ShowMeHow: Translating User Interface Instructions Between Similar Applications

Vidya Ramesh, Charlie Hsu, Maneesh Agrawala, Björn Hartmann
University of California,Berkeley — Computer Science Division
533 Soda Hall, Mailcode 1776, Berkeley, CA 94720
{vidyaramesh,cphsu}@berkeley.edu,{maneesh,bjoern}@cs.berkeley.edu

ABSTRACT
Many people learn how to use complex authoring applications through tutorials. However, user interfaces for authoring tools differ between versions, platforms, and competing products, limiting the utility of tutorials. Our goal is to make tutorials more useful by enabling users to repurpose tutorials between similar applications. We introduce *UI translation interfaces* which enable users to locate commands in one application using the interface language of another application. Our end-user tool, *ShowMeHow*, demonstrates two interaction techniques to accomplish translations: 1) direct manipulation of interface façades and 2) text search for commands using the vocabulary of another application. We discuss tools needed to construct the translation maps that enable these techniques. An initial study (n=12) shows that users can locate unfamiliar commands twice as fast with interface façades. A second study shows that ShowMeHow enables users to work through tutorials written for one application in another application.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors, Documentation

Keywords: tutorials, instructions, mapping, translation

INTRODUCTION

Over the last two decades, we have witnessed a proliferation of software for creating and editing digital content – images, audio, video, animation, diagrams, illustrations, and 3D models. The interfaces of such applications can be dauntingly difficult to learn and use effectively. Often, the more powerful an application is, the more complex its user interface becomes. As a result, many users rely on tutorials to learn how to execute procedural tasks in the applications.

Tutorials describe a sequence of steps users must perform to reach a desired goal. The most effective tutorials contain screenshots and use interface-specific operation names to visually depict the GUI elements that must be manipulated in each step [10]. Creating high-quality tutorials and other documentation materials is often time consuming and requires expertise in the underlying application. As a result, the tutorial space is fragmented — a tutorial in one application often does not have equivalent counterparts for other, similar applications or other versions of the same application.

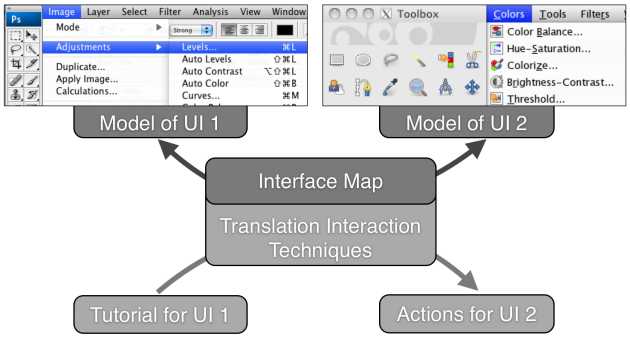

Figure 1: ShowMeHow helps users to translate instructions between applications that offer similar functionality, e.g., the image editing applications Photoshop and GIMP.

Consider a tutorial on red-eye correction in an image manipulation program. Many applications, including Photoshop, GIMP, Darkroom, and Aperture, support correction of red-eye. However, their interfaces are different enough that a tutorial written for one application is not immediately applicable to another. Even for the same application, the user interface often evolves between versions and may be inconsistent across different operating systems (e.g., Office 2003 vs 2007, Office on OS X vs Windows). This variety of interfaces limits the reach of online tutorials.

In this paper we present *UI translation interfaces* that enable users to translate commands in one application to a different version of the application or to another applications in the same domain (Figure 1). We focus on the domain of image manipulation applications, as such applications are complex and their user interfaces differ widely. Our end-user tool, *ShowMeHow*, allows users to to locate commands in GIMP using the interface language of Photoshop.

ShowMeHow introduces two novel *query interactions*: First, users can search for interface commands in one application through *direct manipulation of an interface façade* of another

UIST'11, October 16-19, 2011, Santa Barbara, CA, USA.

application. Interface façades replicate the look of an application (i.e., its menus, toolbars, dialogs) and allow users to leverage their visual memory. We implement façades using screen capture images. Second, users can perform *text searches for interface commands* in one application using command names and keyboard shortcuts in another application. These techniques are based on *interface translation maps*: data structures that relate semantically equivalent actions in different user interfaces. Maps are based on correspondences between pairs of *interface models* which capture the hierarchical structure, visual appearance, and names of UI elements. Maps are currently authored manually; we discuss strategies for automating and crowdsourcing the map construction process.

We conduct two studies showing that ShowMeHow increases success in locating commands across different applications. In the first study, twelve users see 20 Photoshop commands and locate equivalent commands in GIMP, with and without the use of ShowMeHow. When using ShowMeHow UI façades, participants are twice as fast ($\mu = 26$ seconds vs $\mu = 52$ seconds, $p < 0.0001$). However, text search is not significantly faster than the control condition ($\mu = 49$ seconds). Twelve additional participants, all GIMP novices, apply tutorials written for Photoshop within GIMP, with and without ShowMeHow. Participants complete more subtasks with ShowMeHow and rate the tool highly for ease of use, speed and utility as a learning aid.

