A First Look at End-user Visual Computation Supporting Sharing & Reuse with Inflo

Jonathan Lung
and Steve Easterbrook
Department of Computer Science
University of Toronto
Toronto, Ontario
Email: {lungj, sme} @ cs.toronto.edu

***Abstract*—*Inflo* is a web-based tool designed to support quantitative decision-making problems such as carbon calculators. Using the dataflow programming paradigm, *Inflo* provides an extensible environment for end-user programming. *Inflo* is built around the idea of collaboration and openness for opportunistic code-reuse and unplanned code sharing.**

I. Introduction

Spreadsheets are a popular platform for end-user programming. Many ideas can be expressed through a series of formulae — from students' grades for a course to budgets for large corporations. Spreadsheets can also be used for quantitative decision making such as evaluating the cost-effectives of different scenarios. Though spreadsheets have proven versatile, they lack properties that would be useful for such decision making problems.

Because end-user programmers do not intend for their software to be used by a wide audience, their software is generally not engineered for reuse nor is it usually tested to the same extent as software intended for public release [1]. Software engineering (SE) practices such as version control, software libraries, and the ability to comment code are generally unused in end-user SE. Providing a system that automatically supports these tasks in an informal environment may encourage better end-user SE.

II. *Inflo*

A. Overview

Inflo is a web application used to collaboratively construct computations in a graphical environment to support quantitative decision making. Computations are drawn as trees where the root is the result of the computation (see figure I). Each node is a computation step in the same way a cell is the basic unit of computation in a spreadsheet. Unlike spreadsheet cells, nodes have human-readable titles in lieu of cell names and may contain a rich text description to accompany them. The description can be used for anything, including providing references and explaining the content of a node.

B. Constructing a graph

The simplest way to construct a graph is one node at a time. A node is created explicitly through a toolbar button or implicitly while filling in the content of a node. The content of a node may be as simple as a string constant or as complex as a formula referring to the value of other nodes. Nodes whose values are referred to in this manner are automatically connected by an edge in the graph (see figure 1). While entering a formula, an autocomplete widget displays matching nodes to aid in graph construction. Other methods for constructing a graph are discussed in section II-D.

Because a graph can become extremely complex, *Inflo* allows nodes to be collapsed (see bottom nodes in figure I). This is analogous to code-collapsing in text-based programming languages.

C. Preventing, detecting, and eliminating bugs

Inflo's intended use as a collaborative decision-making tool informed many aspects of its design. One area that received attention was the prevention, detection, and elimination of errors in reasoning and computation. One of the envisioned use cases of *Inflo* is back-of-the-envelope calculations created by novice users, so techniques for dealing with errors such as assertion checking and formal logic were avoided.

One method of dealing with bugs is to prevent a certain class of them from occurring. For example, in an argumentation graph, a cycle is formed if and only if circular logic is employed. For this reason, *Inflo* prohibits cycles in the computation graph. As a consequence, *Inflo* is not Turing complete (however, see section II-E).

Inflo supports dimensional units (e.g., "g of CO2e" in figure 1) and their automatic propagation through computations. This automaticity provides not only convenience to the user, but it also serves as a sanity check when the units are not what the user expects. Unit support also plays a role in detecting bugs since users are warned when operations result in a dimensional mismatch. Implicit unit checking has been shown to be effective at identifying spreadsheet errors [2]. However, based on user feedback, when constructing computations or arguments using a top-down approach, worrying about units early on imposes an unnecessary burden on the user, interfering with the construction process. As such, users can indicate provisionality[1] through the generic unit "?".

Because a node may be used more than once in the same calculation and a tree is used to show the computation, the same node could visually appear multiple times. To help visualize

1 *Provisionality* is used in the sense of Green et al. [3]

978-1-4577-1247-0/11/$26.00 ©2011 IEEE

data dependencies and how changes to a node will propagate, a coloured line is drawn from each visual instance of a node towards the root. Further, the collaborative features of *Inflo* are intended to help with debugging and patching through Linus' Law: sharing a graph with more users increases the likelihood that bugs are spotted.

