Fourth Workshop on End-User Software Engineering (WEUSE IV), at the International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 2008, pp. 11-15.

Using Topes to Validate and Reformat Data in End-User Programming Tools

Chris Scaffidi[1], Allen Cypher[2], Sebastian Elbaum[3], Andhy Koesnandar[3], James Lin[2], Brad Myers[1], Mary Shaw[1]

[1] School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
{cscaffid, bam, mary.shaw}
@cs.cmu.edu

[2] Almaden Research Center
IBM
Almaden, CA
{acypher, jameslin}
@us.ibm.com

[3] Computer Science and
Engineering Department
University of Nebraska-Lincoln
{elbaum, akoesnan}
@cs.unl.edu

ABSTRACT

End-user programming tools offer no data types except "string" for many categories of data, such as person names and street addresses. Consequently, these tools cannot automatically validate or reformat these data. To address this problem, we have developed a user-extensible model for string-like data. Each "tope" in this model is a user-defined abstraction that guides the interpretation of strings as a particular kind of data. Specifically, each tope implementation contains software functions for recognizing and reformatting instances of that tope's kind of data. This makes it possible at runtime to distinguish between invalid data, valid data, and questionable data that could be valid or invalid. Once identified, questionable and/or invalid data can be double-checked and possibly corrected, thereby increasing the overall reliability of the data. Valid data can be automatically reformatted to any of the formats appropriate for that kind of data. To show the general applicability of topes, we describe new features that topes have enabled us to provide in four tools.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming environments – *Interactive environments*

General Terms

Reliability, Languages.

1. INTRODUCTION

To understand the software needs of information workers, we conducted a contextual inquiry and other observations of workers [14]. We found that their tasks, such as filling out expense reports and creating employee rosters, often involve categories of short, human-readable, multi-format data. Example kinds of data include phone numbers, state names, and project codes. In many tasks, workers copied and pasted values among web pages, web forms and spreadsheets, often with intervening reformatting. Sometimes, users came upon data with questionable validity—that is, strings that were not obviously valid or obviously invalid—prompting them to double-check values (which they occasionally used anyway).

It would be difficult to automate many such tasks with existing tools for end-user programmers (EUPs), such as web macro tools and spreadsheets, since these tools support only basic primitives such as strings, integers and floating-point numbers. Because these tools do not recognize the kinds of data involved in workers' tasks, they cannot automatically reformat values or identify questionable values. For example, to date, web macro tools have been unable to copy a person name in "Firstname Lastname" format from one web page, then paste it into a web form in "Lastname, Firstname" format [14]. Moreover, these tools could not recognize and alert the user to a questionable person name with an odd mix of uppercase and lowercase letters, such as "Lincolnshire MCC".

The mechanisms currently available for extending tools with new abstractions—regular expressions (regexps) and scripting languages—are inflexible and hard for many people to use [1]. Mechanisms offered by researchers, such as Lapis [7] and data detectors [8], can recognize various data patterns and automate browser operations, but cannot reformat data automatically. Another approach, modeling data as types, would not work well because type-checking algorithms rely on the fact that a value either is or is not a valid instance of a type [11]. That is, strict type systems (even those supporting dependent types and other type refinements [9]) do not allow variables to contain data with questionable validity, as in the case of the data involved in these tasks.

In this paper, we describe a mechanism for extending tools as needed to support custom categories of short string data. Our mechanism relies on a kind of abstraction called "topes". Each tope abstraction defined by an end-user programmer includes functions to detect questionable values for one kind of data and functions to transform data among formats used for that kind of data. Tope implementations can be reused without modification in a variety of tools to validate and reformat data. Section 2 introduces topes, and Section 3 describes new features that topes have enabled us to provide in four tools. Section 4 concludes with a discussion of future work.