While our focus is on increasing the utility of tutorials, UI translation techniques have broader applicability: they may effectively scaffold the learning of new, unfamiliar applications by providing access to the interface of a familiar application as necessary. We conclude with a discussion of directions for future work.

RELATED WORK

A sizable body of work exists on the design of application documentation [2], online help [19, 13, 28], software tutorials [17, 10], and application learnability [11].

Early work pointed out the failures of lengthy manuals – users don't read them – and argued for task-centered learning [4]. Tutorials are effective because they teach techniques in the context of a concrete task. While most tutorials are created manually, it is possible to automatically generate visual step-by-step instructions by recording actions users take in a UI [10]. Graphical histories for image editing show operation histories so users can understand how images came to be, and reapply these histories in new contexts [21, 12, 23].

Noting that readers have trouble interpreting and following tutorials, researchers have proposed interfaces that walk users through operations step-by-step [3], highlight relevant operations with stencils [17], or programmatically interpret and execute how-to instructions [22].

At a lower level, users need to locate and understand individual commands. Various query mechanisms have been proposed to do so, e.g., a natural language interface for UNIX commands [30]. Mac OS X 10.5 introduced a menu search that programmatically opens menu trees and shows the user where a command is located in the tree. But these techniques cannot translate between different interfaces.

Little work has addressed the problem of transferring knowledge between user interfaces. Davis et al. [6] studied behavioral differences between *initial* and *subsequent* users of an application, but do not explore the design space of tools to enable transfer learning. One solution is to change interface appearance: the open-source *GIMPShop* project modifies the GIMP menu structure and application vocabulary to mirror Photoshop's UI [1]. GIMPShop required editing the application source code. ShowMeHow enables users to translate commands without modifying the applications.

The ShowMeHow approach is most closely related to the site-to-service map and introduced by d.mix [14]. In both approaches, a correspondence map between semantically equivalent actions in two different interfaces is used. d.mix enables programmers to use direct manipulation on web pages to synthesize equivalent web service API calls. ShowMeHow uses direct manipulation actions in one graphical interface as query input to retrieve equivalent actions in a second direct manipulation interface.

UI translation maps also enable retargeting. Bricolage [20] identifies equivalent DOM elements in web pages to apply the look and feel of existing pages to new content. UNIFORM can adapt user interfaces to match other interfaces a user is already familiar with [24]. While Bricolage employs machine learning to identify matches, UNIFORM uses string and variable type matching combined with manual editing. ShowMeHow follows UNIFORM's semi-automated approach.

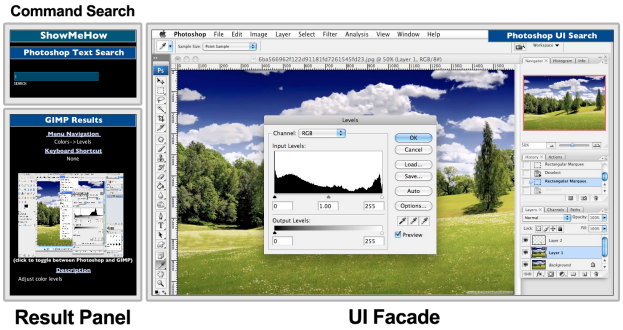

Figure 2: The ShowMeHow user interface offers a UI façade that enables users to locate commands with direct manipulation. Translated commands are shown in the left panel.

SCENARIO: TRANSLATING WITH SHOWMEHOW

Kyle's company is cutting its software budget and migrating from Photoshop to the free GIMP application. He is creating a donation web site for tsunami victims and consults an online tutorial for creating text embedded in the crest of a wave. He starts by creating text with a drop shadow. In Photoshop, this command can be invoked through the LAYER–STYLE–DROPSHADOW menu. Kyle pulls up the ShowMeHow façade and navigates to the submenu (Figure 2, 3A). The ShowMeHow sidebar indicates that the equivalent GIMP command is FILTERS–LIGHTANDSHADOW–DROPSHADOW menu, under the LIGHT AND SHADOW submenu (Figure 3B).

