Socially-Adaptable Interfaces: Crowdsourcing Customization

Ben Lafreniere, Michael Terry
HCI Lab
University of Waterloo
{bjlafren,mterry}@cs.uwaterloo.ca

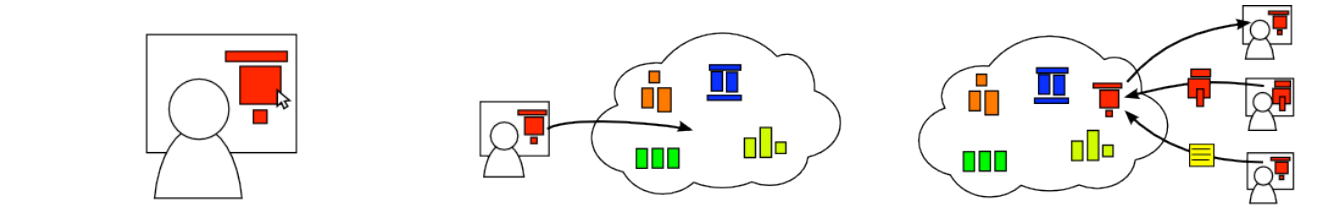

Figure 1.

ABSTRACT
This paper reports our work developing *socially-adaptable interfaces*, interfaces that crowdsource the creation of task-specific interface customizations and instantly share them with all users of the application.

We start with an introduction to the socially-adaptable interface concept and Adaptable Gimp, a system that we are developing to test these ideas. We then discuss a number of new interaction paradigms for feature-rich applications that this work opens up.

Author Keywords
Adaptable interfaces, crowdsourcing, human-powered interfaces, wikis.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

For many reasons, modern desktop applications are packed with more functionality than is required by any given user, and particularly for any given task [4,5,6,7,10]. This excess interface complexity has been shown to have a negative effect on individuals' use of the software, both quantitatively [2,5] and qualitatively [10]. Adaptable interfaces, which give users a mechanism for customizing the interface, have been demonstrated to be an effective solution when correctly designed [9]. Unfortunately, users are hesitant to spend time customizing interfaces, since it takes time away from their primary task [8].

We are exploring *socially-adaptable interfaces* to address excess interface complexity. In this approach, users create *task sets*, task-specific customizations of the application's interface, which are automatically made available to all users of the application through an online repository (see Figure 1). In essence, the application supports crowdsourcing the creation of interface customizations. Once a comprehensive collection of task sets exists, users can quickly customize the interface by using a keyword search to find and install task sets. The result is an *imperative interface* paradigm, in which users issue direct commands to the interface. A keyword search for a task set amounts to issuing the command: "Adapt the interface for performing task X".

In Adaptable Gimp, a system we are developing to test this concept, task sets are collections of commands that can be displayed in a customizable toolbox. Each uploaded task set is associated with a wiki page, allowing any user in the community to add or remove commands from the task set, or contribute free form documentation.

User-created task set documentation acts as an organic bridge between how users conceptualize tasks and the application's functionality for performing those tasks. This and other contextual data created around task sets allows us to automatically categorize users, learn about their tasks, and make conclusions about the application itself. This could feed techniques for connecting users with one another or support human computation paradigms. Contextual data could also allow user interfaces to become *self-correcting*, providing developers with a hotline to their users and a constant stream of quantitative and qualitative data on how their application is used. Finally, it opens new possibilities for researchers to perform large-scale user studies.

In the following sections, we discuss these possibilities further. We start by motivating and more formally defining our socially-adaptable interface concept and introducing Adaptable Gimp.