2. TOPES OVERVIEW

Our approach models each kind of data as an abstraction called a "tope", which contains functions for recognizing and transforming one kind of data [18]. For example, a tope might recognize an email address by looking for a username, followed by an @ symbol and a hostname. At the simplest level, the username and hostname could contain alphanumeric characters, periods, underscores, and certain other characters. Additional constraints would help to identify questionable strings (that are not definitely valid or definitely invalid). For example, an email address with 64 characters in the username would technically be valid but probably questionable.

Multiple patterns are necessary for describing kinds of data that may appear in more than one format. Each format would be recognized with a different function. For example, companies can be referenced

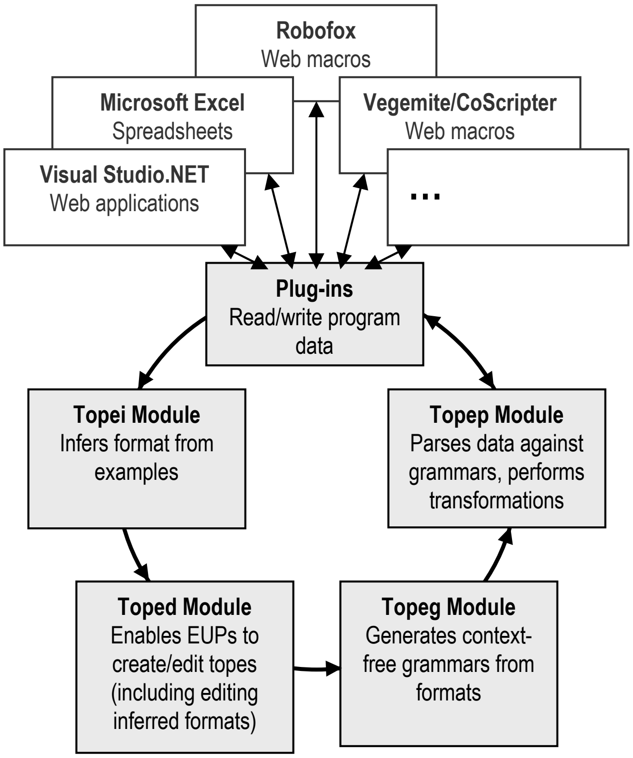
WEUSE IV'08, May 12, 2008, Leipzig, Germany.
Copyright 2008 ACM 978-1-60558-034-0/08/05...$5.00.

by common name, formal name, or stock ticker symbol. Common names are typically one to three words, sometimes containing apostrophes, ampersands, or hyphens. Formal names may be somewhat longer, though rarely more than 100 characters, and they sometimes contain periods, commas, and certain other characters. Ticker symbols are drawn from a finite set of officially registered symbols. These three formats *together* comprise a tope describing how to validate company names. The tope would include functions driven by lookup tables for transforming among these three formats.

Implementing topes

Just as an abstract type is not executable, topes are not directly executable but must be implemented. With the Tope Development Environment (TDE) that we have provided, EUPs perform two steps to implement each of a tope's formats. First, they provide one or more examples of the data to validate. The TDE infers a basic format covering most or all of the examples [13], and it presents this format on-screen (Figure 1). Second, the EUPs review, customize, and test the format in Toped [17], which is a form-based syntax-directed editor (Figure 2).

Figure 1: The TDE (shaded) receives data from plug-ins to programming tools (e.g.: Excel toolbar). From example strings, Topei infers a format that the programmer can customize in Toped, perhaps by adding additional formats or transformations between formats, yielding a tope implementation. This implementation can be saved to disk (not shown below) for reuse in many programs. After Topeg generates grammars from formats, Topep validates data provided by the plug-ins, perhaps yielding error messages that the plug-ins display in the spreadsheet, web application, web macro, or other program. Topep is also responsible for stepping through transformations at runtime to reformat strings. All shaded boxes are also accessible through an API. The only modules with a user interface are Toped (Figure 2) and the Plug-ins (Figures 3-5).

