WikiDo

Nate Kushman Micah Brodsky S.R.K. Branavan Dina Katabi Regina Barzilay Martin Rinard
Massachusetts Institute of Technology

***Abstract*— The Internet has allowed collaboration on an unprecedented scale. Wikipedia, Luis Von Ahn's ESP game, and reCAPTCHA have proven that tasks typically performed by expensive in-house or outsourced teams can instead be delegated to the mass of Internet computer users. These success stories show the opportunity for crowd-sourcing other tasks, such as allowing computer users to help each other answer questions like "How do I make my computer do X?". Such a system would reduce IT cost, user frustration, and machine downtime. The current approach to crowd-sourcing IT tasks, however, only allows users to collaborate on generating text. Anyone who goes through the process of searching help wikis and user forums hoping to find a solution for some computer problem knows the inefficacy and the frustration accompanying such a process. Text is ambiguous and often incomplete, particularly when written by non-experts. This paper presents WikiDo, a system that enables the mass of non-expert users to help each other answer *how-to* computer questions by actually performing the task rather than documenting its solution.**

1. Introduction

As computers have gotten faster, cheaper, and more functional, they have also gotten much more complicated, leading to a situation in which the productivity bottleneck is not the computer, but the users themselves and their ability to effectively use the computer system. Non-expert users regularly encounter tasks that they do not know how to perform such as configuring their home router, removing a virus, or even just emailing a photo. Many users do not have technical support, and hence their first, and often only, resort is a web search. Such searches, however, often lead to a disparate set of user forums written in ambiguous language. They rarely make clear which user configurations are covered by a particular solution; descriptions of different problems overlap; and many documents contain conjectured solutions that may not work. The net result is that users spend a lot of time manually working through large collections of documents trying solutions that often fail to help them perform their task. The situation is particularly difficult for non-expert users who often struggle even with well written documentation.

The main limitation of existing on-line technical help is that it only documents how a task was performed. What a typical user really wants is a system that automatically performs the task for him, taking into account his machine configuration and global preferences, and asking the user only for information which cannot be automatically pulled from his computer, like his password or which photo he would like to e-mail. One-off automation tools exist for particularly arduous tasks such as the Mail Merge wizard in Microsoft Word. Today, however, these automation scripts have to be meticulously programmed by experts, limiting their applicability to only those tasks most in need of automation. For example, Microsoft has recently started a team to automate the solution documents on their knowledge-base web site [7]. Each of these automated scripts is a program handwritten by an expert. As a result, in 6 months of their work, they have automated only about 150 of the hundreds of thousands of knowledge-base articles [9]. Expert-based automation is slow and expensive and hence unlikely to cover the majority of problems that users encounter.

This paper introduces WikiDo, a system that enables the mass of lay users to collaborate on automating computer tasks, instead of just documenting how to perform them. WikiDo aims to build a database of automated solutions for every important computer task. The key characteristic of WikiDo is that a user contributes to this database by simply performing the task. WikiDo records the graphical user interface (GUI) actions as the user performs the task. It aggregates multiple such GUI traces into a canonical sequence of GUI actions parametrized by user-environment, that will successfully accomplish the task on a variety of different configurations.

When a user comes across a task they do not know how to perform, they search the WikiDo database, which takes advantage of the current user's GUI context to try to find the right solution. Users can then either read a text version of the GUI actions, use the solution as a tutorial that will walk them through how to perform the task step by step, or allow the solution to run automatically on their computer to perform the desired task. Furthermore, WikiDo can take a restore point in case the user is unhappy with the result of the solution. Note that while WikiDo solutions can be executed automatically, since they are a sequence of GUI actions, they are also understandable by end users allowing them to be audited for malicious behavior just as easily as current text solutions found on-line.

To scalably build such a database with the help of non-expert users, WikiDo tackles two important challenges.

