

Tool Support for Data Validation by End-User Programmers

Christopher Scaffidi
Institute for Software Research
School of Computer Science
Carnegie Mellon University
cscaffid@cs.cmu.edu

Brad Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
bam@cs.cmu.edu

Mary Shaw
Institute for Software Research
School of Computer Science
Carnegie Mellon University
mary.shaw@cs.cmu.edu

ABSTRACT

End-user programming tools for creating spreadsheets and webforms offer no data types except “string” for storing many kinds of data, such as person names and street addresses. Consequently, these tools cannot automatically validate these data.

To address this problem, we have developed a new 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, such as a mailing address. Specifically, each tope implementation contains software functions for recognizing and reformatting that tope’s kind of data.

With our tools, end-user programmers define new topes and associate them with fields in spreadsheets, webforms, and other programs. 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.

Categories and Subject Descriptors

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

General Terms

Reliability, Languages.

Keywords

Data, validation, end-user programming.

1. INTRODUCTION

Already, over 50 million people create spreadsheets, databases or webforms to collect and organize data [13]. One study of spreadsheets showed that nearly 40% of this data is short, human-readable strings of text, such as phone numbers, email addresses and person names, rather than numbers or formulas [2].

Unfortunately, just as several percent of a spreadsheet’s formulas typically contain errors [6], string data has many errors. Common examples include using a field to store data of the wrong kind (such as putting a person’s first and middle name in a field that should just have a first name [14] or even putting an age into an address field [11]), or storing valid data in the wrong format (such as a phone number with unexpected parentheses around the area code).

These errors occur in part because spreadsheets and webform development tools require programmers to use regular expressions (regexps) or even scripts to validate data. Regexps can only accept or reject a string, with no shades of grey in between. Yet many kinds of data are not well-described in this binary manner. For example, suppose a web application validates email addresses using a regexp based on the official specification [3]. Then it would allow users to enter email addresses with 64 characters in the address’s username (since the specification allows this), but obviously such a long username is highly unusual and questionable: it might be valid, but it probably is invalid. Regexps can only categorize data as valid or invalid, offering no third category for data that theoretically might be valid but still could benefit from manual double-checking by the person who entered that email address (or by some other person). Aside from this expressiveness problem, regexps are also hard to read and write for end-user programmers, who lack professional programming training [1]. Validation scripts are even more time-consuming and difficult to write than regexps, even for professional programmers, who consequently often choose to omit validation on webforms [11].

The second obstacle to more thorough validation is that valid data appear in multiple formats. For example, mailing addresses may specify a full street type such as “Avenue” or an abbreviation such as “Ave.”, and books may be referenced by title or ISBN. Prior to using data, an application typically must put data into a consistent format. Even if a programmer could write a regexp (with many disjunctions) to recognize multi-format data, the regexp still would leave inputs in multiple formats at the end of validation. Ideally, validation should help transform data into the format needed by the main application.

Based on these considerations, we present a tool set supporting a technique for recognizing questionable data and putting data into a consistent format. Our technique enables end-user programmers to define new abstractions called “topes”, each of which robustly describes one kind of data (such as a mailing address) independently of any particular software application and which is reusable across applications and across software development platforms. A tope contains multiple explicitly distinguished formats that recognize valid inputs on a non-binary scale, and it contains transformation functions to map values from one format to another.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE’08, May 10–18, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-079-1/08/05...\$5.00.

2. TOOL SET OVERVIEW

Our tool set is called the Tope Development Environment (TDE). It includes user interfaces for creating topes, as well as implemented algorithms for using topes to validate spreadsheet and web application data. It includes an API that other tool developers can use for novel purposes in other programming platforms.

2.1 Creating Topes

An end-user programmer performs two steps to implement each of a tope's formats. First, the programmer provides one or more examples of the data to validate. The TDE infers a basic format covering most or all of the examples and presents this format on-screen [14] (Figure 1). Second, the programmer reviews, customizes, and tests the format in Toped [9], which is a form-based syntax-directed editor [5]. The programmer can then implement other formats and transformation functions between formats. The TDE stores the tope implementation in an XML file, which can be referenced to validate data in spreadsheets and/or webforms.