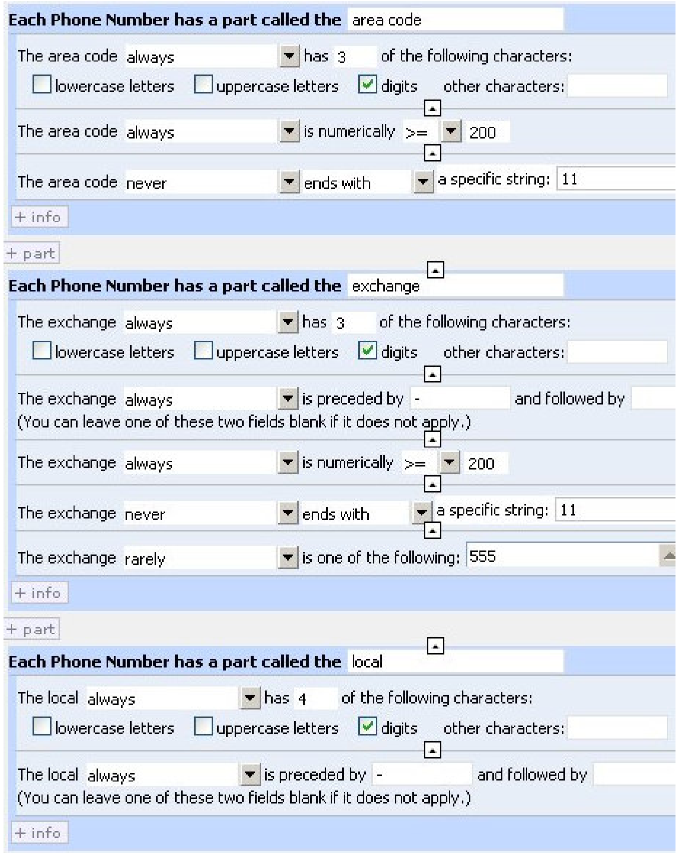

Robofox
Web macros
Microsoft Excel
Spreadsheets
Vegemite/CoScripter
Web macros
Visual Studio.NET
Web applications
...
Plug-ins
Read/write program data
Topei Module
Infers format from examples
Topep Module
Parses data against grammars, performs transformations
Toped Module
Enables EUPs to create/edit topes (including editing inferred formats)
Topeg Module
Generates context-free grammars from formats

To specify how to identify questionable inputs, EUPs can create "soft" constraints that are not always satisfied. For example, EUPs could specify that a phone number's exchange rarely is "555" (a value that was invalid for many decades). Other supported constraints include requiring that a part matches another format, or specifying that the part's value should be in a certain finite set.

From the constraints, Topeg generates a context-free grammar with constraints on the grammar productions [18]. At runtime, Topep parses strings using the grammar, yielding a parse tree. Topep checks grammar production constraints to classify strings as valid, invalid, or somewhere in between (questionable).

Toped includes another sentence-based user interface (with a style like that in Figure 2) for EUPs to define transformations, which operate on parse trees to reformat strings between formats. Transformations can change separators, reorder parts, use lookup tables, change capitalization and call other transformations (as functions) on parts. At runtime, Topep steps through each transformation's instructions to reformat strings.

Figure 2: Toped represents formats as a sequence of constrained parts. For example, a phone number would have an area code, exchange, and local number within that exchange. Constraints can be "always", "almost always", "often", "rarely", or "never" be true and are conjoined. The programmer can add new constraints by clicking on the "+info" buttons and can then select a type of constraint to apply. Supported constraints include specifying that the part should match another format or tope, or specifying that a part can repeat a certain number of times (perhaps with separators).



Each Phone Number has a part called the area code

The area code always ▼ has 3 of the following characters:
□ lowercase letters □ uppercase letters ☑ digits other characters:

The area code always ▼ is numerically >= ▼ 200

The area code never ▼ ends with ▼ a specific string: 11

+ info

+ part

Each Phone Number has a part called the exchange

The exchange always ▼ has 3 of the following characters:
□ lowercase letters □ uppercase letters ☑ digits other characters:

The exchange always ▼ is preceded by - and followed by
(You can leave one of these two fields blank if it does not apply.)