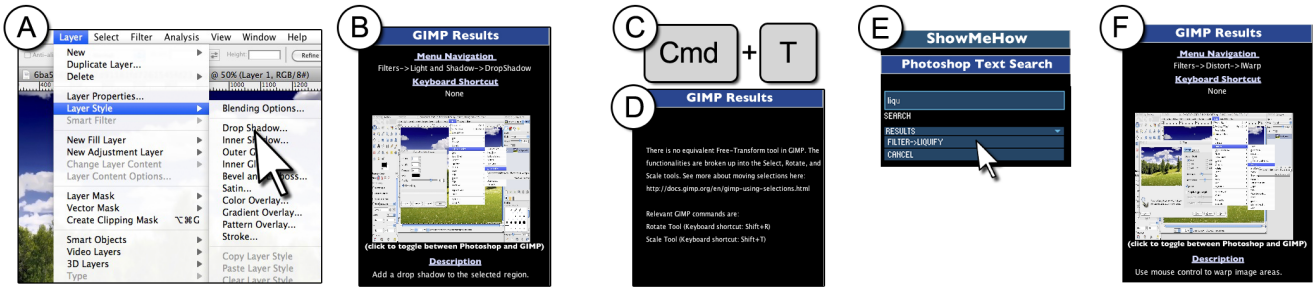

Figure 3: Users locate corresponding commands through manipulation of an interface façade (A,B); keyboard shortcuts (C,D); or text search for command names (E,F).

Kyle switches to GIMP and performs this action. Since Kyle is an accomplished Photoshop user, he performs many operations through keyboard shortcuts. To rotate and translate the text he created, he presses Cmd+T in ShowMeHow (Figure 3C) – this is the keyboard shortcut to perform a free transform operation in Photoshop. ShowMeHow indicates that there is no direct match for this command in GIMP but it lists multiple possible alternatives that can be used to achieve individual aspects of free transform (Figure 3D). Kyle next wants to distort the text. He types in "Liquid" into the ShowMeHow search box as he does not remember the precise name or location of the filter he wants to use. As he types, search results update incrementally and he discovers the FILTER-LIQUIFY command (Figure 3E). Clicking on the result yields the GIMP command FILTERS–DISTORT–IWARP (Figure 3F). With a couple of clicks, Kyle has successfully performed his tasks in GIMP with the help of ShowMeHow.

IMPLEMENTATION

Translating between different user interface actions requires the following fundamental building blocks:

- An *interface model* that describes the structure of the user interface for each application.
- A *translation map* that describes correspondences between elements of two models.
- A *query interaction* in which users select actions from the source interface to find corresponding actions in the target interface.

Interface Models

In order to translate commands, a tool must first have a list of available commands. ShowMeHow captures this information in an interface model. Users invoke commands through different interface widgets: menus, toolbars, palettes, and dialogs. Widgets can be referenced in different ways: by name, by picture, or by keyboard shortcuts. In addition, widgets may not be visible on screen at all times (e.g., submenus, palettes). Therefore, a description of the path required to make an action visible may be needed. Our model uses a tree representation: each node is an action, described by name, image, and optional accelerators. The model does not encode command semantics (what commands mean, or what code they execute).

Many interfaces today are built using declarative specifications (e.g., HTML, XAML) — these source files can be used to build interface models. In *model-based UI systems*, designers author an abstract model, and the system generates a concrete UI (e.g., [9, 26, 25]). Such systems can also provide the interface model for free. However, at least some of the applications we wish to support are closed source and not model-based. Given a concrete application, how might we derive a suitable descriptive model of it?

Much of the needed information is available programmatically — through accessibility APIs, UI framework inspection methods, or interface functions. In our example, both Photoshop and GIMP can generate complete command lists (through EDIT–KEYBOARD SHORTCUTS–SUMMARIZE and HELP–PROCEDURE BROWSER, respectively). For Photoshop, we read these lists with a custom parser, then manually augment them with images of command widgets.

Translation Maps

The translation map relates pairs of interface models. Our maps support three types of entries: 1 : 1 correspondences between commands, *alternatives* where one command has multiple matches, and *workarounds* that use plain text to capture instructions if no direct match is available. Mappings are expressed in an XML file format that relates model elements (Figure 4). Correspondences where one command is equivalent to a sequence of steps can be expressed in the map format, but are not yet supported in the ShowMeHow interface.

Correspondences express semantic equivalence of actions. Establishing equivalence may require human judgment. Thus fully automatic map construction may not be possible. We create maps manually but also examine automatic and crowdsourcing techniques to accelerate map creation.

Automation through String Matching Automatic techniques can be used to seed a map and to aid users in establishing additional correspondences manually, e.g., through suggestion interfaces. Matching command names is a promising first step [24], as commands with similar functionality are likely to have similar names. For different versions of the

```
<map-direct ps_key="menu_image_rotate_arbitrary"
  gimp_key="toolbar_rotate" />
<map-many ps_key="edit_freetransform"
  xtra_desc="There is no equivalent Free-Transform tool.
  See more about moving selections here: http://...">
    <gimp_key id="gimp_toolbar_rotate.jpg"/>
    <gimp_key id="gimp_toolbar_scale"/>
</map-many>
<map-missing ps_key="layer_style"
  info="There is no layer styling in GIMP.
  Some effects can be achieved through filters." />
```

Figure 4: Examples of map entries.

