Wrangler: Interactive Visual Specification of Data Transformation Scripts

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, Jeffrey Heer
{skandel, paepcke, jheer}@cs.stanford.edu; joeh@cs.berkeley.edu

ABSTRACT
Though data analysis tools continue to improve, analysts still expend an inordinate amount of time and effort manipulating data and assessing data quality issues. Such "data wrangling" regularly involves reformatting data values or layout, correcting erroneous or missing values, and integrating multiple data sources. These transforms are often difficult to specify and difficult to reuse across analysis tasks, teams, and tools. In response, we introduce *Wrangler*, an interactive system for creating data transformations. Wrangler combines direct manipulation of visualized data with automatic inference of relevant transforms, enabling analysts to iteratively explore the space of applicable operations and preview their effects. Wrangler leverages semantic data types (e.g., geographic locations, dates, classification codes) to aid validation and type conversion. Interactive histories support review, refinement, and annotation of transformation scripts. User study results show that Wrangler significantly reduces specification time and promotes the use of robust, auditable transforms instead of manual editing.

Author Keywords
Data, cleaning, transformation, validation, visualization, programming by demonstration, mixed-initiative.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: UI

INTRODUCTION

Despite significant advances in technologies for data management and analysis, it remains time-consuming to inspect a data set and mold it to a form that allows meaningful analysis to begin. Analysts must regularly restructure data to make it palatable to databases, statistics packages, and visualization tools. To improve data quality, analysts must also identify and address issues such as misspellings, missing data, unresolved duplicates, and outliers. Our own informal interviews with data analysts have found that these types of transforms constitute the most tedious component of their analytic process. Others estimate that data cleaning is responsible for up to 80% of the development time and cost in data warehousing projects [4]. Such "data wrangling" often requires writing idiosyncratic scripts in programming languages such as Python and Perl, or extensive manual editing using interactive tools such as Microsoft Excel. Moreover, this hurdle discourages many people from working with data in the first place. Sadly, when it comes to the practice of data analysis, "the tedium is the message."

Part of the problem is that reformatting and validating data requires transforms that can be difficult to specify and evaluate. For instance, analysts often split data into meaningful records and attributes—or validate fields such as dates and addresses—using complex regular expressions that are error-prone and tedious to interpret [2, 23]. Converting coded values, such as mapping FIPS codes to U.S. state names, requires integrating data from one or more external tables. The effects of transforms that aggregate data or rearrange data layout can be particularly hard to conceptualize ahead of time. As data sets grow in size and complexity, discovering data quality issues may be as difficult as correcting them.

Of course, transforming and cleaning a data set is only one step in the larger data lifecycle. Data updates and evolving schemas often necessitate the reuse and revision of transformations. Multiple analysts might use transformed data and wish to review or refine the transformations that were previously applied; the importance of capturing data provenance is magnified when data and scripts are shared. As a result, we contend that the proper output of data wrangling is not just transformed data, but an editable and auditable description of the data transformations applied.

This paper presents the design of *Wrangler*, a system for interactive data transformation. We designed Wrangler to help analysts author expressive transformations while simplifying specification and minimizing manual repetition. To do so, Wrangler couples a *mixed-initiative user interface* with an underlying *declarative transformation language*.

With Wrangler, analysts specify transformations by building up a sequence of basic transforms. As users select data, Wrangler *suggests applicable transforms* based on the current context of interaction. Programming-by-demonstration techniques help analysts specify complex criteria such as regular expressions. To ensure relevance, Wrangler enumerates and rank-orders possible transforms using a model that incorporates user input, transform parameters, and the use rate, diversity, and specification difficulty of suggested transform types. To convey the effects of data transforms, Wran-

Submitted for review to CHI 2011.

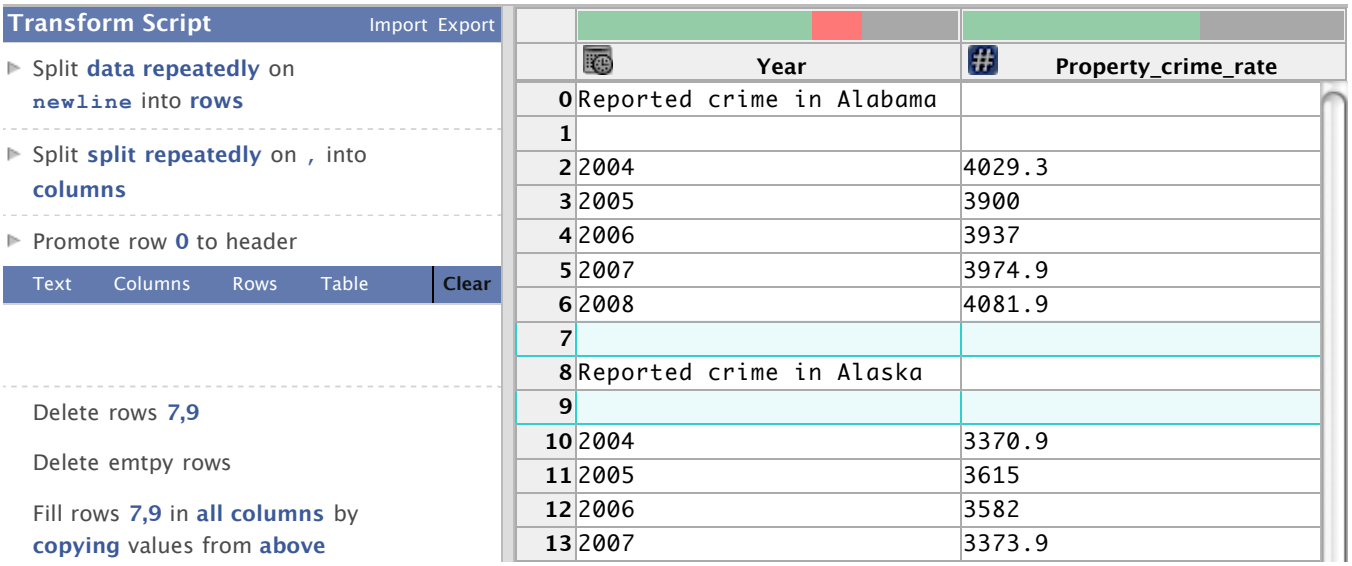

Figure 1. The Wrangler Interface. The left panel contains (from top-to-bottom) a history of transforms, a transform selection menu, and automatically suggested transforms based on the current selection. Bold text within the transform descriptions indicate parameters that can be clicked and revised. The right panel contains an interactive data table; above each column is a data quality meter.

gler provides short *natural language descriptions*—which users can refine via interactive parameters—and *visual previews* of transform results. These techniques enable analysts to rapidly navigate and assess the space of viable transforms.

