A Lightweight Model for End Users' Domain-Specific Data

Christopher Scaffidi
School of Computer Science, Carnegie Mellon University
cscaffid@cs.cmu.edu

Abstract

Many end user programming tools lack adequate support for domain-specific data. We will design a lightweight representation for categories of data, called "topes," and develop simple methods that end users and system administrators can use to define new topes. To evaluate this approach, we will improve programming tools so end users can write programs that recognize data as instances of topes and manipulate them accordingly. We expect that these enhancements will help end users produce higher quality software.

1. Introduction

Many tools exist to help end users create software. For example, office workers now have design tools for spreadsheets, web pages, and databases. However, most tools only allow users to work with data primitives that the tool developer chose in advance to support. Unfortunately, tool developers' notion of what primitives are useful never entirely match users'.

In order to characterize end users' most commonly occurring data, we performed a series of studies. We began by reanalyzing government statistics on Americans' computer use at work, which revealed that the largest end user population is office workers, and the most widely used programming tools are spreadsheets and databases [5]. To more precisely characterize users' programming practices, we surveyed over 800 end users (mostly managers) concerning their use of programming tools and learned that data manipulation programming features are more commonly used than imperative programming features [4]. These studies suggest that a significant part of programming tools' value comes from their support for data manipulations.

However, our recent contextual inquiry of office workers (as well as our survey mentioned above) revealed that tools often provide inadequate support for domain-specific data [6]. For instance, we observed workers doing many "look up" operations while filling out web forms in the browser; one important example was looking up government-approved per diem rates for municipalities. At first, it seemed as though these repetitive actions could be automated with web macros using a tool like Lapis [1]. However, on closer reflection, it became clear that these tools could not automate most tasks because they could not automatically convert data from one format to another; for example, automating per diem lookup would require reformatting state abbreviations to names (e.g.: "OH" to "Ohio"). We saw similar lookup and reformatting tasks in spreadsheets and web page design tools, as well.

In summary, tools often force users to treat data as numbers or undifferentiated "text" rather than ISBNs, phone numbers, address lines, person names, document titles, and so forth. Prior attempts to represent such abstractions have shown various limitations.

For example, as mentioned above, Lapis [1] can recognize various data patterns and automate browser operations but cannot reformat data automatically. A second approach, Apple data detectors, also can detect data patterns, but authoring patterns and relevant operators requires using a full scripting language; "this task is for programmers only" [2]. Moreover, detectors must be installed on each machine rather than configured organization-wide, which could inhibit deployment of organization-specific data detectors. A third approach, formal type systems [3], also requires users to define categories of data using fairly advanced languages.

To succeed, our approach must address these limitations as well as three additional issues. First, different people have different notions of what each abstraction "looks like" (e.g.: American phone numbers versus British phone numbers). Second, there are exceptions for each abstraction (e.g.: "000-121-2833" is not a valid American phone number). Finally, tool designers lack resources and knowledge to write code for every conceivable domain-specific abstraction. For these reasons, workers must be able to define and customize data representations on a per-organization or even per-user basis.

2. Proposed approach

We propose to meet these requirements by enabling end users to extend tools with new data abstractions as needed. Each such abstraction, provisionally called a "tope," is defined by a set of rules:

- For estimating the likelihood that a string of characters is of this tope ("isa")
- For estimating the likelihood that two strings of characters refer to the same instance ("equality")
- For transforming among different formats of this tope ("isotopes")

Tope definitions will be stored in organization-specific repositories and customized by system administrators according to the needs of end users at that organization. For example, administrators at Carnegie Mellon University could customize our repository to define "Oracle string," which is a data abstraction representing project numbers at our organization.

Tope definitions contain only a few rules, so we believe that we can develop methods enabling even ordinary end users to define custom topes for personal use. For example, dialog-driven programming-by-example could be used to define some tope patterns through machine learning. For topes corresponding to a finite set, it may be more convenient to let the user define the list of valid instances. Finally, some topes can be approximately represented using a context-free grammar.

This approach promises two immediate applications. First, topes could be used to provide automatic input validation in end user programs; for example, editors for spreadsheets and web forms could use tope definitions to automatically generate validation code that would highlight potentially erroneous inputs. Second, topes could be used to automate data reformatting; for example, web macro recorders could use tope definitions to detect that data needs reformatting (e.g.: "OH" to "Ohio"), and then reformat the data accordingly within the macro. Automatic data transformations may also prove useful during copy-and-paste operations and during database bulk import/export tasks.

3. Proposed evaluation criteria

The primary goal of this proposal is to provide abstractions supporting the diversity of domain-specific data normally encountered by end users. The approach outlined above aims to achieve this through an extremely lightweight representation amenable to a variety of tope definition methods. We will perform several studies to verify that end users or system administrators can use these methods to define topes, and we will determine which methods are easiest to use for different types of topes. In particular, we will focus on the usability of topes for office workers, since they are currently the largest population of end users [5].

However, this ease of use comes at the cost of power. In contrast to formal type systems, which offer a "method for proving the absence of certain program behaviors," [3] tope rules only will estimate the *likelihood* of isa and equality relationships. In other words, topes cannot offer the ironclad guarantees that formal types can, so it is possible that the benefits of formal types—increased software reliability, security, and readability [3]—may not apply to topes. Thus, we must evaluate whether equipping programming tools with topes actually leads to improvements in key software quality attributes.

Because spreadsheets are presently the most common end user programming tool, we intend to include Excel in our list of tools. Web browsers are widely used, as well, so we anticipate enhancing web macro recorders and web application design tools.

In terms of key quality attributes, the unreliability of end users' programs, especially spreadsheets, has attracted a great deal of attention in the literature, so we will include measures of correctness in our evaluation. In particular, with respect to web applications, we intend to assess whether equipping design tools with topes leads to programs with fewer security flaws. Finally, we will evaluate whether equipping programming tools with topes leads to improved program constructability and maintainability, since using domain-specific abstractions (rather than raw primitives like float) should lead to improved readability.

In the short-term, defining topes and organizing them into repositories will involve extra work by system administrators and end users. However, in the long-term, if many people can reuse the abstractions in a variety of programming tools, then creating high-quality programs may become easier overall. Therefore, we believe that topes offer a promising opportunity to help end users produce more effective software.

4. Acknowledgements

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

5. References

[1] Miller, R., and Myers, B. Integrating a Command Shell into a Web Browser. *Proc. USENIX*, 2000, 171-182.

[2] Nardi, B., Miller, J., and Wright, D. Collaborative, Programmable Intelligent Agents. ACM, Vol. 41, No. 3, 1998, 96-104.

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

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

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

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