	Matches Found	% of PS commands	False Positives	Accuracy
String matching	133	19.9%	17	87.2%
MTurk, ≥ 2 workers in agreement	93	13.9%	25	73.1%
Not found by string matching	66	9.9%	24	63.6%
MTurk, single confident worker	99	14.8%	48	51.5%
Not found by string matching	72	10.7%	41	43.0%

Figure 5: Results from constructing mapping tables automatically and through crowdsourcing on Mechanical Turk.

	Photoshop CS3	GIMP 2.6
Correct string match	Image,Adjustments, Color Balance...	Colors, Color Balance...
Incorrect string match	View,Show,Grid	Filters,Render,Pattern,Grid...
Correct manual match	Filter,Last Filter	Filters,Repeat Last
Incorrect manual match	Filter,Sketch,Torn Edges...	Filters,Edge-Detect,Edge...

Figure 6: Example map entries generated through string matching, and on Mechanical Turk.

same application, many commands and their position in the interface hierarchy will remain identical. To understand how interfaces change across versions, we ran Chawathe's change detection algorithm [5] on XML trees of two Photoshop versions – CS and CS3. 72% of commands remained identical.

When translating between different applications in the same genre (e.g., Photoshop and GIMP), different naming schemes and UI organization reduce the number of tree correspondences. To find matches across applications, our string matching algorithm performs a case-insensitive search over leaf command names, ignoring ancestors and omitting special characters. As a benchmark, we compared menus and tools in Photoshop CS3 (670 entries) and GIMP 2.6 (502 entries). 19.9% of Photoshop command names matched GIMP names, with 12.8% false positives (see Figure 5 for statistics and Figure 6 for examples of correct and incorrect matches).

Crowdsourcing the Map Semantically identical commands can be described in syntactically different ways, so string matching has limited reach. To find additional equivalences, we investigated whether non-expert users can match commands. This process lends itself to crowdsourcing as it is *embarrassingly parallel*: map entries are independent of each other and can be provided by different users. ShowMeHow's Web-based authoring tool presents users with a single command in a source application and asks them to identify a match in a target application (Figure 7). We deployed this tool on Amazon's Mechanical Turk labor market, paying $0.11 each for 1000 mappings from Photoshop to GIMP. A total of 941 jobs for 513 distinct Photoshop commands were completed by 45 workers. We analyzed mappings where

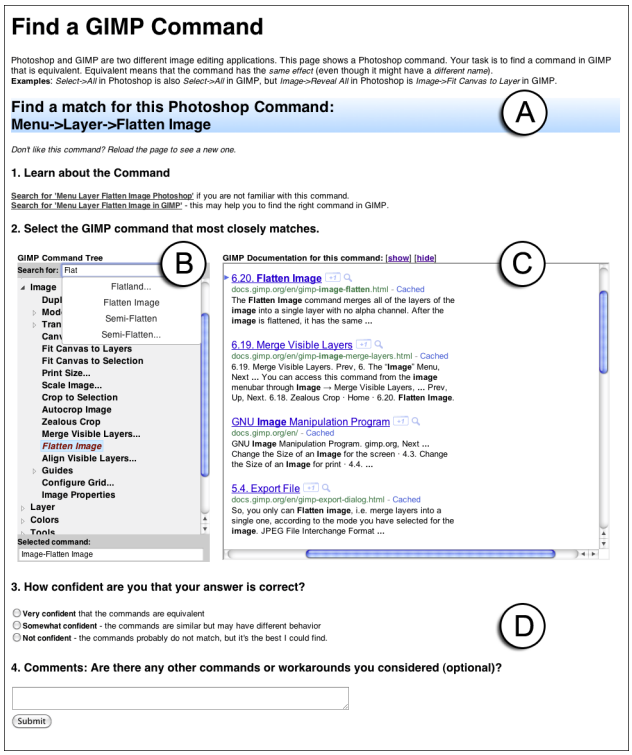

Figure 7: Users author correspondences in a Web interface. The UI shows a source command (A) and asks users to find an equivalent target command (B). Incremental auto-completion shows potential matches. Automatic search over application documentation helps users to verify chosen commands (C). Users also indicate their confidence (D) and workaround information if no match exists.

multiple workers agreed on the target command and mappings where workers indicated high confidence in their answer (see Figure 5). The results suggest that non-expert workers can add a significant number of useful entries not found by string matching, but that additional verification (e.g., voting) is necessary to limit false positives.

To quantify the cost of building a map by experts, we also hired three freelancers with Photoshop experience on oDesk. At $5-$15 per hour, these workers created 31-41 mappings per hour, leading to a price per map entry of $0.12-$0.41. The best worker had 94% accuracy on mappings where she indicated high confidence in her answer.

Query Interaction

Query interaction techniques have two phases: query specification, and translation result display. Queries can be specified through interface façades and text search.

