and in the column headers support menus.

Some subjects tended to drag already merged field tags from columns to create facets while the others dragged elemental field tags from the Details column to create merged facets. The latter behavior forced the user to re-merge fields she has already merged; this is both inefficient and error-prone as some subjects did forget to re-merge fields. Potluck should have automatically suggested or defaulted to the merged field whenever an elemental field that has been merged is used.

G4 did not initially merge facets in Task #1 to filter for people in directing positions. Instead, he created two facets, `position` and `title`, from the two sources separately and used missing this field to achieve the goal. In either facet, he selected directing positions as well as missing this field so that records in the other source were not excluded. This required on his part deeper understanding of how faceted browsing worked. When asked to achieve the goal without using missing this field , he discovered that he could merge facets.

Simultaneous editing. All subjects were able to edit several phone numbers using the simultaneous editing feature. G1 anticipated this feature even before clicking edit, asking out loud, "can I edit them all together?" She later used the feature to delete first names from people's full names to get a field of last names. This action properly utilized the simultaneous editing feature's power but destroyed data (the first names). We can make Potluck alert the user of this loss and offer a convenient way to apply the edit on a copy of the original field instead.

G4 tried to move the leading "A" from publication titles to the end (e.g., "Tale of Two Cities, A") using simultaneous editing (a reasonable goal) but the facilitator explained that the feature did not support that case. L2 and G6 tried to swap first names and last names so that publications could be sorted by their authors'

last names. L2 selected a last name in the simultaneous editing dialog box and dragged it to the front of the corresponding first name; unfortunately, a bug prevented this from working. G6 used keyboard shortcuts for cut-and-paste and succeeded. These subjects' actions indicated some intuitiveness in using cut-and-paste and drag-and-drop for simultaneous editing.

G3 expressed that she did not want to see all phone numbers in the simultaneous editing dialog box but only their templates. G5 and L3 edited only the first group of phone numbers, and L4 edited only the first and third groups, neglecting the groups that were not scrolled into view. To avoid such oversight, we will need to make it more apparent which pieces of data an edit does and does not affect.

Librarians vs. general subjects. Among the five librarians, four were catalogers (who characterize physical artifacts such as books and enter their metadata into databases), and one was a programmer responsible for integrating large data sets. While the catalogers showed no significant difference with the general subjects in their use of Potluck, the programmer, L1, was clearly an outlier: he created 10 columns and 7 facets in total. He was very excited about the user interface of Potluck and described his data integration work, consisting of manual data entry and Perl scripting, to be tedious and painful.

G6, who also needed programming skills to deal with some data for his work, expressed equal enthusiasm for Potluck. He used simultaneous editing to swap first name and last name. Thus, while there was no noticeable difference between the subjects from the general population and the librarians, who purportedly work with data and metadata on a daily basis, there was a difference between programmers and non-programmers in how much they appreciated Potluck. Programmers, who have encountered difficulties in dealing with data even with their programming skills, appreciated Potluck more. Non-programmers accomplished the tasks in the study equally well, but were not equally excited perhaps because there was not enough reusable data on the Web for them to feel the need to mash up data themselves. However, when there will be more reusable data in the future, interfaces like that of Potluck have the potential to level the playing field for non-programmers, making them as effective as programmers for the task of mashing up data.

6 Related Work

Piggy Bank [6] and Tabulator [3] illustrate how personal mash-up tools with generic functionality can let individual Web users satisfy their own unique combinations of mash-up needs. While Tabulator consumes only RDF data, Piggy Bank can also run screen scrapers to extract data from HTML pages—a design to overcome the current shortage of RDF data on the Web. While the proliferation of structured data on the Web will hopefully eliminate the need to scrape fragile HTML, the data still has to be cleaned up and aligned before it can appear coherent to the user and thus become useful. The amount of broken HTML code at the present forebodes messy real-world RDF in the future, broken perhaps not just in syntax but also in semantics. Personal mash-up tools like Piggy Bank and Tabulator have largely ignored this problem, assuming that once data is in RDF, conventional visualizations

and browsing techniques previously designed to work on individual coherent data sets can be applied readily on mashed up data.