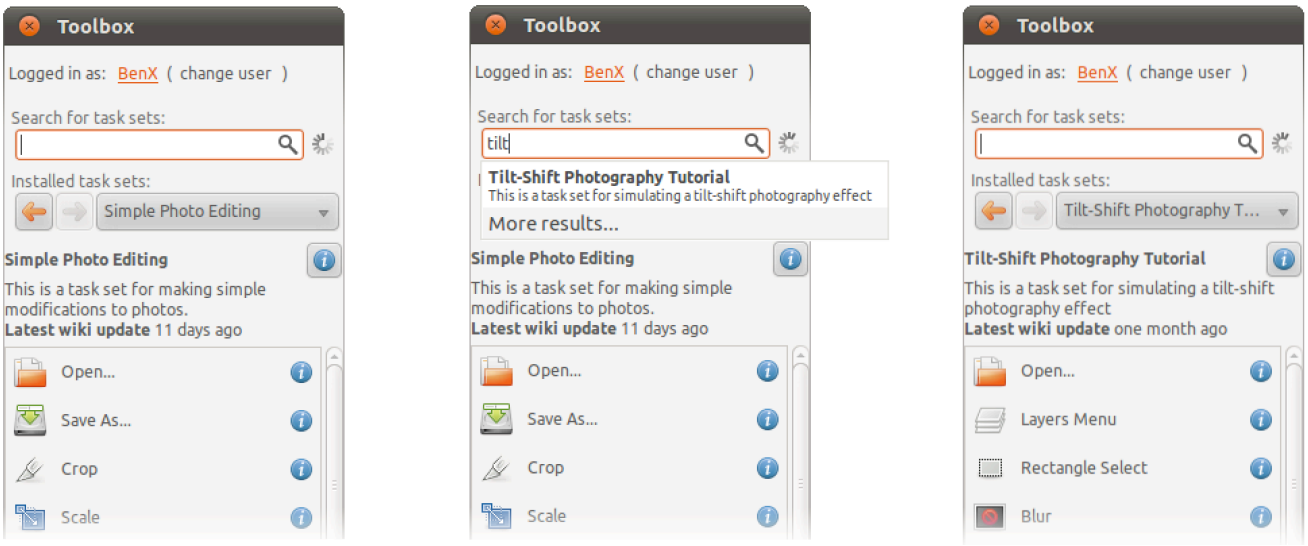

Figure 2. The process of searching for and installing a task set in Adaptable Gimp.

SOCIALLY-ADAPTABLE INTERFACES

Functionality in feature-rich applications is seldom organized to help the user with perform specific tasks. Rather, it is organized around housing all of the available functionality somewhere in the interface, typically in a hierarchical organization. Even if the user understands the logic underlying the organization, commands for any individual task are spread throughout the interface. For novice users, this means that the interface provides few clues as to what commands to use for a given task. Experienced users may know the commands to use, but could still benefit from having them in a single location.

The solution to this problem seems obvious: organize functionality based on tasks. Unfortunately, this is difficult to do because feature-rich applications support huge numbers of distinct tasks and the organizations dictated by these tasks may conflict with one another.

We could overcome this problem in two ways. First, we could develop a single interface that organizes functionality based on a generalization across tasks. The Ribbon interface used in recent versions of Microsoft Office has adopted this approach; commonly-used commands are grouped by the broad category of task they are involved in (for instance, there is a Page Setup category that includes commands for setting margins, page orientation, page size, columns, breaks, etc.)

The second approach is an interface with many different organizations of functionality. Our approach falls into this category, with no bound on how many customizations can be created.

Similar to existing adaptable interfaces, our approach makes a part of the application's interface customizable, and gives the user an in-application method of adapting the interface. Interface customizations are artifacts, which we call *task sets*. The user can have many task sets installed at once and switch between them as needed. This supports (and encourages) the use of different task sets for different user tasks.

When a task set is first created it is uploaded to a shared online repository and a wiki page is associated with it. At this point, any user can modify the interface customization that the task set represents, or contribute to its documentation page.

Task sets in the online repository are available to all users from within the application's interface. A user who wants to perform a particular task can enter keywords into a search box, searching the repository and returning a list of matching task sets. Selecting a result installs the task set and customizes the user's interface. While using a task set, task set documentation can be viewed either on the web or from within the application itself, giving the user immediate access to instructions, notes, or other information contributed by the community.

Compared with existing adaptable approaches, there is less of a need to encourage individual users to spend time creating interface customizations. For a given task, only one user need create a task set, which may be used, modified and improved by others in the user community.

In the next section, we describe Adaptable Gimp, an application with a socially-adaptable interface that we are currently developing.

ADAPTABLE GIMP

Adaptable Gimp is a modified version of the GNU Image Manipulation Program (GIMP), an open-source image manipulation application with features similar to Adobe Photoshop.