(1) Merging GUI traces while handling mistakes and configuration differences: Since WikiDo obtains solutions from non-expert users, we expect many of these solutions to contain mistakes and redundant actions. Naively filtering out actions that are not common across all traces, however, will remove not only mistakes and redundancies, but also any actions that are specific to a particular user configuration. WikiDo's approach to distinguishing these two cases is inspired by the write-coalescing of log-structured file-systems [19]. WikiDo models the state of the system and tracks which of the differing actions contributes to the final committed state and which get aborted or over-written. Action differences which do not affect the final state are

filtered out as mistakes, while the remaining differences are retained and used to parametrize the branches of the automated playback. The playback algorithm then chooses among the retained branches based on the system state and user inputs.

(2) Translating the existing corpus of textual information: To reach the point where the WikiDo database contains most common tasks, WikiDo should take advantage of the preexisting corpus of on-line help articles, like the ones available in the Microsoft Knowledge Base [8]. Ideally, these on-line resources could be translated to GUI actions fully automatically. Existing machine learning (ML) translation algorithms, however, cannot yet handle such tasks sufficiently well. Even very recent research that focuses directly on translating Microsoft Knowledge Base articles to GUI actions is able to correctly translate only 37% of articles [13]. To deal with this inaccuracy, WikiDo employs crowd-sourcing to boost the power of the ML translator. WikiDo runs the translator as a sub-component, but identifies the sentences on which it fails by using an ML classifier that takes advantage of both the state of the system and the linguistic information in the document. It then gives these incorrectly translated sentences to humans, and asks the humans to execute them as GUI actions in a virtual machine (VM). This approach allows WikiDo to maximize the leverage it can get from the crowd-sourced translations by focusing them on the difficult sentences, and translating the easier ones automatically.

To give a sense of the potential of WikiDo we present preliminary results from our prototype implementation.

We tested the automation capability of WikiDo on 5 computer tasks, completed by 12 Computer Science students. Even with detailed instructions, the students failed to correctly complete the task in 20% of the cases. For each task, we used WikiDo to aggregate the 12 executions into a canonical automated solution. For all 5 tasks, WikiDo produced correct canonical solutions which play back successfully on a variety of different machines.

We also tested WikiDo's ability to combine machine learning translation with crowd-sourced translation. Given 120 documents from the Microsoft Knowledge-base [8], a state of the art ML translator was able to correctly translate only 37% of these documents. WikiDo was able to boost the translator performance to 100% by asking users to demonstrate only 9-10% of the actions.

2. Illustrative Example

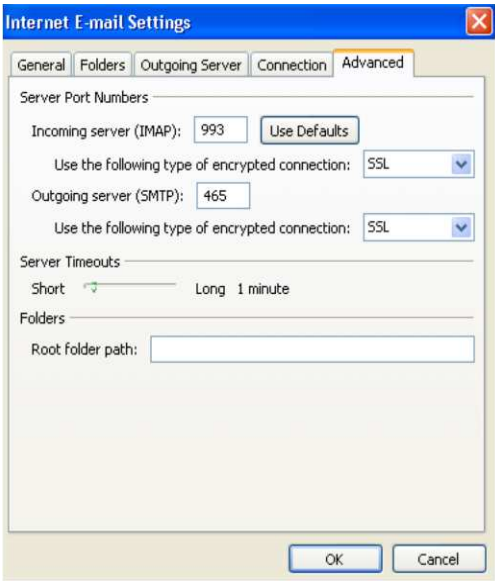
A system that aims to automate computer tasks based on user executions needs to attend to many subtleties. We illustrate these subtleties by considering the task of configuring a new IMAP account in Microsoft Outlook, which is documented in Fig. 1a.