Building Interface Façades Façades enable users to leverage visual memory; this is especially valuable for commands invoked through tool icons. Interface façades should let users explore the user interface as if the façade were the real application — to the extent necessary so the user can locate commands. This implies that menus should unroll; toolbar buttons should expand; dialogs should appear at the appropriate times.

To achieve these goals, we execute each command and screen capture the entire application. To add interactivity, different images have to be hyperlinked to each other — specifically, nodes in the UI model have to link to their children. We use image maps to express these links: polygons in image coordinates that describe both the location of *hot spot* areas and the a target state in the UI model (i.e., the identifier of an XML node). We created a tool to generate these image maps: the tool presents the user with a captured application image, and prompts the user to draw a bounding box for a given child command.

At runtime, façades implement a variation of *event bubbling*: the façade keeps track of the last active node. When the user clicks, the façade hit-tests cursor locations against all hot spots on the current node. If a hot spot is hit, the façade navigates to the correct child node in the interface tree. If no hot spot is hit, the façade bubbles the event upwards in the tree, repeating hit tests on each parent until the root. This algorithm allow users to jump between different tree levels.

Text Search: Commands and Shortcuts ShowMeHow provides a text search box with incremental search. As characters are entered, ShowMeHow searches the source application's UI model and shows all matches in a result list. ShowMeHow currently uses regular expression matching to find results. Clicking on any of the results is equivalent to selecting a command in the façade — it updates the façade view appropriately.

Expert users commonly rely on keyboard shortcuts for frequently used commands. To handle keyboard shortcuts, ShowMeHow intercepts keystrokes outside the search box and finds equivalent accelerators in the UI model. Again, to provide appropriate feedback, the façade view is updated to the corresponding model state whenever keyboard shortcuts are received.

Presenting Translated Commands ShowMeHow provides realtime translation correspondence back to the user. The result panel displays how to achieve the equivalent command in GIMP. It provides menu traversal instructions, a keyboard shortcut, a description of the current state, and a thumbnail of a GIMP screenshot corresponding to the current façade state. To see results in more detail, the thumbnail can be clicked to toggle the main display between the Photoshop façade and a full-size version of the GIMP screenshot.

USER EXPERIENCES WITH SHOWMEHOW

Our evaluation of ShowMeHow seeks to answer the following questions:

- *Utility:* Given instructions for a source application, are users able to operate a target application effectively with the aid of ShowMeHow?
- *Productivity:* Can we quantify the productivity gain of using ShowMeHow?
- *Usability:* What are the unique benefits and shortcomings of the different interaction techniques to access the translation map?

We conducted two user studies with 12 participants each, all recruited from UC Berkeley. Participants were required to have some expertise in Photoshop, but little or no expertise with GIMP. Participants self-rated their Photoshop proficiency on a scale from 1 (Novice) to 5 (Expert). For study 1, $\mu = 3.2$, $\sigma = 0.7$; for study 2, $\mu = 3.0$, $\sigma = 0.7$.

Study 1: Measuring Performance

In the first study, 12 participants were asked to translate individual commands from Photoshop to GIMP. Participants saw the name of the command and a screenshot of it. In our within-subjects design, participants completed half of the translations with ShowMeHow, and half without. Participants performed 20 translations of commands in Photoshop that existed in different UI locations in GIMP. For example, the FILE–OPEN action is the same in both interfaces, and thus was *not* used. Chosen commands included the MAGNETIC LASSO (maps to INTELLIGENT SCISSORS) and IMAGE– ADJUSTMENTS–LEVELS (maps to COLORS–LEVELS).

In each task we gave participants a Photoshop command and asked them to identify and execute the equivalent command in GIMP. Users alternated between using ShowMeHow and the control (no ShowMeHow) every 5 tasks. Start conditions were counterbalanced. We measured the time taken to successfully identify the command in GIMP and, for ShowMeHow trials, the type of query mechanism used (façade, text or shortcut). Time was measured from the moment of viewing the command to the moment when the user located the command in GIMP and hovered. The experimenter did not suggest which interaction method to use. If a participant had not identified the correct command after 90 seconds, the task was terminated and counted as a failure. Incorrect command choices were counted as false attempts.

Performance Results

Users identified GIMP commands more quickly when using ShowMeHow (see Figure 8). Without ShowMeHow, users took $\mu = 52$ seconds to successfully identify a command in GIMP. With ShowMeHow's Photoshop UI façade, users were twice as fast, taking $\mu = 26$ seconds to find a task. (two-sample $t(216) = 6.64$, $p < 0.0001$). When using ShowMeHow's text search, users averaged 49.7 seconds to find a task (not significant, two-sample $t(137) = 0.31$, $p > 0.05$). We hypothesize that the slower time for text search was due to engineering issues; some users remarked that the text search was slow and only searched for exact substring matches.