An exception to these tools is WebScripter [18], which lets casual users create coherent reports out of data collected from several sources, offering data alignment features for that purpose. Although the desired target audience is casual users, WebScripter's interface is still expert-oriented, full of jargon such as "DAML", "class", "instance", etc. WebScripter offers no feature for fixing data at the syntactic level (e.g., swapping first name and last name) and it has not been formally evaluated on actual users.

There are also web applications such as Dapper [1] that let users scrape existing web sites for data and serve that data up in structured formats as "feeds," or make use of data already scraped by other people. These web applications still offer very limited capabilities for cleaning up data, integrating data in different schemas, and constructing rich visualizations. Those that offer more capabilities, such as Ning [2], require programming.

In research, data alignment tools have been built mostly for experts and research has focused primarily on data modeling theories and automated agents for ontology alignment [7, 8, 10, 11, 12, 13, 14, 16] rather than on user interfaces for making practical use of aggregated data. Because they specialize only in data alignment, they implicitly assume that users work with the data in delineated stages, first aligning the data and cleaning it up, and then making use of that data in some other tools. We believe that users actually work iteratively on data, switching from aligning and clean up the data to using the data, and back, as they get to know the data better over time. Furthermore, these tools tend to work on ontological abstractions, basing their interface interactions on concepts such as classes. Casual users have little knowledge about data modeling and ontological abstractions, and little interest in learning.

Faceted browsing was pioneered by Yee et. al. [19] and recently adopted by the Semantic Web research community [4, 15, 17]. We extended faceted browsing to allow selection of records that are missing data. Simultaneous editing has been researched previously [9] and we extended it with the automatic clustering of field values into columns.

7 Conclusion

In this paper, we presented Potluck, a tool for casual users—those without programming skills and data modeling expertise—to mash up data by 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 the simultaneous editing technique 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 solicited excitement from programmers who, even with their programming skills, had great difficulties in integrating data.

8 Acknowledgements

9 References

[1] Dapper: The Data Mapper. http://www.dapper.net/.

[2] Ning - Create your own Social Networks! http://www.ning.com/.

[3] Berners-Lee, T., et. al. Tabulator: Exploring and Analyzing linked data on the Semantic Web. *SWUI* 2006.

[4] Hildebrand, M. and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. *ISWC* 2006.

[5] Huynh, D., D. Karger, and R. Miller. Exhibit: Lightweight Structured Data Publishing. *WWW* 2007.

[6] Huynh, D., S. Mazzocchi, D. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. *ISWC* 2005.

[7] Kalfoglou, Y. and M. Schorlemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, Volume 18, Issue 1, 2003.

[8] Laera, L., V. Tamma, J. Euzenat, T. Bench-Capon, and T. Payne. Reaching Agreement over Ontology Alignments. *ISWC* 2006.

[9] Miller, R. and B. Myers. Multiple Selections in Smart Text Editing. *IUI* 2002.

[10] Mitra, P., N. F. Noy, and A. R. Jaiswal. Ontology Mapping Discovery with Uncertainty. *ISWC* 2004.

[11] Mocan, A., E. Cimpian, M. Kerrigan. Formal Model for Ontology Mapping Creation. *ISWC* 2006.

[12] Noy, N. F. and M. A. Musen. SMART: Automated Support for Ontology Merging and Alignment. *Workshop on Knowledge Acquisition, Modeling and Management*, 1999.

[13] Noy, N. F. Semantic Integration: A Survey Of Ontology-Based Approaches. *SIGMOD Record*, Special Issue on Semantic Integration, 33 (4), December, 2004.

[14] Noy, N. F., M. A. Musen. THE PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. *International Journal of Human-Computer Studies*, 2003.

[15] Oren, E., R. Delbru, and S. Decker. Extending faceted navigation for RDF data. *ISWC* 2006.

[16] Scharffe, F. Schema Mappings for the Web. *ISWC* 2006.

[17] schraefel, m. c., D. Smith, A. Russel, A. Owens, C. Harris, and M. Wilson. The mSpace Classical Music Explorer: Improving Access to Classical Music for Real People. *Music-Network Open Workshop*, Integration of Music in Multimedia Applications, 2005.

[18] Yan, B., M. Frank, P. Szekely, R. Neches, and J. Lopez. WebScripter: Grass-roots Ontology Alignment via End-User Report Creation. *ISWC* 2003.

[19] Yee, P., K. Swearingen, K. Li, and M. Hearst. Faceted Metadata for Image Search and Browsing. *CHI* 2003.