The Adaptable Gimp toolbox is shown in Figure 2. From top to bottom, the toolbox consists of 1) a search box for searching the online repository, 2) an area for selecting

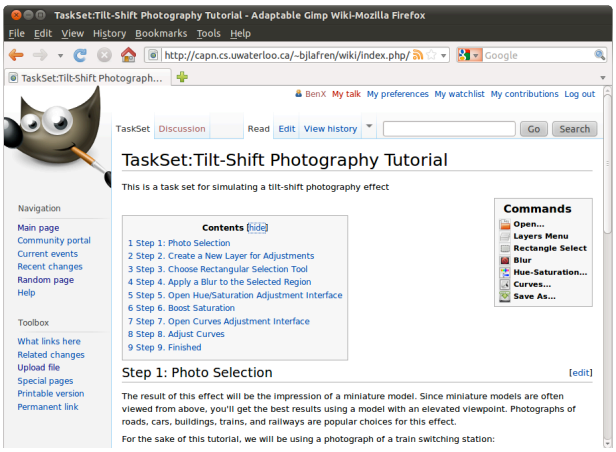

Figure 3. Each task set has an associated wiki page that can be edited by the user community to add/remove commands, or edit descriptive text.

among installed task sets using a dropdown, and 3) an area containing commands and information from the currently selected task set. Figure 2 shows the process of searching for and installing a *Tilt-Shift Photography Tutorial* task set.

The documentation page for the task set selected in Figure 2 is shown in Figure 3. We are using a modified version of the popular Mediawiki software to run the Adaptable Gimp wiki. Editing the page shown in Figure 3 allows the user to edit the documentation, but also presents an interface for editing the commands contained in the task set.

By providing an unstructured medium for editing task set documentation, we hope to encourage users to document task sets in interesting and inventive ways. Some of the uses that we anticipate for task set documents include: tutorials for performing specific tasks, documenting usage of a set of related commands, providing rationale for including commands in a task set, and documenting a user's experience using the application. Documentation on the wiki can be viewed from within the application as well, lowering the barrier to referring to it while using a task set.

Current status

As of the time of writing, we are nearing completion of the Adaptable Gimp application and associated wiki software. We plan to release the software to the public in early 2011.

In the next section, we will discuss some of the research possibilities for socially-adaptable interfaces.

RESEARCH QUESTIONS

In this section, we look at some of the research areas opened up by socially-adaptable interfaces, including:

- Building communities around desktop applications
- Studying imperative interfaces
- Crowdsourcing interface design
- Connecting users with one another
- Using the rich context created around users, tasks, and the application
- Connecting researchers and developers with the user community

We'll discuss each of these areas in turn.

Building customization communities

Two initially important research questions are *How do we build the community required to support a socially-adaptable interface?*, and *What will such a community look like?* These are important questions to answer to show that this is a viable technique, and to make it generalizable to a wide variety of applications.

A key question is how to provide a compelling user experience before the community has reached critical mass and become self-sustaining. For Adaptable Gimp, we plan to seed the repository with task sets for common tasks, culled from a study of ingimp, an instrumented version of GIMP [11], and GIMP-related search queries sampled from Google query logs [3].

Crowdsourcing interface design

Socially-adaptable interfaces give the community of users control over portions of an application's interface. We've kept this simple in Adaptable Gimp, where task sets consist of subsets of GIMP commands, but there is the potential to support much more radical interface customizations.

One could imagine many ways that interfaces could be customized to suit a particular task. Task sets could be designed to use particular hardware peripherals, such as WACOM tablets or touch-screen monitors. They could support input from sources that the developers may never have anticipated (say, input from a microphone varying brush thinkness). They could also be designed with the user performing the task in mind, providing special interfaces for children, users with special needs, or crowdsourced workers in systems such as Soylent [1].

Security is an obvious challenge that would need to be overcome before task sets could include scripts or custom UI elements, but the potential of allowing a community to radically tailor interfaces for particular tasks is compelling enough to make it worthwhile.

Connecting the user community

Socially-adaptable interfaces have the potential to connect users with other members of the user community in a much tighter way than is typical for desktop applications. The wiki serves as a natural place for users to discuss the application, its features, task sets, or anything else. Users can help each another learn about the application, solve problems, or connect over shared experiences.

The interface could also explicitly support direct connections between users to assist with performing tasks. Task sets could be equipped with a "Click here to talk with other users of this task set" feature, or even a "Do this for

me" feature that would farm out work to community volunteers or paid workers in a Soylent-style human-computation paradigm [1].