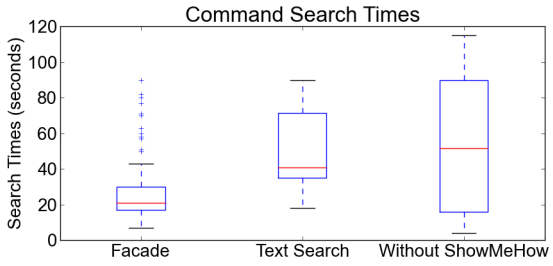

Figure 8: UI façades are significantly faster than text search and aid users in finding commands faster.

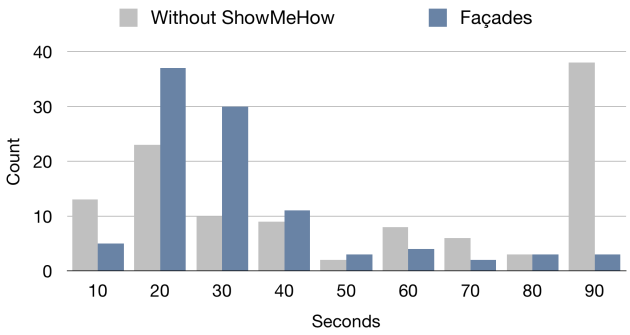

Figure 9: Histogram of task times in the control and façade conditions. Façades have a greater minimum cost; but without Façades, users are often lost.

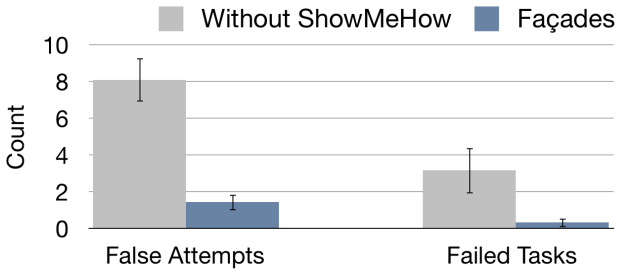

Figure 10: Participants had more false attempts and misses without ShowMeHow. Error bars show 95% CI.

Figure 9 compares search time histograms for ShowMeHow façades and the control condition. ShowMeHow has a larger minimum task time, because of the extra interaction steps and application switching. Without ShowMeHow, some commands are found quickly, but users are often lost if they cannot find the command right away.

ShowMeHow also led to fewer false attempts ($\mu = 1.4$ vs 8.1, two-tailed $t(22) = 3.40$, $p < 0.01$) and fewer failed tasks ($\mu = 0.3$ vs 3.2 per participant, two-tailed $t(22) = 4.19$, $p < 0.001$, see Figure 10). For example, when attempting to replace a range of colors in an image with another (Photoshop: IMAGE–ADJUSTMENTS–REPLACE COLOR, GIMP: COLORS–MAP–ROTATE COLORS), users without ShowMeHow would tend to incorrectly identify commands such as COLORIZE or COLORIFY as equivalents. Participants still made some false attempts with ShowMeHow; we believe these are due to participants forgetting query results while switching between ShowMeHow and GIMP.

Study 2: Translating Entire Tutorials

In the second study, participants completed a multi-step image manipulation tutorial written for a familiar interface (Photoshop) within an unfamiliar interface (GIMP). Participants performed this task twice, with two different tutorials: once with the aid of ShowMeHow and once without. Condition order and assignment of tutorial to condition were counterbalanced. Tutorials were taken from public web sites. Afterwards, participants completed a survey with Likert-scale and free-response questions.

Experimenters gave participants a brief overview of GIMP

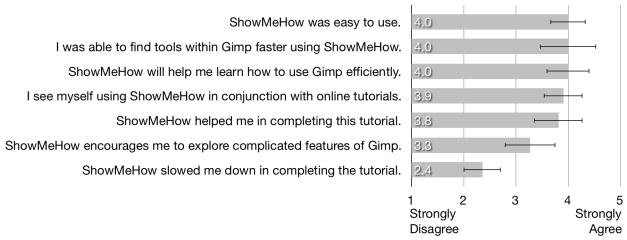

Figure 11: Likert scale ratings for ShowMeHow in study 2 Ratings range from 1:Strongly disagree to 5:Strongly agree. Error bars show 95% CI.

that introduced major tools and fundamental differences from Photoshop. Participants had 20 minutes for each tutorial and were stopped if the tutorial was not completed at this point. The tutorials comprised 9 and 12 steps; each step had 2 to 3 subtasks. Participants were encouraged to move on to the next subtask if they were unable to complete a subtask.

Participants were able to accomplish 3.6 more subtasks with ShowMeHow than without for the first tutorial. However, we observed that participants used ShowMeHow to find only the most difficult commands; many opted to directly struggle with the GIMP UI for commands they perceived as easier to find. Actively encouraging use of ShowMeHow by detecting when users "fumble" is an area for future work. On average, a participant looked up 6.74 commands in ShowMeHow. We also noticed that participants tended to strongly prefer one query technique — though there was no clear favorite across users. Only 2 participants preferred using keyboard shortcuts over the other methods while 5 participants each favored either the mock interface or text search.