The exchange always ▼ is numerically >= ▼ 200

The exchange never ▼ ends with ▼ a specific string: 11

The exchange rarely ▼ is one of the following: 555

+ info

+ part

Each Phone Number has a part called the local

The local always ▼ has 4 of the following characters:
□ lowercase letters □ uppercase letters ☑ digits other characters:

The local always ▼ is preceded by - and followed by
(You can leave one of these two fields blank if it does not apply.)

+ info

3. TOOL FEATURES BASED ON TOPES

By taking advantage of topes, we have implemented custom runtime assertions for strings in web macros, transformation of strings in web macros that operate on tabular data structures, typo detection in spreadsheets, and input validation in web forms designed by EUPs. This section presents these new features.

3.1 Runtime assertions in web macros

A web macro tool like Robofox [5] watches EUPs perform operations in a web browser and attempts to determine the intent behind those actions. The tool generates a macro to represent that intent (generally as a sequence of steps) which the tool can later execute on new data.

For example, with Robofox watching, EUPs might go to a certain URL and copy a person's name from a particular tag in the web page. In order to identify which tag was selected, Robofox would record the tag's XPath and HTML ID attribute. In addition, it would record a visual pattern, which is an expression that identifies tags based on their proximity to key labels such as "Name:". EUPs could demonstrate pasting the person's name into a web form on another page. When Robofox replays the macro later, it would go back to the original URL, find the tag referenced by the XPath, ID and visual pattern, copy whatever text appeared in that location at runtime (which might be a different name than when Robofox created the macro), and then paste the name into the form on the next page.

Unfortunately, web sites evolve: We have documented many cases where webmasters changed pages' structure, added or removed form fields, and made other changes that would have confused web macro tools [14]. When Robofox executes the macro described above, the person name tag might have moved on the page. While Robofox has sophisticated heuristics that combine XPath, ID and visual path information to locate moved tags, these heuristics might lead Robofox to locate the wrong tag—such as a tag containing a social security number, credit card number, or some other data. Robofox would then proceed to execute the macro using this wrong information. For example, this might cause Robofox to paste a social security number into the form in the other page, which obviously would be highly undesirable.

To counter errors of this kind, we have extended Robofox with assertions based on tope formats. When constructing a macro, EUPs can highlight a clipboard item, which is a variable that is initialized by a copy operation in the macro, and open the TDE to create a new format or select an existing format stored on the computer. Robofox then creates an assertion specifying that after the copy operation, the clipboard should contain a string that matches the specified format. At runtime, if a string violates any constraint in the format, then Robofox displays a warning popup to explain that the assertion is violated, enabling EUPs to modify the macro or cancel execution if necessary.

Robofox lacks a formal plug-in interface, so in this case, our plug-in is a JavaScript library called by Robofox code. Since Robofox was one of the first tools that we integrated with the TDE, these assertions cannot reference topes implemented in later, more feature-rich versions of the TDE. Consequently, Robofox assertions can only reference topes that contain a single format, and they cannot reference formats whose parts reference other formats (for example, a citation format where parts such as the author names and page numbers are defined in separate formats). However, even in its current form, the integrated tool provides a powerful method for detecting when Robofox's clipboard contains incorrect text.

3.2 Transformation of strings in web macros

The CoScripter web macro tool—formerly called "Koala" [6]—has a new component called Vegemite, which allows EUPs to copy and paste data from web pages into tables in a "scratch space". While EUPs can type strings directly into scratch space cells, they can also create web macros that compute table cell values by posting other cell values through a web form and retrieving strings from the web server.

Topes fill a critical need in supporting reformatting operations involved in information workers' tasks [14]. For example, one task required reading phone numbers in one format from a page, then pasting the phone numbers in another format into a spreadsheet. To date, macro tools have been unable to perform these conversions, since they contain no specification for how to reformat strings between formats.