Rich contextual data

Socially-adaptable interfaces create an immense amount of contextual data around the interface and users' interactions with it. Community documentation, logs of task set searches, users' task set usage, and discussions around a task set each tell us much about users, the task they are performing, and the application itself.

For example, the act of searching for and installing a task set is essentially a declaration of the user's intended task. As a result, logs of search keywords provide records of users' intentions over time. This is data at a much higher level than traditional logs of command usage or interface events.

One use of this data would be to make inferences about users' skill levels or areas of expertise. This information could then be fed back into the interface, automatically tailoring it to the user, highlighting features that the user might be interested in, or connecting the user with similar users in the community.

Contextual data could also help one understand tasks performed with the application. For a given task, which commands do users most often use? Does this vary with a user's skill level? What tasks are related to one another? Answers to these types of questions could be used to organize task sets into broad categories (e.g. task sets for graphic artists, task sets for new users, task sets for users coming from a Photoshop background). They could also feed into intelligent tutoring systems, suggesting sequences of task sets to bootstrap a user's knowledge of the application.

Finally, contextual data could help us understand the application itself. For example, the corpus of task sets created by the community indicates the types of tasks users are using the application for. It may also give insights into tasks that the default interface provides poor support for. For instance, the existance of a task set might indicate that commands in the application are at too fine a level of granularity.

Connecting users with researchers and developers

In addition to generating data, socially-adaptable systems could serve as a platform for studying the user community. Researchers could administer surveys, perform remote user studies, or solicit qualitative feedback from users. In an interface where social interactions are an integral part of how the interface works, users may be more willing to volunteer their time to provide feedback to researchers or developers.

Developers could add simple feedback mechanisms into the interface to quickly gauge the opinion of the user community. For example, task sets could be outfitted with a "This task set should be a single command in the next version" button.

There may also be some particular benefits to socially-adaptable interfaces for open source software. Our previous work indicates that a primary motivation for open source developers to address usability issues is high-quality feedback from users [11]. By creating a public discourse around the interface, socially-adaptable interfaces provide passive feedback to developers as well as an easy way to connect with enthusiastic members of the user community.

AUTHOR BIO

Ben Lafreniere is a Ph.D. candidate at the University of Waterloo Human Computer Interaction lab, under the supervision of Professor Michael Terry. Ben's current work on socially-adaptable interfaces was motivated by previous work analyzing data from ingimp, an instrumented version of GIMP that gathered rich usage data from users [6].

REFERENCES

1. Bernstein, M.S., Little, G., Miller, R.C., et al. Soylent: a word processor with a crowd inside. *Proceedings of UIST 2010,* (2010), 313–322.
2. Carroll, J.M. and Carrithers, C. Training wheels in a user interface. *Commun. ACM 27*, 8 (1984), 800–806.
3. Fourney, A., Mann, R., and Terry, M. Characterizing the Usability of Interactive Applications Through Query Log Analysis. *Proc CHI '11*, 10 pages.
4. Greenberg, S. *The computer user as toolsmith: The use, reuse, and organization of computer-based tools*. Cambridge University Press, New York, USA, 1993.
5. Hsi, I. and Potts, C. Studying the Evolution and Enhancement of Software Features. *Proceedings of ICSM '00*, (2000), 143–151.
6. Lafreniere, B., Bunt, A., Whissell, J., Clarke, C.L.A., and Terry, M. Characterizing large-scale use of a direct manipulation application in the wild. *Proceedings of GI 2010*, (2010), 8 pages.
7. Linton, Joy, and Schaefer. Building user and expert models by long-term observation of application usage. *Proceedings of UM '99*, Springer-Verlag New York, Inc. (1999), 129–138.
8. Mackay, W.E. Triggers and barriers to customizing software. *Proceedings of CHI '91*, (1991), 153–160.
9. McGrenere, J., Baecker, R.M., and Booth, K.S. A field evaluation of an adaptable two-interface design for feature-rich software. *ACM Trans. Comput.-Hum. Interact. 14*, 1 (2007).
10. McGrenere, J. and Moore, G. Are we all in the same "bloat"? *Proceedings of GI 2000*, (2000), 187-196.
11. Terry, M., Kay, M., and Lafreniere, B. Perceptions and practices of usability in the free/open source software (FoSS) community. *Proc of CHI 2010*, 999–1008.