As analysts transform data, their steps are recorded in a script to facilitate reuse and provide documentation of data provenance. Wrangler's *interactive history viewer* supports review, refinement, and annotation of these scripts. Wrangler's high-level language supports a variety of runtime platforms: Wrangler scripts can be run in a web browser using JavaScript or translated into MapReduce or Python code.

We also present a controlled user study comparing Wrangler and Excel across a set of data wrangling tasks. We find that Wrangler significantly reduces specification time and promotes the use of robust transforms rather than manual editing. Wrangler is one piece of a larger effort to address bottlenecks in the data lifecycle by carefully integrating insights and methods from the HCI and database communities.

RELATED WORK

The database and machine learning communities have contributed a number of algorithmic techniques for aiding data cleaning and integration. These techniques include methods for detecting erroneous values [10, 11], information extraction [1, 24], entity resolution [6], type inference [7], and schema matching [9, 20]. In the Wrangler interface we seek to surface such techniques in an accessible manner.

A number of commercial and research systems provide graphical interfaces leveraging the above methods. Many of these tools provide interfaces for schema matching or entity resolution [3, 9, 15, 22]. Toped^{++} [23] is an interface for creating *Topes*, objects that validate and transform data. Topes support transformations such as text formatting and lookups, but provide little support for filtering, reshaping, or aggregation. Bellman [5] helps users understand the structure and quality of a database, but does not enable transformations.

Many data cleaning applications apply direct manipulation and programming-by-demonstration (PBD) methods to specific cleaning tasks. Users of SWYN [2] build regular expressions by providing example text selections and can evaluate their effect in visual previews. Potluck [13] applies simultaneous text editing [18] to merge data sources. Karma [25] infers text extractors and transformations for web data from examples entered in a table. Vegemite [17] applies PBD to integrate web data, automates the use of web services, and generates shareable scripts. Other interfaces [14] apply PBD to data integration via copy and paste actions.

Wrangler applies a number of these techniques. Wrangler infers regular expressions from example text selections [2] and provides support for mass editing [13, 18]. Wrangler uses semantic roles akin to Topes [23] and provides natural language descriptions in interaction scripts [17]. However, Wrangler differs in important ways. Most PBD tools for data support either text extraction or data integration, but lack operations such as reshaping, aggregation, and missing value imputation. In the case of ambiguous input, these tools either offer no recourse or require users to specify more examples. With the exception of Vegemite [17], prior tools do not generate editable scripts to document provenance.

Most closely related to Wrangler is prior work on interactive data cleaning tools. Potter's Wheel [21] provides a transformation language for data formatting and outlier detection. Wrangler's transformation language extends that of Potter's Wheel, with some key differences discussed later. Ajax [8] also provides a graphical interface to specify transforms for reformatting and entity resolution. Both tools provide limited support for direct manipulation: interaction is largely restricted to menu-based commands or entering programming statements. GridWorks [12] leverages Freebase to enable entity resolution and discrepancy detection. It provides summarization and filtering support through faceted histograms. Though users can specify some commands graphically, others must be written in a command language. Moreover, the

system assumes that input data arrives in a proper tabular format, limiting the forms of data to which it can be applied.

Wrangler builds on this prior work to contribute novel techniques for specifying data transforms. Wrangler's support of programming-by-demonstration techniques is enabled by a more general inference engine that generates and rank-orders suggested transformations in response to user interactions. Analysts can navigate the space of transforms by directly selecting data, indicating a desired transform via menus, and modifying a related transform; each of these actions leads Wrangler to further refine the set of suggestions. To help analysts understand the effects of an operation before they commit to it, Wrangler's natural language transform descriptions are augmented by a novel preview mechanism that visualizes transform results. In concert, these techniques help analysts hone in on a desired transformation.

USAGE SCENARIO

Consider an example wrangling task, using housing crime data from the U.S. Bureau of Labor Statistics. The data were downloaded as a CSV (comma-separated values) file, but are not immediately usable by other tools: the data contains empty lines, U.S. states are organized in disjoint matrices, and the state names are embedded in other text. We describe how an analyst can use Wrangler to transform the data into more usable formats (Figures 1–7).

The analyst begins by pasting the text of the file into the Wrangler input box (alternatively, the analyst could upload the file). The interface now shows a data table occupying most of the screen (Fig. 1). To the left of the data table is a panel containing an interactive transformation history and a transform editor. The history already contains three transforms, as Wrangler inferred that the data was in CSV format and so split the text into rows on newline characters, split the rows into columns on commas, and then promoted the first row to be the table header. Note that the analyst could undo any transform by clicking the red undo button to the right of the transform, or could edit the transform descriptions in place. In this case, the analyst has no need.

The analyst then begins wrangling the file into a usable form. First, she *ctrl-clicks* row headers for two empty rows (7 and 9) to select them; the transformation editor suggests possible operations in response (Fig. 1). The first suggestion is to delete just the selected rows. The analyst can navigate the suggestions using the *up* and *down* arrows on the keyboard, or by mousing over the description in the editor on the left. As she navigates the suggestions, Wrangler previews the effects of the transforms in the data table. For deletions, the preview highlights the candidate deleted rows in red (Fig. 2). The analyst mouses over the suggestion to delete all empty rows in the table and clicks the green *add* button to execute the transform. The system then adds the deletion operation to the transformation history.

The analyst would like to compare data across states, so she first needs to extract the state names and add them to each row of the data. She selects the text 'Alaska' in row 6 of the

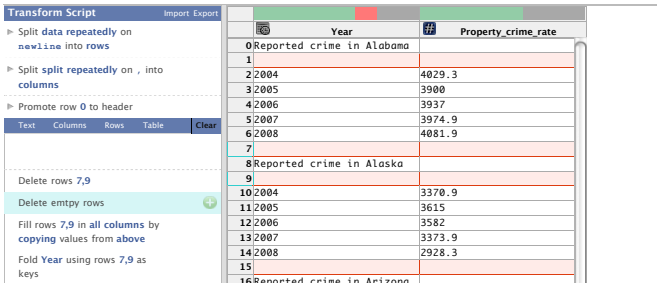

Figure 2. Row deletion. The analyst selects two empty rows and chooses a *delete* transform. Red highlights preview which rows will be deleted.

