A Data Model to Support End User Software Engineering

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

Abstract

Many end user programming tools such as spreadsheets and databases offer poor support for representing data at a level of abstraction that is intuitive to users. For example, users must work with "strings" rather than person names, phone numbers, or street addresses. As a result, validating and manipulating data is difficult.

This thesis develops a new user-extensible model for semi-structured data items. Each "tope" within this model defines how to recognize a kind of data item based on format and context, and how to transform that kind of item among valid formats. To show the usefulness of this model, we provide an environment to help end-user programmers to create, share, and apply topes, enabling these users to quickly implement data validation and reformatting functionality.

1. Problem overview

Many tools exist for creating spreadsheets, web applications, and databases. In many cases, the intended user population is the "end-user programmer," a person who has enough skill to create simple software but who is not a professional software developer [8].

To characterize end users' needs for programming features, we performed several studies. Reanalysis of government data revealed that office workers comprise the largest end user population, and spreadsheets and databases are the most widely used programming tools [8]. To more precisely characterize users' programming practices, we surveyed over 800 end users (mostly managers) concerning use of programming tools and learned that data structure abstractions are more widely used than imperative programming features [6]. These studies suggest that a significant part of programming tools' value comes from their support for data manipulations.

However, this survey and our recent user studies revealed that tools provide inadequate support for domain-specific data [5][9]. For instance, browser-automation tools like Lapis [2] are unable to automate repetitive work in browsers because these tools cannot automatically reformat data (such as a state from "OH" to "Ohio"). We have documented many scenarios that are poorly supported by existing browser-automation tools [5], and we saw similar lookup and reformatting tasks in spreadsheets and web page design tools. For these and similar reasons, 25% of our survey respondents mentioned obstacles related to data reuse [9]. In fact, when we interviewed creators of "person locator" web sites after Hurricane Katrina, we learned that even professional programmers sometimes struggle with manipulating and validating small semi-structured data items such as phone numbers and mailing addresses [7].

Prior attempts to represent such abstractions have limitations. For example, Lapis [2] can recognize data patterns and automate browser operations but cannot reformat data. This limitation also applies to context-free grammars and regular expressions, which, in addition, are hard for end users to understand and create [1]. Apple data detectors can detect data patterns, but authoring patterns and relevant operators requires using a scripting language; "this task is for programmers only" [3]. Another approach, formal type systems [4] (and object-oriented languages), also requires users to define categories of data using fairly advanced languages.

2. Overview of approach

A successful approach must address the related work's limitations and meet four additional requirements. First, tool designers lack resources and knowledge to create every domain-specific abstraction, so we must enable ordinary end users to define new abstractions. Second, different people have different expectations of data abstractions (e.g.: American phone numbers versus British), so our model must be expressive and flexible. Third, each abstraction has exceptions (e.g.: person last names may have a space or hyphen), so our model must accommodate valid data items that do not exactly match the dominant format. Finally, some formats are used by many people rather than custom to each person, so abstractions should be sharable.

To meet these requirements, we will enable users to define topes, each of which is a family of formats. A tope will contain functions for estimating the confidence that a string is properly formatted and for reformatting strings among valid formats. Topes will be stored in organization-specific repositories and customized by system administrators per the needs of users at that organization. For example, administrators at Carnegie Mellon could customize our repository to define "Oracle string," our local data abstraction representing project numbers. Vendors could offer topes as well and host them public repositories; there may be other general tope repositories on the web for topes used by many organizations.

3. Progress to date and contributions

The primary contribution is the *topes* idea, a lightweight model for defining data formats and transformations among them. We will implement a network of repositories so users can store and organize topes, and we will create plug-ins so programming tools can apply topes directly or use them to generate appropriate code for programs created by users.

We have built the *basic topes editor and parser*, with a user interface that represents topes in English so users can create, review, and customize topes. Our parser tests a string against a format and indicates a level of confidence that the string is properly formatted; our parser generates natural language explanations when it assigns low confidence. We will extend the system so users can define data reformatting.

We have implemented a *format inference algorithm* that examines example data and creates a format. Users can review and customize the format, then store it and use it to find outliers in the training data.

We will design a *tope guessing algorithm* to look at strings and determine which topes in the repository are probably appropriate for describing those strings.

We will design a *tope meta-model* to record co-occurrence of topes with contextual cues, so the system can parse and guess topes more accurately. To help users evaluate which topes to trust, we will adapt software credentials [10] to record namespaces, authorship, accuracy and other information.

4. Methods and evaluation criteria

We will evaluate the model's expressiveness by defining topes for data that we observed in our studies [5][9]; we will conduct user studies to evaluate usability.

Ease of use will come at the cost of power. Whereas formal types systems offer a "method for proving the absence of certain program behaviors" [4], topes estimate confidence of validity. Since topes offer no ironclad guarantees, the benefits of formal types—increased reliability, security, and readability [4]—may not apply to topes.

Thus, after synthesizing the pieces into a working system, we will perform user studies to evaluate whether the environment enables users to produce programs with improved quality. In particular, we will examine whether topes enable users to create spreadsheets and web applications that are more correct, as these are two widely used kinds of programs [8]. In addition, we will evaluate if equipping tools with topes leads to improved constructability and maintainability, since using user-intuitive abstractions (rather than primitives like floats or strings) should lead to improved readability.

5. Acknowledgements

Thank you to Brad Myers and Mary Shaw for numerous suggestions concerning topes. This work was funded in part by the National Science Foundation (ITR-0325273) via the EUSES Consortium and by the National Science Foundation under Grant CCF-0438929. 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.

6. References

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

[2] R. Miller, B. Myers, Integrating a Command Shell into a Web Browser, *USENIX 2000 Annual Technical Conference*, 2000, 171-182.

[3] B. Nardi, J. Miller, D. Wright, Collaborative, Programmable Intelligent Agents, *Comm. ACM*, *41*, 3 (March 1998), 96-104.

[4] B. Pierce, *Types and Programming Languages*, 2002.

[5] C. Scaffidi, A. Cypher, S. Elbaum, A. Koesnandar, B. Myers, *The EUSES Web Macro Scenario Corpus, Version 1.0*, Technical report CMU-HCII-06-105, Carnegie Mellon University, November 2006.

[6] C. Scaffidi, A. Ko, B. Myers, M. Shaw, Dimensions Characterizing Programming Feature Usage by Information Workers, *Proc. 2006 Symp. Visual Languages and Human-Centric Computing*, 2006.

[7] C. Scaffidi, B. Myers, M. Shaw, Trial by Water: Creating Hurricane Katrina "Person Locator" Web Sites, *Leadership at a Distance*, 2007, to appear.

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

[9] C. Scaffidi, M. Shaw, B. Myers, Games Programs Play: Obstacles to Data Reuse, *Proc. 2nd Workshop on End User Software Engineering*, 2006, 22-24.

[10] M. Shaw, Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does, *Proc. 8th Intl. Workshop on Software Specification and Design*, 1996, 181-185.