To support string reformatting, we implemented a proof-of-concept feature as a popup menu in Vegemite (which, like Robofox, lacks a plug-in interface, so our Vegemite plug-in is a JavaScript library). After populating a cell value with a string, EUPs can right-click on the cell value and select "Copy", which copies the string into the system clipboard. (Alternatively, EUPs could put a string on the system clipboard by executing a system copy command in another application.) Clicking on an empty cell and selecting "Paste Special" displays all possible reformatted versions of the string on the clipboard (Figure 3). EUPs can select a version, which Vegemite then pastes into the cell.

Vegemite populates the list of options by iterating through all known topes and testing the string with each format in each tope using Topep. For each successful parse, Vegemite calls the tope's transformation functions (again through the Topep API) to generate previews of the value as it would appear in the tope's other formats. In addition, Vegemite displays what parts of the string could be extracted from the value, based on the parts that compose the format, as shown in Figure 2.

For example, Figure 3 shows the options for pasting a phone number. It also shows a "Places" table, whose second column was populated by copying each cell of the first column and pasting the state abbreviation (which was extracted through a tope that recognizes strings in "City, ST" format). The third column was populated by copying each cell of the second column and pasting the string using the state name format.

Figure 3: Copying and pasting columns, with intervening extraction of parts or transformation of strings with topes.

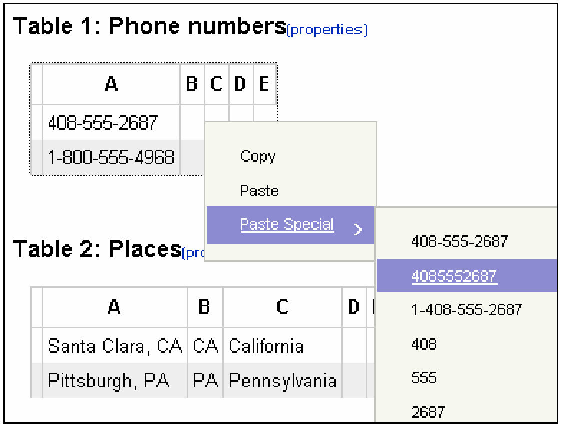


Table 1: Phone numbers(properties)

A	B	C	D	E
408-555-2687				
1-800-555-4968				

Table 2: Places(pr

A	B	C	D
Santa Clara, CA	CA	California	
Pittsburgh, PA	PA	Pennsylvania	

This proof-of-concept has three limitations to be addressed in future versions of Vegemite. First, Vegemite provides no user interface buttons or other controls to launch Toped, so EUPs presently have no way to add new formats in Vegemite. Second, this feature operates on individual cells; to save EUPs time, we could extend this feature to iteratively populate every cell in a column using the selected reformatting or extraction operation. Third, while the feature allows EUPs to populate cells through direct gestures, selected operations are not recorded for replaying later on different data; addressing this limitation will require improved integration between Vegemite and the replay facilities of CoScripter. While these three limitations highlight further opportunities to improve the usability of Vegemite, they do not diminish topes' contribution as a mechanism to recognize and transform string data.

3.3 Typo detection in spreadsheets

Among EUPs, spreadsheets are the most common platform for implementing computations and generating reports [16]. Prompted by the high error rate in spreadsheets [10], researchers have provided techniques for validating formulas and numeric data [2][12].