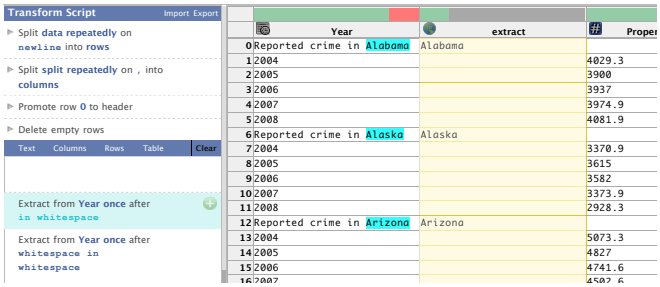

Figure 3. Text extraction. The analyst selects state names to *extract* them into a new column. Yellow highlights show a preview of the result.

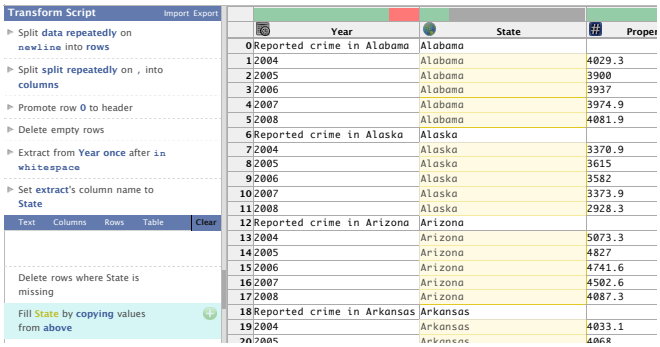

Figure 4. Filling missing values. The analyst populates empty cells by clicking the gray bar in the data quality meter above the "State" column, and then selecting a *fill* transform.

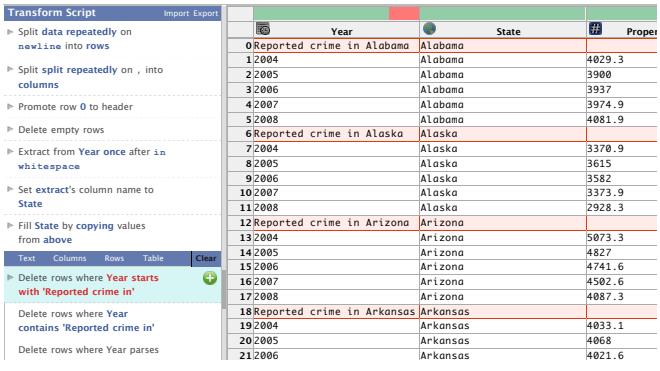

Figure 5. Deleting rows. The analyst selects text in unwanted rows and selects a *delete* operation within the "Rows" menu. Red highlighting previews which rows will be deleted.

"Year" column. Wrangler initially interprets this as selecting text at positions 18-24. The analyst updates Wrangler's inference by selecting 'Arizona' in the cell six rows below. Wrangler now suggests extracting text occurring after the

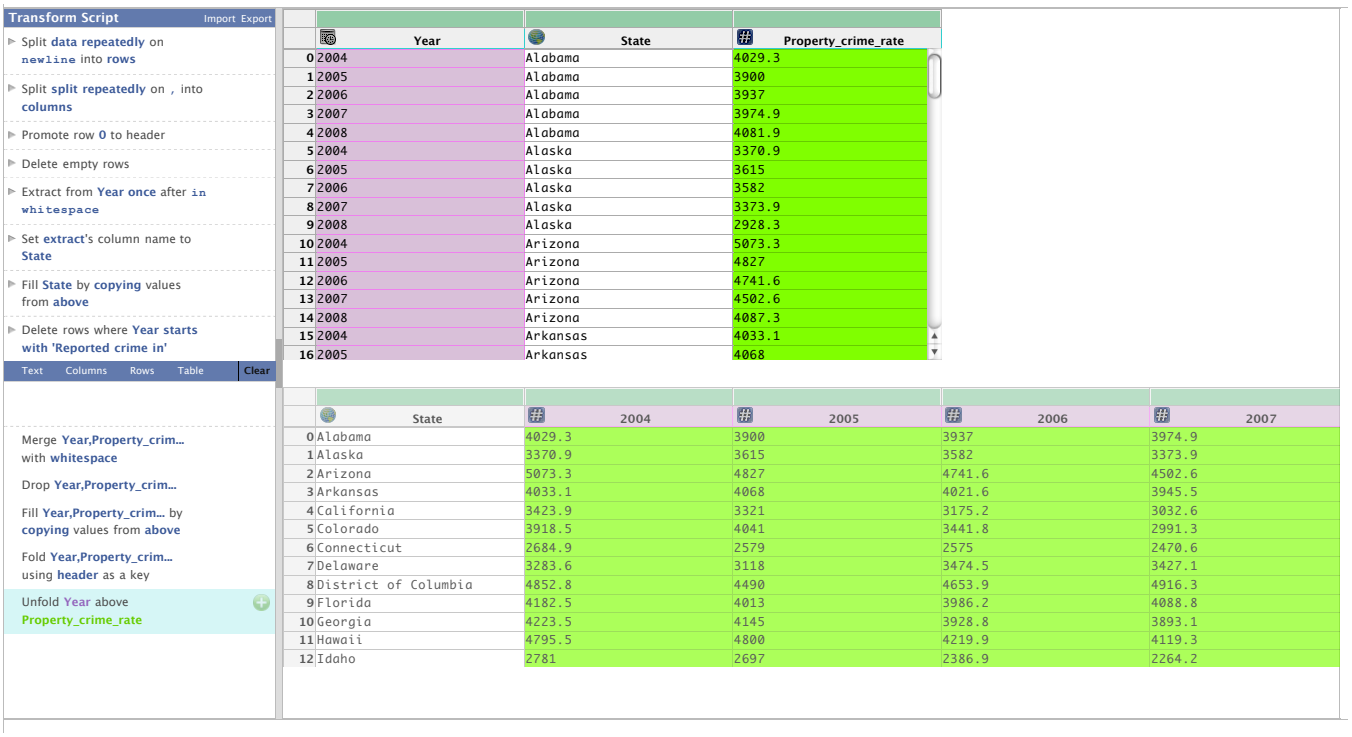

Figure 6. Table reshaping. The analyst selects two columns, and then elects to *unfold* them to create a cross-tabulation. A ghosted table overlay previews the result. Color highlights show the correspondence of data between the start and end states.

string "in " (Fig. 3). The analyst executes this transform and renames the resulting column "State". She notices that the column is sparsely populated. These missing values are indicated by the gray bar in the *data quality meter* above the column. The analyst clicks the gray bar and Wrangler suggests transforms for missing values. The analyst chooses to fill empty cells with the value from above (Fig. 4).