Unlike regexps, topes not only distinguish between valid and invalid data, but they also can identify data that is questionable because it deviates slightly from the anticipated format. This is achieved by letting the programmer specify "soft" constraints that are often (but not always) satisfied by inputs. For example, the programmer could specify that an email address almost always contains 3 to 30 characters (Figure 2). The list of supported constraints includes requiring that a part should match another existing format, or specifying that a part can repeat a certain number of times (perhaps with separators between repetitions).

From these constraints, the TDE generates a context-free grammar for the format, with constraints on the grammar's productions, for parsing inputs at runtime [10]. Transformation functions operate on parse trees to reformat strings between formats. Transformations can change separators, reorder parts, use lookup tables on parts, change capitalization of parts, and call other transformations (as functions) on parts.

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 the Toped UI, 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. All shaded boxes are also accessible through an API.

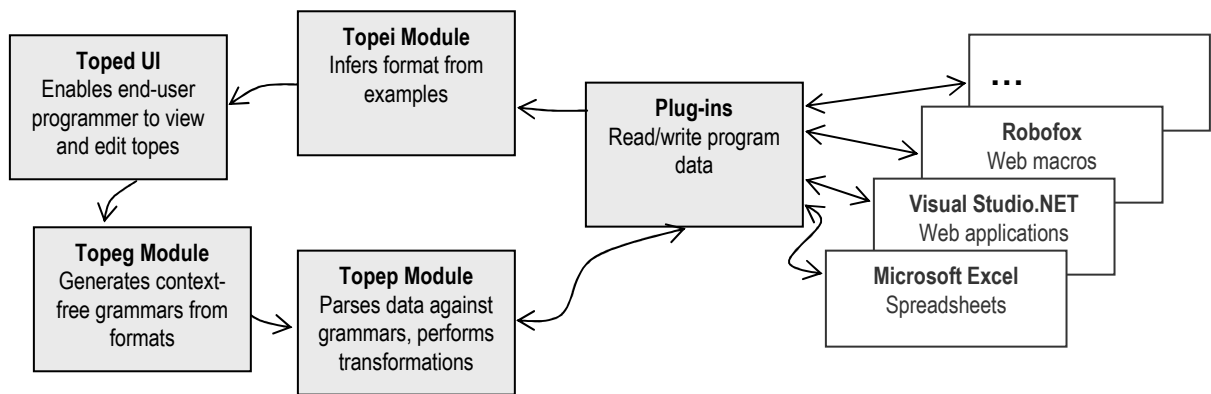


Figure 2: Toped represents formats as a sequence of constrained parts. For example, an email address would have a username, hostname, and domain (not shown, to conserve space). Constraints can be "always", "almost always", "often", "rarely", or "never" be true and are conjoined. The programmer can add new constraints by clicking on the "+info" button 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 the part should be a number in a certain range.

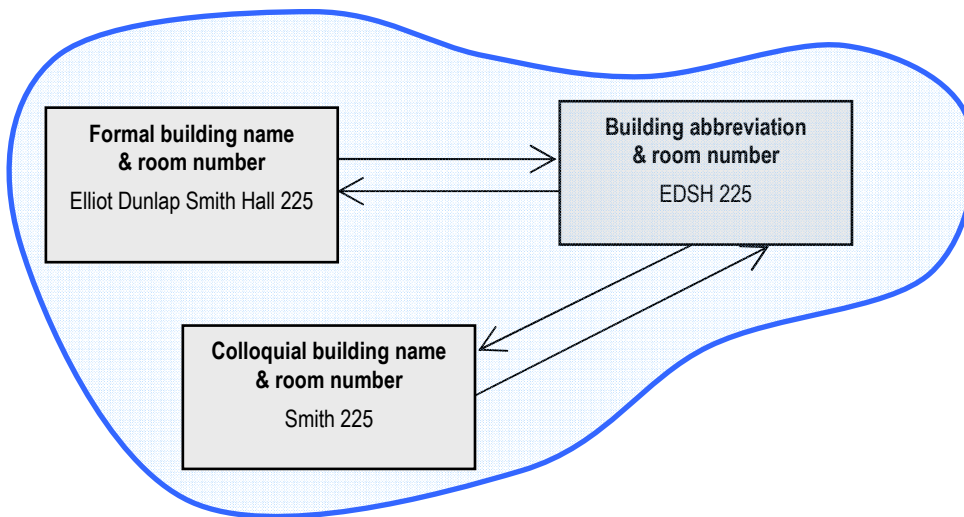


Figure 3: This notional depiction of a tope shows the three primary formats (grey boxes) used for room numbers at Carnegie Mellon University. Each format has two parts: a building name and a room number. The building name can be written in three ways. The four transformation functions (arrows) would use a lookup table to convert among formats.