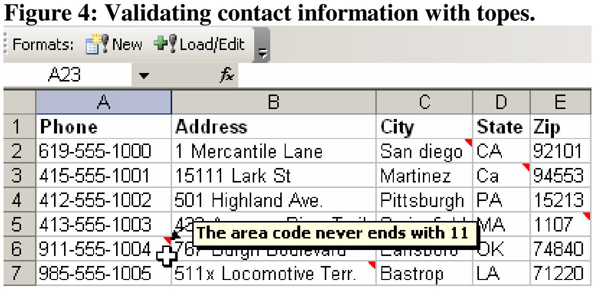
Yet in one study, nearly 40% of spreadsheet cells contained non-numeric, non-date textual data [3], which is consistent with a previous study which found that nearly 70% of spreadsheets were created for reporting purposes [4]. Despite the importance of textual data, spreadsheets offer no support for validating strings. (Though Excel lets users associate "social security number" or other labels with cells, this does not actually validate the data.)

To support finding and correcting typos in string data, we have provided a Microsoft Excel toolbar plug-in. Clicking the "New" button (Figure 4) starts the TDE, which infers a new format from highlighted cells and presents it for review and editing. Based on the specified format, the plug-in validates each cell and flags invalid cells with a small red triangle and a comment. To generate error messages, Topep concatenates together the violated constraints. EUPs can browse through these comments using Excel's Reviewing features, correcting or ignoring errors as desired.

EUPs can reuse and extend topes with additional formats and transformations via the "Load/Edit" button, thereby associating a multi-format tope with the highlighted cells. When making this association, the plug-in asks for a preferred format within that tope for these particular cells. For each cell, the plug-in parses the string with each of the tope's formats and selects the format that best matches the cell's string. If this best match differs from than the preferred format for the cell, then the plug-in calls the tope's transformation functions to put the cell's string into the preferred format.

To evaluate the effectiveness of validating spreadsheets with topes, we implemented topes for the 32 most common kinds of data in the "database" section of the EUSES spreadsheet corpus [3][18]. We found that our topes were 3.5 times as accurate as simply validating these kinds of data using regexps available on the web (which EUPs would generally not know how to create, anyway).

Figure 4: Validating contact information with topes.



	A	B	C	D	E
1	**Phone**	**Address**	**City**	**State**	**Zip**
2	619-555-1000	1 Mercantile Lane	San diego	CA	92101
3	415-555-1001	15111 Lark St	Martinez	Ca	94553
4	412-555-1002	501 Highland Ave.	Pittsburgh	PA	15213
5	413-555-1003	42[illegible]	[illegible]	MA	1107
6	911-555-1004	767 Burgh Boulevard	[illegible]	OK	74840
7	985-555-1005	511x Locomotive Terr.	Bastrop	LA	71220

The area code never ends with 11

3.4 Input validation in web applications

Even professional programmers often omit validation for input fields, including many form fields in "person locator" web applications that were created in the aftermath of Hurricane Katrina [18]. We interviewed six site creators, who explained that they intentionally omitted validation in order to provide end users with maximal flexibility. While they conceded that this resulted in accepting some invalid data, they emphasized that aggressive validation might have prevented many people from entering valid data.

We found similar examples of unvalidated fields in many other applications such as Google Base, which contains various forms including one for describing job openings. For example, Google has not implemented validation for text fields that accept a job type, industry, employer, or education level. Even for numeric fields, the forms accept unreasonable values (such as a salary of "-45").

EUPs generally have less programming training than professional programmers, and tools for EUPs offer no more support for validation than do tools for professional programmers—in both cases requiring programmers to create a regexp or a script to effect validation. Thus, even more than professional programmers, EUPs probably struggle to implement validation with existing techniques.

As in web macros and spreadsheets, the fundamental problem is that for many kinds of data, it is difficult to conclusively determine validity. For instance, no regexp can definitively distinguish whether an input field has a valid person name. No matter what regexp is created, there is certainly a valid person name that violates the regexp.

However, this limitation is even more serious in the web application domain than in the web macro and spreadsheet domains, since there is no way for web application users to override an "overzealous" regexp that rejects an unusual but valid input. (In contrast, for example, spreadsheet users can simply ignore red triangles and error flags inserted by Excel.) Consequently, web application designers must omit validation so as to avoid rejecting any invalid inputs.