Looking at the "Year" column, the analyst notices a red bar in the data quality meter indicating inconsistent data types. Wrangler has inferred that the column primarily contains numbers, and so has flagged non-numeric values as potential errors. She decides to remove the rows containing the text 'Reported crime in'. She selects the text 'Reported crime in' in row 0. Wrangler suggests split, extract, and cut transforms, but no delete operations. In response, the analyst selects the *Delete if...* command from the *Rows* menu in the transform editor. This action reorders the suggestions so that *delete* commands have higher ranking. She finds the suggestion that deletes the unwanted rows containing the selected text (Fig. 5) and executes the transform.

At this point the analyst has wrangled the data into a proper relational format, sufficient for export to database and visualization tools. But now suppose she would like to create a cross-tabulation of crime rates by state and year for subsequent graphing in Excel. She selects the "Year" and "Property_crime_rate" columns, previews the suggested *unfold* operation (Fig. 6), then executes it to create the desired cross-tab. The *unfold* operation creates new columns for each unique value found in the "Year" column, and reorganizes the "Property_crime_rate" values by placing each in the appropriate cell in the resulting matrix.

```
split('data').on(NEWLINE).max_splits(NO_MAX)
split('split').on(COMMA).max_splits(NO_MAX)
columnName().row(0)
delete(isEmpty())
extract('Year').on(/.*/).after(/in /)
columnName('extract').to('State')
fill('State').method(COPY).direction(DOWN)
delete('Year starts with "Reported crime in"')
unfold('Year').above('Property_crime_rate')
```

Figure 7. The result of the analyst's data wrangling session is a declarative data cleaning script, shown here as generated JavaScript code.

The analyst's process results in a transformation script written in a declarative transformation language. The script provides an auditable description of the transformation enabling later inspection, reuse, and modification. The analyst can also annotate these transformations with her rationale. By clicking the *Export* button above the transformation history, the analyst can either save the transformed data or generate runnable code implementing the transformation (Figure 7).

DESIGN PROCESS

We based Wrangler on a transformation language with a handful of operators. Originally we thought that each of these operators might correspond to a single interaction with example data in a table view. However, after considering different mappings and evaluating their implications, we were unable to devise an intuitive and unambiguous mapping between simple gestures and the full expressiveness of the language. A given interaction could imply multiple transforms and multiple interactions might imply the same transform.

Although this many-to-many relationship between the language and interaction might complicate our interface, we found the relationship to be relatively sparse in practice: the number of likely transforms for a given gesture is small. As a result, we took a mixed-initiative approach. Instead of mapping an interaction to a single transform, we surface likely transforms as an ordered list of suggestions. We then focused on rapid means for users to navigate—prune, refine, and evaluate—these suggestions to find a desired transform.

Wrangler is a browser-based web application, written in JavaScript. In the next section we describe the Wrangler transformation language. We then present the Wrangler interface and its techniques for navigating suggestion space. Next, we describe Wrangler's mechanisms for verification. We go on to discuss the technical details of our suggestion inference.

THE WRANGLER TRANSFORMATION LANGUAGE

Underlying the Wrangler interface is a declarative data transformation language that extends the Potter's Wheel language [21], which in turn borrows from SchemaSQL [16]. Our language design process was guided by prior work [8, 21] and empirical data; we gathered data sets from varied sources (e.g., data.gov, NGOs, log files, web APIs) and used them to drive language requirements. Language statements manipulate *data tables*: collections of numbered rows and named columns of data. Wrangler treats raw text as a "degenerate" table containing one row and one column. The language consists of eight classes of transforms, described below.

Map transforms map one input data row to zero, one, or multiple output rows. *Delete* transforms (one-to-zero) accept predicates determining which rows to remove. One-to-one transforms include *extracting*, *cutting*, and *splitting* values into multiple columns, reformatting, simple arithmetic, and updates. One-to-many transforms include operations for splitting data into new rows, such as splitting a text file on newlines and unnesting arrays and sets.

Lookups and joins incorporate data from external tables. Wrangler includes extensible lookup tables to support common types of transformations, such as mapping zip codes to state names for aggregation across states. Currently Wrangler supports two types of joins: equi-joins and approximate joins using string edit distance. These joins are useful for lookups and correcting typos for known data types.

Reshape transforms manipulate table structure and schema. *Fold* transforms collapse multiple columns to two or more columns containing key-value sets. *Unfold* transforms create new column headers from data values. See [21] for an extended discussion. Reshaping transforms are common in tools such as R and Excel Pivot Tables, and are necessary for higher-order restructuring of data.

Positional transforms include *Fill* and *Lag* operations. *Fill* operations generate values based on neighboring values in a row or column and so depend on the sort order of the table. For example, an analyst might fill empty cells with preceding non-empty values. The *Lag* operator shifts the values of a column up or down by a specified number of rows.

The language also includes functions for **sorting**, **aggregation** (e.g., sum, min, max, mean, standard deviation), and **key generation** (a.k.a., *skolemization*). Finally, the language contains **schema** transforms to set column names, specify column data types, and assign semantic roles.

To aid data validation and transformation, Wrangler provides *data types* (e.g., integers, numbers, dates, strings) and higher-level *semantic roles* (e.g., geographic location, classification codes, currencies). Data types comprise standard primitives and associated parsing functions. Semantic roles consist of additional functions for parsing and formatting values, plus zero or more transformation functions that map between related roles. As an example, consider a semantic role defining a *zip code*. The zip code role can check that a zip code parses correctly (i.e., is a 5 digit number) and that it is a valid zip code (checking against an external dictionary of known zipcodes). The zip code role can also register mapping functions, e.g., to return the containing state or a central lat-lon coordinate. Wrangler leverages types and roles for parsing, validation, and transform suggestion. The Wrangler semantic role system is extensible, but currently supports a limited set of common roles such as geographic locations, government codes, currencies, and dates.