In our post-test survey Likert questions, participants gave the highest marks to ShowMeHow for three dimensions: ease of use, speedup, and helping them learn how to use GIMP (Figure 11). Users felt that ShowMeHow helped them and made it easier to accomplish tasks in GIMP, even though our quantitative analysis did not show a statistically significant difference in time. One user remarked that "it was quite clear what I had to do in order to make something work in GIMP" and another that "ShowMeHow was really helpful for quick [lookups]". Participants were less enthusiastic about using ShowMeHow as a tool to openly explore how the two interfaces differed: it's main benefit is to quickly look up commands on demand, then disappear.

Participants also suggested usability improvements: search should cover keywords and related words, as web search engines do; our implementation only searched for exact command names. This limitation was partially ameliorated by the display of incremental results, which allowed participants to refine their search as they saw possible results.

We found that differences in parameters and parameter presets caused the biggest amount of confusion for our participants once they located the correct command. This suggests mapping command parameters in future work.

LIMITATIONS

There are several limitations of our approach that we plan to address in future work.

Multi-Step Translations Not Supported

Our current approach only handles single-step correspondences: one command in the source interface corresponds to one command (or a set of alternative commands) in the target interface. This limitation is primarily due to the chosen query interaction techniques. It is straightforward to describe multi-step sequences in the translation map. However, the current query interactions do not lend themselves to expressing multi-step queries: because the façade is not a functional interface, it is awkward to perform multiple steps in a row as the user cannot see the result of any one step.

Command Parameters Not Supported

ShowMeHow matches high-level commands, but does not handle parameters for those commands. However, most commands are parameterized. For example, the Rotate Canvas dialog takes an angle parameter It is possible to extend the current map syntax to cover parameters, but this would significantly increase the effort needed to capture correspondences. When parameters do not map directly onto each other, human expertise will be needed. We also observed that participants in our study tended to copy values from the tutorials directly into the application. If parameters do not match between programs, this practice yielded unexpected results.

User Must Switch Applications Frequently

Once a user has found a translated action, she still has to switch to the target application and perform the command displayed by ShowMeHow. This leaves a sizable *gulf of execution* [15]: the user may forget the precise menu path or shortcut during the application switch, especially if both applications cannot be shown side-by-side. Such short-term memory problems occurred repeatedly in our evaluation. They stem from the decision to implement ShowMeHow as an independent application. Integrating ShowMeHow into a target application could enable directly showing and executing actions in context. However, some authoring tools do not permit extensions to receive UI events. Accessibility APIs can, but many complex UIs do not use platform widgets. Pixel-based reverse engineering techniques like Prefab [7] might be able to extract event data at the appropriate level.

Façades Only Offer Partial Functionality

Users expect façades to act like the real application. Our Photoshop façade implements basic interactions with menus and toolbars. However, interactions that depend on application state are not yet supported. For example, users might try to add a new layer and then access layer-specific commands in context menus. Since our façade does not model application state, we are unable to translate such actions.

Users May Not Learn Target Interface

One unintended side-effect observed in our evaluations is that once users have located the appropriate command in the target application, they often proceeded to take the shortest possible path to execute that action — copying the keyboard shortcut. However, the shortcut does not help users understand how to locate the action inside the UI. Users who take advantage of the menu traversal instructions reap additional learning benefits. Unintended side-effects caused by assistive interfaces have been observed elsewhere and termed the "paradox of the assisted user" [29].

FUTURE WORK

Future work should investigate alternative approaches to generating user interface translation maps as well as additional interaction techniques.

Automate Map Construction

Two avenues for automation appear feasible. First, it may be possible to learn correspondences from pairs of demonstrations of the same task in different UIs. Application instrumentation can be be used to record what users do; in fact, some applications already have recording facilities for user-defined macros. Second, it may be possible to mine Web pages for correspondences. Command names appear on official documentation sites, blogs and forums. Correspondences be established from pages with similar features. Query-feature graphs [8] which relate application functions to search queries are a first step in this direction.

Further inspiration for automating map creation can be taken from analogous problems in *multi-version program analysis* in software engineering [18] and *schema matching* in database systems [27]. In particular, Potluck shows that better interaction techniques can reduce the cost of authoring equivalences [16].

New Query Interactions: Translate Tutorials

If ShowMeHow is predominantly used to translate tutorials found on the Web, the translation functionality could be built directly into the Web browser. We envision that ShowMeHow could automatically parse a loaded tutorial in the browser, find command names contained in the tutorial, and annotate these commands with possible translations. Such an annotated tutorial would obviate the need for an extra translation application, but poses the additional challenge of robustly identifying commands in unstructured text.