(a) Different configurations require different GUI actions: When started, Outlook launches a different window depending on whether a prior account has been configured. As a result, different users need to perform different GUI

Configure your Outlook IMAP client

1. Start Outlook for the first time, or go to: **Tools→Account Settings→New→IMAP→Next**.
2. At **Auto Account Setup**, choose **Manually** .
3. Click **Next** twice.
4. Enter your name and your email address and set: **Account Type**=IMAP .
5. Choose your Incoming and Outgoing mail servers and enter your password.
6. Click **More Settings** and click the **Outgoing Server** tab.
7. Check the check box for authentication and the second radio button and enter your user name and password.
8. And on the **Advanced** tab, change both **None** drop-downs to **SSL**, then change **25** to **465**.

(a) Help Document



(b) Step 8

Fig. 1. **Documentation for configuring IMAP for Outlook clients.**

actions while performing this task. WikiDo must recognize these differing sequences of GUI actions as valid execution branches, and distinguish them from differences caused by user mistakes.

(b) Users make mistakes even with detailed directions: When we asked students to follow the instructions in Fig. 1a, we find that many students repeatedly failed to correctly perform step 8, which is shown in Fig. 1b. The port numbers are on the left side of the dialog box and so users tend to change them first and then change the drop-downs, even though the directions specify the opposite order. Unfortunately, when the drop-downs are changed Outlook automatically resets the port numbers. Users that fail to notice this eventually find that Outlook cannot connect with the mail server, but they are given little information on what they did wrong. While some users repeat the mistake on successive attempts and eventually give up, others eventually recognize their mistake and correct it. To take advantage of user traces with mistakes like this, WikiDo needs a mechanism to detect

mistakes and remove them from the canonicalized trace.

(c) Some actions cannot be automated: This includes actions which require a user to enter their name, password, or other user-specific inputs. WikiDo can avoid leaking passwords since the GUI naturally obfuscates these entries, providing WikiDo a simple accurate heuristic to avoid recording passwords. However, WikiDo needs to recognize which actions require user input and mark them as such in the automated solution.

(d) Machine learning-based translation cannot cope with implicit directions: We have also tried to automate the text in Fig. 1a using a state-of-the-art machine learning algorithm that maps text to GUI actions [13]. Unfortunately, some steps were left out of the documentation, as is often the case, because they are obvious to humans. For example, after step 8 it is clear to the human user that he must click OK to finish the task.In contrast, the ML translator stalls at this point, leaving the dialog box open and the task incomplete. While one could tell the ML algorithm about this particular case, it is difficult to address the general problem of implicit commands.

3. WikiDo

The main goal of WikiDo is to build a database of computer task automations, based on crowd-sourced help from a community of Internet users. Users contribute by simply performing the necessary GUI actions, either remotely in a virtual machine (VM) running on the WikiDo server, or by downloading a client which records the actions performed locally on their own machine. In either case, WikiDo currently uses the Microsoft Active Accessibility interface [6] to record GUI actions. Developed to enable accessibility aids for users with impaired vision, the accessibility interface has been built into all versions of the Windows platform since Windows 98 and is now widely supported [6]. Apple's OS X already provides a similar accessibility framework [4], and the Linux community is working to standardize on a single accessibility interface as well [1]. The accessibility interface allows WikiDo to both record and playback GUI actions without any modification to the operating system, and allows it to work seamless for both desktop and web-based applications.

There are two ways for users to contribute to the database. First, they can simply perform some task that they already know how to perform, then upload their GUI recording to the WikiDo server. Alternatively, they can contribute by helping to translate existing text documents. In this case, WikiDo will give the user a sentence to translate within an existing VM running on its servers. The user then simply performs the actions described in the text sentence. The rest of this section describes in detail how WikiDo takes advantage of these two types of contributions by both merging multiple noisy GUI traces for the same database entry into a single clean canonical trace, and by translating entire text databases into GUI action traces with as few user translations as possible.

It is important to recognize that WikiDo's automated solutions are intended to be best-effort. Thus, WikiDo takes a Microsoft System Restore Point before automatically performing any task, and immediately rolls back if the user does not confirm that the task was successfully performed. This user feedback is also used to inform the translation and merging process.