The design of the Wrangler language co-evolved with the Wrangler interface described in subsequent sections. We desired a consistent mapping between transforms presented in the interface and statements in the language. Disconnects between the two might cause confusion [19], especially when analysts try to interpret code-generated scripts. As a result, we chose to introduce some redundancy in the language. Some operations, particularly *positional* transforms, are equivalent to a series of lower-level transforms (i.e., using *key* transforms, self-joins, and scalar functions). We present such high-level operations in the interface as a single transform if they are commonly invoked and their lower-level realization is unintuitive. We then add corresponding high-level operators to the language. The result is a clear one-to-one mapping between transforms presented in the interface and statements in output scripts.

Prior work [16, 21] shows that our basic set of transforms is sufficient to handle all one-to-one and one-to-many transforms. Through both our own practice and discussions with analysts, we believe our extended language is sufficient to handle a large variety of data wrangling tasks.

THE WRANGLER INTERFACE DESIGN

The goal of the Wrangler interface is to enable data analysts to author expressive transformations with minimal difficulty and tedium. To this aim, our interface combines direct manipulation, menu-based command selection, automatic suggestion, and transform refinement. This synthesis of techniques enables analysts to navigate the space of applicable transforms using whatever means they find most convenient.

Both novices and experts can find it difficult to specify transform parameters such as regular expressions. While direct manipulation selections can help, inference is required to

suggest transforms without programming. To reduce the gulf of execution [19], Wrangler uses an *inference engine* that suggests data transformations based on user input, data type or semantic role, and a number of empirically-derived heuristics. These suggestions are intended to facilitate the discovery and application of more complicated transforms.

However, suggested transforms (and their consequences) may be difficult to understand. To reduce the gulf of evaluation [19], Wrangler provides *natural language descriptions* and *visual transform previews*. Natural language descriptions are intended to enhance analysts' ability to review and refine transformation steps. Lightweight annotation enables communication of analyst intent. Wrangler also couples verification (run in the background as data is transformed) with visualization to help users discover data quality issues.

Figure 8. Editable Natural Language Descriptions. (a) An example of an editable description; highlighted text indicates editable parameters. (b) Clicking on a parameter reveals an in-place editor. (c) After editing, the description may update to include new parameters. In this case, a new window size parameter is displayed for the moving average.

Basic Interactions

The Wrangler interface supports six basic interactions within the data table. Users can select rows, select columns, click bars in the data quality meter, select text within a cell, edit data values within the table (for mass editing [13, 18]), and assign column names, data types or semantic roles. Users can also choose transforms from the menu or refine suggestions by editing transform descriptions as described below.

Automated Transformation Suggestions

As a user interacts with data, Wrangler generates a context-sensitive list of suggested transforms. Transforms are represented by editable natural language descriptions. Users can cycle through suggestions using the up and down keys on the keyboard or by mousing over a specific suggestion. When a suggestion is highlighted, Wrangler previews the effect of the suggestion in a visualization (discussed later). By cycling through suggestions and viewing their previews, users can identify the desired transform.

Navigating the space of suggestions efficiently involves a number of considerations. In some cases the set of possible suggestions is large (e.g., in the hundreds), but we wish to show only a relevant handful in the interface to avoid overload. Instead of enumerating the entire suggestion space, users can prune and reorder the space in three ways. First, users can provide more examples to disambiguate input to the inference engine. Providing examples is especially effective for text selections needed for splitting, extraction, and reformatting; two or three well-chosen examples typically suffice. Second, users can filter the space of transforms by selecting an operator from the transform menu. Third, users can edit a transform by altering the parameters of a transform to a desired state. Wrangler does not immediately execute selected suggestions. Instead, Wrangler promotes the suggestion to be the *current working transform*. The user can edit this transform directly; as a user edits parameters, the suggestion space updates to reflect these edits. A user can also continue interacting with the table to generate new suggestions that use the working transform as context.

Natural Language Descriptions

To aid apprehension and refinement of suggested transforms, Wrangler generates short natural language descriptions of the transform type and parameters. These descriptions are editable, with parameters presented as bold hyperlinks (Fig. 8). Clicking a link reveals an in-place editor appropriate to the parameter (Fig. 8b). Enumerable variables (such as the direction of a fill) are mapped to selection widgets (e.g., drop-down menus) while free-form text parameters are mapped to text editors with autocomplete.

We designed these descriptions to be concise; default parameters that are not critical to understanding may be omitted. For example, the *unless between* parameter for split operations indicates regions of text to ignore while splitting. In most cases, this parameter is left undefined and including it would bloat the description. To edit hidden parameters users can click the expansion arrow to the left of the description, revealing an editor with entries for all parameters.

We also sought to make parameters within descriptions readable by non-experts. For instance, we translate regular expressions into natural language via pattern substitution (e.g., (\d+) to 'number' and ^ to 'leading'). This translation can make some descriptions less concise but increases readability. Translation is only performed with regular expressions generated by the Wrangler inference engine. If a user types in a custom expression, Wrangler will reflect their input.

To generate descriptions, Wrangler assigns each transform parameter five functions, controlling the type, prefix, suffix, visibility, and description. The type function indicates which type of in-place editor to use. The prefix and suffix functions generate surrounding text for a parameter and take as input both the value of the parameter and all other parameters for the given transform. The visibility function determines whether or not the description includes a reference to the parameter at all; the return value may depend on the value of the parameter and all other parameters. The description function generates a readable representation for the parameter. This function is most useful for translating regular expressions and simplifying complex selection clauses (e.g., "A=null and B=null and C=null" to "the row is empty"). Wrangler strives for syntactic consistency across descriptions, e.g., the transform name is always the first word and the operand columns appear after the transform name.

Visual Transformation Previews

Wrangler uses *visual previews* to enable users to quickly evaluate the effect of a transform. For most transforms, Wrangler displays these previews in the source data, and not as a separate visualization (e.g., a side-by-side before and after preview). In-place previews provide a visual economy that serves a number of goals. First, displaying two versions of a table inherently forces both versions to be small, which is particularly frustrating when the differences are sparse. Second, presenting in-place modifications draws user attention to the effect of the transformation in its original context, without requiring a shift in focus across multiple tables. As we discuss next, in-place previews better afford direct manipulation for users to revise the current transform.