CONCLUSION

This paper introduced *UI translation interfaces*. These interfaces can help users adapt tutorials from one application version to another, and between different applications within the same genre. Translation interfaces can also help users leverage existing knowledge when users change software versions. UI translation interfaces are based on interface models and maps which relate pairs of models. We introduced *interface façades* which model the appearance, but not the functionality, of an application. Façades require more effort to construct, but yield larger gains. In our study, façades effectively reduced the time it took users to locate commands in an unfamiliar interface and eased the frustration of dealing with complicated, foreign interfaces. ShowMeHow is a first step in studying the translation of user interface actions between applications in a similar genre.

ACKNOWLEDGMENTS

We thank Intel and Google for their support. This research was supported by NSF grant CCF-0643552

REFERENCES

1. GIMPshop. http://www.gimpshop.com/. Retrieved 4/2011.
2. Ron Baecker. Showing instead of telling. In *Proceedings of the 20th annual international conference on Computer documentation*, SIGDOC '02, pages 10–16, New York, NY, USA, 2002. ACM.

3. Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. Docwizards: a system for authoring follow-me documentation wizards. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 191–200, New York, NY, USA, 2005. ACM.
4. John M. Carroll. *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. MIT Press, Cambridge, MA, USA, 1990.
5. Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. *SIGMOD Record*, 25:493–504, June 1996.
6. Sid Davis and Susan Wiedenbeck. The mediating effects of intrinsic motivation, ease of use and usefulness perceptions on performance in first-time and subsequent computer users. *Interacting with Computers*, 13(5):549 – 580, 2001.
7. Morgan Dixon and James Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1525–1534, New York, NY, USA, 2010. ACM.
8. Adam Fourney, Richard Mann, and Michael Terry. Queryfeature graphs:bridging user vocabulary and system functionality. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, New York, NY, USA, 2011. ACM.
9. Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12-13):910–950, 2010.
10. Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration. *ACM Transactions on Graphics*, 28:66:1–66:9, July 2009.
11. Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 649–658, New York, NY, USA, 2009. ACM.
12. Tovi Grossman, Justin Matejka, and George Fitzmaurice. Chronicle: capture, exploration, and playback of document workflow histories. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 143–152, New York, NY, USA, 2010. ACM.
13. Susan M. Harrison. A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 82–89, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
14. Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. Programming by a sample: rapidly creating web applications with d.mix. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 241–250, New York, NY, USA, 2007. ACM.
15. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. *Human-Computer Interaction*, 1:311–338, 1985.
16. David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: semi-ontology alignment for casual users. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ISWC'07/ASWC'07, pages 903–910, Berlin, Heidelberg, 2007. Springer-Verlag.
17. Caitlin Kelleher and Randy Pausch. Stencils-based tutorials: design and evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 541–550, New York, NY, USA, 2005. ACM.
18. Miryung Kim and David Notkin. Program element matching for multi-version program analyses. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 58–64, New York, NY, USA, 2006. ACM.
19. Kevin Knabe. Apple guide: a case study in user-aided design of online help. In *Conference companion on Human factors in computing systems*, CHI '95, pages 286–287, New York, NY, USA, 1995. ACM.
20. Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R. Klemmer. Bricolage: Example-based retargeting for web design. In *Proceedings of the 29th international conference on Human factors in computing systems*, CHI '11, New York, NY, USA, 2011. ACM.
21. David Kurlander and Steven Feiner. A history-based macro by example system. In *Proceedings of the 5th annual ACM symposium on User interface software and technology*, UIST '92, pages 99–106, New York, NY, USA, 1992. ACM.
22. Tessa Lau, Clemens Drews, and Jeffrey Nichols. Interpreting written how-to instructions. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1433–1438, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
23. Toshio Nakamura and Takeo Igarashi. An application-independent system for visualizing user operation history. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 23–32, New York, NY, USA, 2008. ACM.
24. Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. Uniform: automatically generating consistent remote control user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 611–620, New York, NY, USA, 2006. ACM.
25. Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK, 1st edition, 1999.
26. Angel R. Puerta. A model-based interface development environment. *IEEE Software*, 14:40–47, July 1997.
27. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, December 2001.
28. Christian Spannagel, Raimund Girwidz, Herbert Löthe, Andreas Zendler, and Ulrik Schroeder. Animated demonstrations and training wheels interfaces in a complex learning environment. *Interacting with Computers*, 20:97–111, January 2008.
29. Christof C. van Nimwegen, Daniel D. Burgos, Herre H. van Oostendorp, and Hermina H. J. M. Schijf. The paradox of the assisted user: guidance can be counterproductive. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 917–926, New York, NY, USA, 2006. ACM.
30. Robert Wilensky, Yigal Arens, and David Chin. Talking to unix in english: an overview of uc. *Communications of the ACM*, 27:574–593, June 1984.