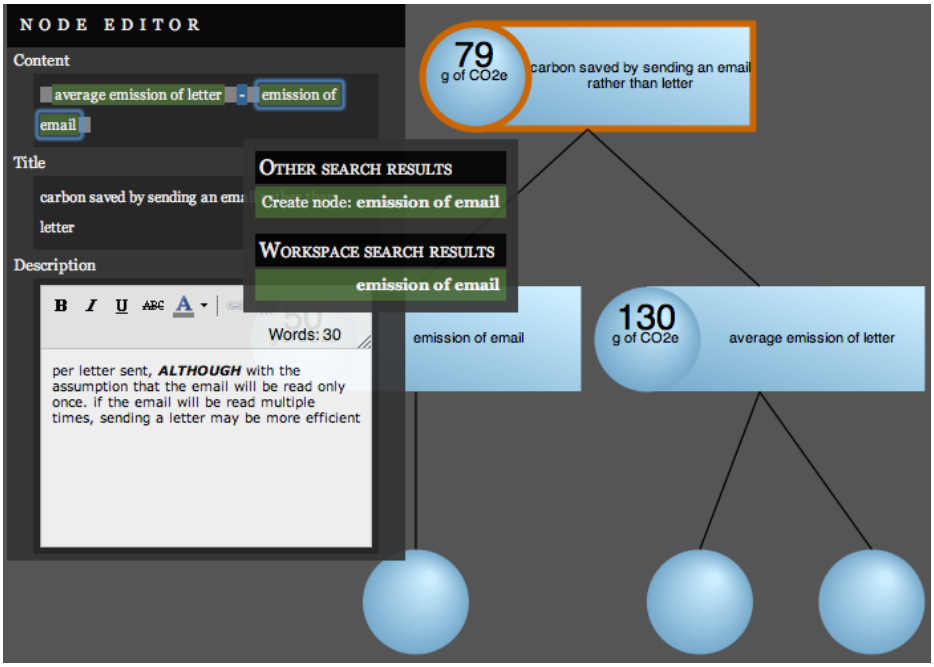

Fig. 1. A sample argument shown as an Inflo graph.

D. Sharing & Reuse

Each version of a node has a unique HTTP URL to allow sharing and reuse of computations. The URL for a node can be obtained by drag-and-dropping it into an application. For example, to share an *Inflo* graph on a blog, one can drag-and-drop a node from *Inflo* into the text field used for writing the blog post. Following these links result in a new instance of *Inflo* being started displaying an editable copy of the version linked to. Changes result in new versions being created leaving the linked-to version unaffected.

Subsets of a graph (or entire ones) can be incorporated into new graphs. One way of doing this is by dragging a node from an existing graph into a different graph, importing it and all its dependencies into the second graph. This allows opportunistic code/knowledge reuse at any level of abstraction. For example, once the carbon footprint of an e-mail is calculated, even as part of another calculation, it can be used in other calculations by dragging and dropping.

To facilitate reuse, *Inflo* includes the ability to publish graphs. Versions of graphs and nodes can be made publicly searchable in a collectively curated library. When graphs are published in this fashion, they appear in a revision history.

The ability to reuse nodes can expedite the graph creation process, too. To support this, the autocomplete feature mentioned in section II-B searches not only the current graph for matching nodes, but also publicly shared nodes. In addition to speed, using previously created nodes has an added benefit: when viewing a graph, *Inflo* notifies the viewer when newer versions of nodes within the graph exist, for example when an estimated value is replaced with empirical data. Users may then choose to update the graph to the newest versions of nodes, serving as a software patching mechanism.

E. Extensibility

Inflo was designed with user extension in mind. One of the many ways in which users can extend *Inflo* is through the use of user-defined macros written in Javascript. Using these macros, users can bypass limitations in the computations that can be expressed in *Inflo*. Though these macros behave largely as built-in functions, there are slight differences due to security concerns over arbitrary Javascript code execution.

III. Conclusion

Inflo introduces a novel way to do dataflow programming through a design revolving around collaboration. From reusing, sharing, and discovering nodes to the web-native links that allow interoperability with existing web services, *Inflo* integrates tools normally reserved for professional software development into an environment intended for novice end-users.

References

[1] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, pp. 21:1–21:44, April 2011. [Online]. Available: http://doi.acm.org/10.1145/1922649.1922658

[2] C. Chambers and M. Erwig, "Dimension inference in spreadsheets," in *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, sept. 2008, pp. 123 –130.

[3] T. Green, A. Blandford, L. Church, C. Roast, and S. Clarke, "Cognitive dimensions: Achievements, new directions, and open questions," *Journal of Visual Languages & Computing*, vol. 17, no. 4, pp. 328 – 365, 2006, ten Years of Cognitive Dimensions. [On-line]. Available: http://www.sciencedirect.com/science/article/pii/S1045926X06000280