Wrangler maps transform types to at least one of five preview classes: selection, deletion, update, column and table. In defining these mappings, we attempted to convey a transform's effect with minimum displacement of the original data. This stability allows users to continue interacting with the original data, e.g., to provide new selection examples.

Selection previews highlight relevant regions of text in all affected cells (Fig. 3). *Deletion* previews color to-be-deleted cells in red (Fig. 2). *Update* previews overwrite values in a column and indicate differences with yellow highlights (Fig. 4). *Column* previews display new derived columns, e.g., as results from an *extract* operation (Fig. 3). The one case for which we show a side-by side display of versions is when previewing transformations that alter the layout of data (*fold* and *unfold*). These transforms alter the structure of the table to such an extent that the best preview is to show another table (Fig. 6, 9). Such *Table* previews use color highlights to match input data to their new locations in the output table. Some transforms map to multiple classes; for instance, an *extract* command uses both selection and column previews.

When possible, previews also indicate where the user can modify the transform through either direct manipulation or description refinement. Simply highlighting selected text or cells works well for certain transformations. For example, by highlighting the text selected by a regular expression for each cell, users can determine which examples they need to fix to update the transform. For reshape transformations, Wrangler highlights the input data in the same color as the corresponding output in the secondary table. For instance, in a *fold* operation, data values that will become keys are colored to match the keys in the output table (Fig. 9). Wrangler also highlights the parameters in the transform description using the same colors as those generated in previews (Fig. 3–6). The consistent use of colors allows users to associate clauses in a description with their effects in the table.

Transformation Histories and Export

As successive transforms are applied, Wrangler adds their descriptions to an interactive *transformation history viewer*. Users can edit individual transform descriptions and selectively enable and disable prior transforms. Upon changes, Wrangler runs the edited script and updates the data table. Toggling or editing a transform may result in downstream errors; Wrangler highlights broken transforms in red and provides an error message to aid debugging.

split_1	split_2	split_3	split_4	split_5
	2004	2004	2004	2003
STATE	Participation Rate 2004	Mean SAT I Verbal	Mean SAT I Math	Participation Rate 2003
New York	87	497	510	82
Connecticut	85	515	515	84
Massachusetts	85	518	523	82
New Jersey	83	501	514	85
New Hampshire	80	522	521	75
D.C.	77	489	476	77
Maine	76	505	501	70
Pennsylvania	74	501	502	73
Delaware	73	500	499	73
Georgia	73	494	493	66
Rhode Island	72	503	502	74
Virginia	71	515	509	71
North Carolina	70	499	507	68
Maryland	68	511	515	68
Florida	67	499	499	61
Vermont	66	516	512	70

split_1	fold	fold_1	fold_2
New York	2004	Participation Rate 2004	87
New York	2004	Mean SAT I Verbal	497
New York	2004	Mean SAT I Math	510
New York	2003	Participation Rate 2003	82
New York	2003	Mean SAT I Verbal	496
New York	2003	Mean SAT I Math	510
Connecticut	2004	Participation Rate 2004	85
Connecticut	2004	Mean SAT I Verbal	515
Connecticut	2004	Mean SAT I Math	515
Connecticut	2003	Participation Rate 2003	84
Connecticut	2003	Mean SAT I Verbal	512
Connecticut	2003	Mean SAT I Math	514
Massachusetts	2004	Participation Rate 2004	85
Massachusetts	2004	Mean SAT I Verbal	518
Massachusetts	2004	Mean SAT I Math	523

Figure 9. Visual preview of a *fold* operation. For transforms that rearrange table layout, Wrangler previews the output table and uses color highlights to show the correspondence of values across table states.

Wrangler scripts support lightweight text annotations. Analysts can use annotations to document their rationale for a particular transform and may help future users better understand data provenance. To annotate a transform, users can click the *edit* icon next to the desired transform and write their annotation in the resulting text editor. Users can view an annotation by mousing over the same *edit* icon. These annotations appear as comments in code-generated scripts. Users can export both generated scripts and transformed data; clicking the *Export* button in the transform history invokes export options. Analysts can later run saved or exported scripts on new data sources, modifying the script as needed.

TYPES, ROLES, AND VERIFICATION

It is often difficult to discover data quality issues and therefore difficult to address them by constructing the appropriate transform. Wrangler aids discovery of data quality issues through the use of data types and semantic roles.

As users transform data, Wrangler attempts to infer the data type and semantic role for each column. Wrangler applies validation functions to a sample of the user's data to infer these types. Wrangler assigns a column the data type that validates for over half of the non-missing values. If multiple data types satisfy this criteria, Wrangler assigns the more specific data type (e.g., *integer* is more specific than *double*). Wrangler infers semantic roles analogously. An icon in the column header indicates the semantic role of the column, or the underlying data type if no role has been assigned. Clicking on the icon reveals a pop-up menu with which users can manually assign a type or role.

Atop each column resides a *data quality meter* that provides a validation summary. A divided bar graph indicates the proportion of values in the column that verify completely. Values that parse successfully are indicated in green; values that match the type but do not match the role (e.g., a 6 digit zip code) are shown in yellow; those that do not match the type (e.g., 'One' does not parse as an integer) are shown in red;

and missing data are shown in gray. Clicking a bar generates suggested transforms for that category. For instance, clicking the *missing values* bar will suggest transforms to fill in missing values or delete those rows. Clicking the *fails role* bar will suggest transforms such as a similarity-join on misspelled country names.

THE WRANGLER INFERENCE ENGINE

The Wrangler inference engine is responsible for generating the ranked list of suggested transforms. The engine models transforms in the language as points in a multidimensional feature space. Inputs to the engine consist of user interactions; the current working transformation; and data descriptions such as column data types, semantic roles, and summary statistics. In this section, we discuss the criteria the inference engine uses to rank transforms by relevance.

The engine attempts to infer both the type of transform (e.g., *split* or *fill*) and the operands to this transform. Wrangler infers sets of parameters from user selections and edits, which Wrangler associates with one or more operand types. For example, selecting text in a cell triggers inference of text selection criteria, column selection, and row selection. Wrangler infers values for these operands independent of all other parameters, e.g., it infers an ordered list of regular expressions for text selection based solely on the selected text, a process otherwise independent of which rows are selected.