For example, a simple American phone number tope might have three formats: (###) ###-####, ###-###-####, and ###.###.####. It might have four transformation functions. One transformation would convert from format #1 to #2 by deleting “(“ and replacing “)” with “-”. The second would convert from #2 to #3 by changing hyphens to periods. The other two functions would be the inverses of those above. Since transformations can be chained, the number of transformations grows linearly with the number of formats. Figure 3 depicts another sample tope, highlighting the graph-like structure of each tope, with formats at the graph nodes and transformations on graph edges [11].

2.2 Using Topes

To validate data, the programmer selects a range of spreadsheet cells or a webform’s text field, then browses through the existing topes to select an appropriate one (or creates a new one).

In spreadsheets, the TDE’s Excel plug-in immediately reads each spreadsheet cell and passes its text into the TDE’s parser. The parser returns 1 if the parse succeeds and the string violates no constraints; it returns 0 if the parse fails or the string violates constraints that should always be true; and it returns a number in between 0 and 1 if the string violates soft constraints. The parser provides a summary of any violated constraints, which the plug-in displays as a targeted error message describing what is wrong with that string. For example, providing an unusual person last name like “von Neumann” might yield, “The last name almost always starts with an uppercase letter.” Spreadsheet users can choose to fix or ignore flagged data.

When a programmer uses the Microsoft Visual Studio.NET plug-in to associate a tope with a webform text field, the TDE generates JavaScript that checks inputs at runtime and displays targeted error messages for invalid data. The programmer can specify that questionable inputs should be allowed, but that a warning should appear so that the application user who provided the unusual value can double-check it. This approach rejects obviously invalid inputs but allows questionable inputs if they are confirmed. (The programmer can also specify alternate settings, such as always rejecting any input that is not definitely valid.)

Finally, the programmer can specify a “preferred format” in the tope for each spreadsheet cell and webform field. If the input is valid, but in the wrong format, then the plug-in (or generated JavaScript) will automatically transform the data into the preferred format. This is achieved by executing one or more of the tope’s transformation functions. The resulting value is displayed on-screen so that the application user can review the result before it is submitted through the webform to the server.

City:	<input type="text" value="New haven"/>	The city's word always starts with 1 uppercase letter
State:	<input type="text" value="CX"/>	The state abbrev always is one of these: AL, AK, AZ...
Zip:	<input type="text" value="8445"/>	The zip always has 5 digits

Figure 4: Because targeted error messages are based on the specific webform inputs provided at runtime, they are much more descriptive than typical messages in existing systems, such as “Invalid input. Please enter a valid phone number.”

3. VALIDATION AND TOOL MATURITY

To evaluate expressiveness, we have used the TDE to implement and use topes for dozens of kinds of data. These include 32 categories of data that occur most prevalently in the EUSES spreadsheet corpus’s “database” section [2][11], as well as 14 categories of data that we identified by logging what four administrative assistants typed into their web browsers over a 3 week period [10]. During these studies, we found four common kinds of topes: numeric topes such as area codes, proper nouns such as street names, closed-set enumerations such as American state names and abbreviations, and structured hierarchical data such as mailing addresses.

To evaluate usability, we conducted a user study of the format editor (in Toped) and found that it enables administrative assistants and students to quickly and correctly implement validation [9]. In a total of less than 30 minutes, participants implemented formats for phone numbers and two kinds of data for which we have never been able to find a regexp on the web: company names and primary address lines (just the street address, not the entire mailing address). For comparison, we asked a separate group of participants to perform these tasks

with the Lapis data-description language (which cannot identify questionable data or describe transformations) [4]. Toped users completed 59% more formats than Lapis users, and the resulting formats were 45% more accurate. Moreover, though not perfectly comparable, it appears that subjects completed our tasks with Toped more quickly and accurately than subjects completed similar tasks with regexps in an earlier study conducted during the development of the SWYN regexp editor [1].