Topes offer a solution: warn the application user about questionable inputs, so that they can be double-checked rather than rejected outright. To demonstrate, we created a plug-in for the web-form design tool in Microsoft Visual Studio.NET (which comes in an Express Edition for EUPs).

In Visual Studio.NET, the normal way for EUPs to create validation is to drag and drop a textbox widget from a toolbox onto the web form, then to drag and drop a `RegularExpressionValidator` widget alongside the textbox. They then specify a regexp and a fixed textual error message. At runtime, if an input violates the regexp, then the error message appears in red to the right of the textbox.

Our plug-in takes the form of a new validator widget. After dragging and dropping a textbox, EUPs drag our widget from the toolbox and drop it alongside the textbox. Once dropped on the page, the validator gives the option of selecting an existing tope, or creating a new tope by typing in examples of the data to be validated. The validator passes these examples to the TDE, which infers a format from the examples and presents it for review and customization before the tope's description file is stored on the web server.

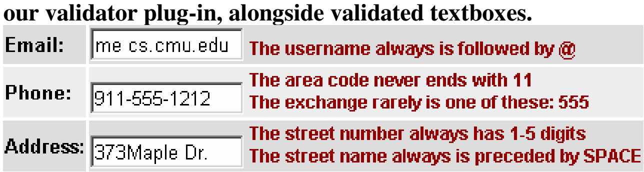
EUPs can add additional formats and transformations to the tope, if desired, and select a preferred format that this textbox's inputs should match. The validator automatically uses the tope to generate the necessary code for validating inputs. At runtime, this code parses the input string with each format to find the best matching format. If this format differs from the preferred format, then the code calls transformation functions to put the string into the preferred format.

If the string (after any transformation functions) matches the preferred format perfectly, then the code accepts the string. If the string does not match the format's grammar at all, or violates an "always" or "never" constraint, then the input is rejected; error messages are displayed using the standard red text beside the textbox (Figure 5).

If the string is questionable, the generated code displays an overridable warning message (in a popup window) so the end user can double-check and possibly correct the string before it is accepted. (The application's programmer can also specify alternate settings, such as always rejecting any input that does not match the preferred format exactly, thus dispensing with the warning message.)

We tested whether web application data could be accurately validated by the 32 topes that we implemented based on spreadsheet data (as mentioned in Section 3.3) [18]. We extracted data from the Google Base web application and one Hurricane Katrina web site, then identified 12 cases where tope implementations could be reused. We found that our topes were at least as accurate on web data as on spreadsheet data (and, in a few cases, more accurate, since the web data was from less diverse sources than the spreadsheet corpus).

Figure 5: Targeted human-readable error messages appear in our validator plug-in, alongside validated textboxes.



Email:	me cs.cmu.edu	The username always is followed by @
Phone:	911-555-1212	The area code never ends with 11 The exchange rarely is one of these: 555
Address:	373Maple Dr.	The street number always has 1-5 digits The street name always is preceded by SPACE

4. DISCUSSION AND FUTURE WORK

The general applicability of the topes model is shown by the wide variety of tool features that we have developed using topes, and is shown by the fact that topes are reusable across programs created in so many tools. Integrating the TDE into tools is straightforward once a new feature based on topes has been devised. For example, only two days were required to create a general-purpose, tope-based C# library for validating and reformatting strings embedded in HTML or XML [15].

We have received interest from other research teams who want to integrate the TDE with other tools. In response, we open-sourced all components of the TDE except for Toped (which requires a few enhancements discussed below). The plug-ins, inference algorithm (Topei), generator for context-free grammars (Topeg), and the parser (Topep) are available as C# libraries, and the parser is also available as a Java library (at http://www.cs.cmu.edu/~cscaffid/software.shtml).