To infer text selections, Wrangler generates a set of selections (including regular expressions) that match the examples, ranked by decreasing description length. To rank other parameters, such as column selections and enumerable parameters (low-cardinality parameters with a pre-defined set of alternatives), we order alternatives by their frequency of occurrence in a collection of data wrangling scripts created using our test corpus of data sets.

After inferring potential parameters, Wrangler enumerates transforms that accept as input at least one of the inferred parameters. Continuing the text selection example from above, *split*, *cut*, and *extract* transforms all accept each of the inferred parameters (e.g., split column *name* on *','* for *all rows*), whereas *delete* transforms accept only row selection criteria. Wrangler will instantiate suggested transforms using the cross product of the inferred parameters, leaving all other parameters at their default values. Wrangler then filters these transforms based on the data description. For example, Wrangler will not suggest unfolding numeric columns with high cardinality, as it is unlikely a user wants to create a wide table with columns named as numbers. Wrangler also filters the suggestions based on explicit actions given by the user: if a user chooses a transform from the menu or has selected a current working transform, Wrangler will assign higher rank to similar transforms.

Wrangler also considers specification difficulty and transform diversity. Some parameters are harder than others to specify, such as regular expressions and row selection clauses. Wrangler preferentially ranks transforms that differ along those parameters. In contrast, users can easily edit other parameters such as the direction of a fill or the text to use as glue in a merge. Based on hand-coded rules, Wrangler tags transform parameters as *hard* or *easy* to specify and subsequently ranks transforms with hard parameters above those with easy parameters. Also, Wrangler provides examples of diverse types of transforms to enable discovery. Therefore Wrangler will lower the ranking of transforms that have the same transform type as ones already suggested.

Figure 10. Suggested Transforms. (a) Sample suggestions generated after the user selects the text 'Alaska'. The suggestions range across parameters (e.g., regular expressions) and across transformation types (*split*, *extract*, *cut*). (b) The suggestions update after the user provides another selection example: 'Arizona'.

We hand-coded our inference rules in an intuitive and pragmatic fashion. In the future, our mapping of transforms to an abstract feature space will allow us to experiment with standard machine learning approaches for inference.

COMPARATIVE EVALUATION WITH EXCEL

We conducted a user study comparing Wrangler to Microsoft Excel, a popular spreadsheet application. We chose Excel because it is by far the most popular tool for manipulating data, while most of the PBD tools discussed earlier have limited use. Excel also provides an expressiveness of transforms comparable to Wrangler, though often by quite different means. The goals of the study were to test the usability of the Wrangler interface, compare completion time and accuracy across tools, and observe data cleaning strategies.

Participants and Methods

We recruited 12 participants, all graduate students or professionals who work with data in some capacity. Subjects rated their prior experience with Excel on a 10-point scale (1 being basic knowledge and 10 being expert); the median score was 5. Participants had never used the Wrangler interface.

We presented a 10 minute tutorial on using Wrangler to each subject, describing Wrangler's interface and how to create, edit, and execute transforms. After the tutorial, we asked subjects to complete three tasks using both Wrangler and Excel. We randomized the presentation of tasks and tools across subjects. In each task, we asked subjects to transform a data set into a new format, presented to them as a picture.

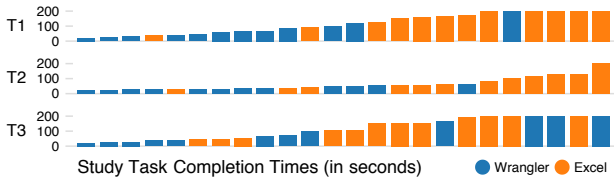

Figure 11. Task completion times, sorted and capped at 200 seconds. Median Wrangler performance is at least twice as fast in all tasks.

Each task focused on a transform type: text extraction, missing value imputation, or reshaping table structure.

Task 1: Extract Text. In this task, we asked users to extract the number of bedrooms and housing price from housing listings on craigslist. The original data set contained one cell for each listing, with all the information in a text string. The target data set consisted of two columns: one for the number of bedrooms and one for the housing price.

Task 2: Fill Missing Values. We gave users data containing year-by-year agricultural data for three countries. Some of the values in the data set were blank. The target data set contained the same data with all missing values replaced with the closest non-empty value from a previous year.[1]

Task 3: Reshape Table Structure. Users started with three columns of housing data: year, month, and price. The target data set contained the same data formatted as a cross-tab: the data contained one row for each year, with the 12 months as column headers and housing prices as cell values.

When using Excel, we allowed subjects to ask for references to functions they could describe concretely (e.g., we would answer *"how do I split a cell?"* but not *"how do I get the number of bedrooms out?"*). For Wrangler tasks, we did not respond to user inquiries. We permitted a maximum of 10 minutes per task. Each data set had at most 30 rows and 4 columns; complete manual manipulation in Excel was attainable within the time limits. Afterwards, each user completed a post-study questionnaire.

Wrangler Accelerates Transform Specification

Across all tasks, median performance in Wrangler is at least twice as fast as Excel (Fig. 11). Applying non-parametric Mann-Whitney U tests, we found that Wrangler significantly outperformed Excel in both Task 1 ($U(12)$=17.5, $p<0.002$) and Task 2 ($U(12)$=18.5, $p<0.003$). For Task 3 we found no significant difference between completion times ($U(12)$=51, p=0.236). However, we noticed that subjects who reported little experience with Excel (and presumably, less data analysis experience) performed poorly in Task 3 using both tools. Restricting our attention to the 8 users who reported an experience rating of 4 or higher, we found Wrangler again significantly outperforms Excel ($U(8)$=9, $p<0.02$).

These results show that not only can novice Wrangler users specify data transformations more quickly, those who are already skilled Excel users benefit even more. Furthermore, the user study tasks involved small data sets amenable to manual manipulation; as data set size grows, we expect the benefits of Wrangler to come into even sharper relief. Of course, larger data sets also complicate the process of assessing a transform's effect and might benefit from additional validation and visualization techniques.

Strategies for Navigating Suggestion Space

When using Wrangler, subjects applied different navigation strategies for different tasks. These strategies were largely consistent across users. For operations with complex text selection criteria, users most frequently provided multiple examples to prune the space of transforms. For all other operations, they would usually perform one selection or edit and then cycle through the presented suggestions.

If users did not find a transform immediately, their most common recourse was to filter suggestions by selecting a transform type from the menu. If only imperfect matches were found, users would select the nearest transform and edit its parameters. However, users did this only when other methods of navigation had failed.