As evidence of usefulness, we have not only integrated the TDE with Excel and Visual Studio.NET, but other researchers (with our help) integrated an early version of the TDE into Robofox [7]. End-user programmers use this tool to create web macros, which are programs that instruct a browser to perform a series of operations. For example, a web macro might tell a browser to visit a certain URL, copy a stock ticker symbol from a certain location on the page, go to another URL, paste the stock ticker symbol into a webform, submit the webform, and so on. One key problem with web macros is that if the web site's structure changes, then a stock ticker symbol may no longer appear in the anticipated location [8]. Thus, at runtime, the web macro might copy arbitrary text from that location. To guard against this, Robofox now allows end-user programmers to create an assertion specifying that the clipboard's contents should match a certain tope format. If the assertion fails, then the macro pauses and asks the user to verify that execution should proceed. This substantially increases the robustness of web macros against site evolution. We have also integrated the TDE with other programming tools, including a second web macro tool (CoScripter) to support transformation of strings in tabular data structures [7], and we have implemented an API to validate and transform strings in XML data [12].

Although these evaluations illustrate the expressiveness, relative usability, and usefulness of the TDE, they also highlighted several opportunities for improvement, particularly in the user interface of the editor (Toped). Although it is possible to express topes for many kinds of data, implementing multi-format topes can be tedious. As one example, if a programmer wants 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. (Our usability study participants did not run into this tediousness because they only implemented a single format for each kind of data.)

Because of these limitations in Toped, we have not yet open-sourced that module. However, the plug-ins, inference algorithm (Topei), generator for grammars (Topeg), and the parser (Topep) are available as C# libraries, and the parser is also available as a Java library¹.

Over the next few months, we will correct the known deficiencies in the TDE. In addition, to further support tope reuse, we are extending the TDE with a repository system where people can publish and find tope implementations. Repository search mechanisms will enable software engineers to identify suitable tope implementations based on quality criteria and based on relevance to new applications. Developing a repository will enable us to collect actual tope implementations as well as feedback from peo-

ple using topes in real applications. This will facilitate incremental TDE improvements to further assist software engineers as they implement and reuse topes to validate data.

4. ACKNOWLEDGMENTS

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

5. 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] Fisher II, M., and 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.
- [3] Internet Engineering Task Force. *RFC 2821: Simple Mail Transfer Protocol*, <http://tools.ietf.org/rfc/rfc2821.txt>
- [4] Miller, R., and Myers, B. Outlier Finding: Focusing Human Attention on Possible Errors. *Proc. 14th Symp. on User Interface Software and Technology*, 2001, 81-90.
- [5] Nardi, B. *A Small Matter of Programming: Perspectives on End User Computing*, MIT Press, 1993.
- [6] Panko, R. What We Know About Spreadsheet Errors. *J. End User Computing*, 10, 2 (Spring 1998), 15-21.
- [7] Scaffidi, C., Cypher, A., Elbaum, S., Koesnandar, A., Lin, J., Myers, B., and Shaw, M. Using Topes to Validate and Reformat Data in End-User Programming Tools. *4th Workshop on End-User Software Engineering*, at the 30th Intl. Conf. Software Engineering, 2008, to appear.
- [8] Scaffidi, C., Cypher, A., Elbaum, S., Koesnandar, A., and Myers, B. Scenario-Based Requirements for Web Macro Tools. *Proc. 2007 Symp. Visual Lang. and Human-Centric Computing*, 197-204.
- [9] 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.
- [10] Scaffidi, C., Myers, B., and Shaw, M. *The Topes Format Editor and Parser*. Technical Report CMU-ISRI-07-104 / CMU-HCII-07-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2007.
- [11] Scaffidi, C., Myers, B., and Shaw, M. Topes: Reusable Abstractions for Validating Data. *Proc. 30th Intl. Conf. Software Engineering*, 2008, to appear.
- [12] 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.
- [13] Scaffidi, C., Shaw, M., and Myers, B. Estimating the Numbers of End Users and End User Programmers. *Proc. 2005 Symp. Visual Lang. and Human-Centric Computing*, 2005, 207-214.
- [14] Scaffidi, C. Unsupervised Inference of Data Formats in Human-Readable Notation. *Proc. 9th Intl. Conf. Enterprise Integration Systems – HCI Volume*, 2007, 236-241.

¹ Available at <http://www.cs.cmu.edu/~cscaffid/software.shtml>