Over the next few months, we will release an improved version of the TDE to correct minor deficiencies in Toped that were revealed by a preliminary user study [17]. Specifically, although it is possible to express topes for many kinds of data, implementing multi-format topes in the current Toped can be tedious. As one example, if EUPs want to add a certain constraint to more than one format, then it is necessary to open each format in the editor and manually add the constraint to each format. This extra work could be reduced by providing a mechanism to specify that a constraint should be applied to more than one format.

In addition, we will implement a repository server where programmers can publish and share formats with one another. This will facilitate incremental TDE improvements to further assist EUPs as they implement and reuse topes to validate data.

5. ACKNOWLEDGMENTS

This work was funded in part by the EUSES Consortium via NSF (ITR-0325273) and by NSF under CCF-0438929 and CCF-0613823. Opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the sponsors' views.

6. REFERENCES

[1] Blackwell, A. SWYN: A Visual Representation for Regular Expressions. *Your Wish Is My Command: Programming by Example*, Morgan Kaufmann, 2001, 245-270.

[2] Burnett, M., et al. End-User Software Engineering with Assertions in the Spreadsheet Paradigm. *Proc. Intl. Conf. Soft. Eng.*, 2003, 93-103.

[3] Fisher II, M., Rothermel, G. *The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms*. Tech. Rpt. 04-12-03, University of Nebraska—Lincoln, 2004.

[4] Hall, J. A Risk and Control-Oriented Study of the Practices of Spreadsheet Application Developers. *Proc. 29th Hawaii Intl. Conf. System Sciences*, 1996, 364-373.

[5] Koesnandar, A. *Building Dependable Web Macros Using Robofox*, Master's Thesis, Computer Science and Engineering Dept., University of Nebraska - Lincoln, 2007.

[6] Little, G., et al. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *Proc. Conf. Human Factors in Computing Systems*, 2007, 943-946.

[7] Miller, R., Myers, B. Outlier Finding: Focusing Human Attention on Possible Errors. *Proc. 14th Symp. on User Interface Software and Technology*, 2001, 81-90.

[8] Nardi, B., Miller, J., Wright, D. Collaborative, Programmable Intelligent Agents. *CACM*, Vol. 41, No. 3, 1998, 96-104.

[9] Ou, X., et. al. *Dynamic Typing with Dependent Types*, Tech. Rpt. TR-695-04, Dept. Comp. Sci., Princeton University, 2004.

[10] Panko, R. What We Know About Spreadsheet Errors. J. End User Computing, 10, 2 (Spring 1998), 15-21.

[11] Pierce, B. *Types and Programming Languages*, MIT Press, 2002.

[12] Rothermel, G., et al. WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation. *Proc. Intl. Conf. Soft. Eng.*, 2000, 230-239.

[13] Scaffidi, C. Unsupervised Inference of Data Formats in Human-Readable Notation. *Proc. 9th Intl. Conf. Enterprise Integration Systems – HCI Volume*, 2007, 236-241.

[14] Scaffidi, C, Cypher, A., Elbaum, S., Koesnandar, A., Myers, B. Scenario-Based Requirements for Web Macro Tools. *Proc. 2007 Symp. Visual Lang. and Human-Centric Computing*, 2007, 197-204.

[15] Scaffidi, C., Shaw, M. Accommodating Data Heterogeneity in ULS Systems. *2nd Intl. Workshop on Ultra-Large-Scale Software-Intensive Systems,* at the 30th Intl. Conf. Software Engineering, to appear.

[16] Scaffidi, C., Shaw, M., Myers, B. Estimating the Numbers of End Users and End User Programmers. *Proc. 2005 Symp. Visual Lang. and Human-Centric Computing,* 2005, 207-214.

[17] Scaffidi, C., Myers, B., and Shaw, M. Toped: Enabling End-User Programmers to Describe Data, *Conf. on Human Factors in Computing Systems – Work-in-Progress posters*, 2008, to appear.

[18] Scaffidi, C., Myers, B., Shaw, M. Topes: Reusable Abstractions for Validating Data, *Proc 30th Intl. Conf. Software Engineering*, 2008, to appear.