These navigation strategies worked well when users understood the nature of the desired transform, even if they did not know how to specify it. However, we found that users of both tools experienced difficulty when they lacked a conceptual model of the transform. For instance, Task 3 exhibited an uneven distribution of completion times; 7 of the 10 fastest times and 3 of the 4 slowest times were in Wrangler. Wrangler does not provide the recourse of manual editing, hence users who got stuck fared slightly better in Excel. However, those familiar with pivot operations in Excel uniformly performed the task more quickly with Wrangler.

We also observed one recurring pitfall: a few users got stuck in a "cul-de-sac" of suggestion space by incorrectly filtering (e.g., by selecting a specific transform type from the menu). When this happened, some users kept searching and refining only these filtered transforms. This pitfall was most common in Task 3, where users might mistakenly filter all but *fold* operations when an *unfold* operation was needed. One way to alleviate this pitfall may be to suggest some non-matching transforms related to the one selected, in effect treating filtering criteria as guidelines rather than strict rules.

Previewing Transform Effects

Subjects reported that the previews were more useful than transform descriptions for evaluating transforms. One subject noted, *"I just look at the picture."* Users with more programming experience spent time reading the descriptions, whereas users with limited programming experience relied almost entirely on the previews. Relying too heavily on previews introduces reusability issues, as the preview does not capture a transform's effect over new data. For instance, multiple extraction transforms will extract the same text given a certain sample of data. Without reading and understanding the description, users may not understand how this transform would perform on data not currently present in the table.

[1] We acknowledge that this is not an ideal cleaning solution for this data, but it served as a useful test nonetheless.

User Sentiment

Anecdotally, users tended to enjoy working with Wrangler. Three users asked *"Can I have this?"* or *"When can I use this?"*. In comparison to Excel and other packages, one subject said *"It's so much less frustrating. It is like soooo much less frustrating."* Another participant sent us a message three days after the experiment: *"i was thinking more about wrangler. it is pretty cool."* The most commonly reported advantages of using Wrangler were the ability to preview data and use direct manipulation to specify transforms. One user responded, *"It's super easy to see whats going to happen."* Another reported, *"you can select stuff and it works."*

The most commonly reported advantages of Excel were the ability to create graphs and to edit data layout manually. Charting is not supported in Wrangler, but visualization is an important component of data cleaning we plan to address more deeply in future work. By design, Wrangler does not afford users the same flexibility to layout data as in Excel. This impediment relates to the pitfall described above; since users cannot perform arbitrary editing in Wrangler, the recourse is less obvious when they get stuck.

CONCLUSION AND FUTURE WORK

This paper introduced Wrangler, an interface and underlying language for data transformation. The system provides a mixed-initiative interface that maps a set of user interactions to a space of suggested transforms, combined with natural language transform descriptions and visual previews to help assess each suggestion. With this set of techniques, we find that users can rapidly navigate to a desired transform.

Our user study demonstrates that novice Wrangler users can perform data cleaning tasks significantly faster than in Excel, an effect even more pronounced for skilled Excel users. We found that users are comfortable switching navigation strategies in Wrangler to suit a specific task, but can sometimes get stuck—in either tool—if they are unfamiliar with the available transforms. Future work should help users form data cleaning strategies, perhaps through improved tutorials.

We plan to release Wrangler as a public web application. Deployment to a larger audience will allow us to further study wrangling strategies and will provide usage data for further improving our inference model. To enable use by a wide audience, future work will include scaling Wrangler to handle larger data sets. Wrangler is currently implemented purely as a client-side application; we will extend it to include a scalable server component. We also plan to introduce more summary visualizations, coupled with outlier detection methods, to aid data diagnostics. More study is needed to assess how visualizations can best support data profiling.

Looking forward, Wrangler addresses only a subset of the hurdles faced by data analysts. As data processing has become more sophisticated, there has been little progress on improving the tedious parts of the pipeline: data entry, data (re)formatting, data cleaning, etc. The result is that people with highly specialized skills (e.g., statistics, molecular biology, micro-economics) spend more time in tedious "wrangling" tasks than they do in exercising their specialty, while less technical audiences such as journalists are unnecessarily shut out. We believe that more research integrating methods from HCI, visualization, databases, and statistics can play a vital role in making data more accessible and informative.

REFERENCES

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD*, pages 337–348, 2003.
2. A. F. Blackwell. SWYN: A visual representation for regular expressions. In *Your Wish is my Command: Programming by Example*, pages 245–270, 2001.
3. L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive generation of integrated schemas. In *ACM SIGMOD*, pages 833–846, 2008.
4. T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, 2003.
5. T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *ACM SIGMOD*, pages 240–251, 2002.
6. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.
7. K. Fisher and R. Gruber. Pads: a domain-specific language for processing ad hoc data. In *ACM PLDI*, pages 295–304, 2005.
8. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: an extensible data cleaning tool. In *ACM SIGMOD*, page 590, 2000.
9. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD*, pages 805–810, 2005.
10. J. M. Hellerstein. Quantitative data cleaning for large databases, 2008. White Paper, United Nations Economic Commission for Europe.
11. V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.
12. D. Huynh and S. Mazzocchi. Freebase GridWorks. `http://code.google.com/p/freebase-gridworks/`.
13. D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: semi-ontology alignment for casual users. In *ISWC*, pages 903–910, 2007.
14. Z. G. Ives, C. A. Knoblock, S. Minton, M. Jacob, P. Pratim, T. R. Tuchinda, J. Luis, A. Maria, and M. C. Gazen. Interactive data integration through smart copy & paste. In *CIDR*, 2009.
15. H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE TVCG*, 14(5):999–1014, 2008.
16. L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Trans. Database Syst.*, 26(4):476–519, 2001.
17. J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *IUI*, pages 97–106, 2009.
18. R. C. Miller and B. A. Myers. Interactive simultaneous editing of multiple text regions. In *USENIX Tech. Conf.*, pages 161–174, 2001.
19. D. A. Norman. *The Design of Everyday Things*. Basic Books, 2002.
20. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.
21. V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.
22. G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *ACM CHI*, pages 431–439, 2005.
23. C. Scaffidi, B. Myers, and M. Shaw. Intelligently creating and recommending reusable reformatting rules. In *ACM IUI*, pages 297–306, 2009.
24. S. Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, 1999.
25. R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. In *ACM IUI*, pages 139–148, 2008.