3.1. A Canonical Solution From Multiple GUI Traces

For each task, WikiDo collects GUI traces from multiple users and aggregates them into a canonical GUI script that, when executed on a user machine, can automatically perform the corresponding task. Generating a canonical solution requires: (1) filtering out user mistakes and irrelevant actions; (2) identifying user and environment specific GUI actions so that they can be made into parameters in the canonical solution. WikiDo performs these transformations by converting the concrete GUI actions into a sequence of abstract actions on which the transformations are performed.

3.1.1) Abstract Model: In the abstract model all actions are performed on *widgets*, which are each a part of a *view*. A widget could be a text-box, a button, etc, while a view is identified by all the widgets contained in the GUI window in which an action is performed. For example, in Fig. 1b, the tab labeled Advanced is a particular view in that window and the button labeled User Defaults is a widget in that view.

There are three types of abstract actions in WikiDo's model:

Update-Actions: Update-Actions create a pending change to the system state. Examples of Update-Actions include editing the state of an existing widget, such as typing into a text box or checking a check-box, and adding or removing entries in the system state, e.g. an operation which adds or removes an item from a list-box.

Commit/Abort-Actions: These actions cause pending changes made by Update-Actions to be written back into the system state. A typical example of a Commit-Action is pressing the OK button, which commits all changes in all widgets in the corresponding window. An Abort-Action is the opposite, and aborts any pending state changes in the corresponding window, e.g., pressing a Cancel button.

Navigate-Actions: These change the set of available widgets by changing the current view. Examples of Navigate-Actions include opening a dialog box, selecting a different tab in a tabbed window, or going to the next step of a wizard by pressing the Next button.

Note that a single concrete GUI action may be converted into multiple abstract actions. For example, pressing the OK button both commits the pending states in the corresponding window and navigates to a new view.

3.1.2) Filter Mistakes: WikiDo first filters out mistakes in an individual abstract action trace. For example, a user may open a given dialog box, type a value into an edit box, and then close the dialog box only to realize he made a mistake. He therefore opens the dialog box again to change the value. The goal is to remove all actions from the trace

associated with the first opening of the dialog box. A three pass algorithm achieves this goal as follows:

(a) Filtering Out Update-Actions: The first pass processes the trace *backward* to remove all Update-Actions on a particular widget except the last Update-Action that actually gets committed. To do this, the algorithm walks backwards through the trace maintaining two sets of widgets: a pending set and a committed set. When the algorithm sees a Commit-Action for some widget(s), it checks if the widget is already in the committed set, and if not, it adds it to the pending set. When the algorithm reaches an Update-Action, it checks whether the widget is pending and if so, it moves it to the committed set. If the widget is not pending, it removes that Update-Action from the trace. At the end of the backward pass, there is at most one Update-Action for each widget, and each Update-Action is followed by a Commit-Action for that widget. There may also be redundant Commit-Actions with no Update-Action. We will remove those in the next step.

(b) Filtering Out Commit-Actions and Abort-Actions: The previous step may leave Commit-Actions that no longer have associated Update-Actions. To remove these the algorithm walks *forward* through the trace maintaining a set of pending widgets. When it reaches an Update-Action, it adds the affected widget to the pending set. When it reaches a Commit-Action, it checks whether the affected widget is pending, and if so it removes it from the pending set. Otherwise, it removes the Commit-Action from the trace.

(c) Filtering Out Navigate-Actions: The trace may also contain unnecessary Navigate-Actions which appear either after removing unnecessary Update-Actions and Commit-Actions, or just because of isolated user navigation mistakes (e.g., clicking the wrong tab). These unnecessary Navigate-Actions are removed by removing any action sequence containing only Navigation-Actions as long as we can navigate immediately from the available views before the sequence to the next view after the sequence. We determine if such a navigation is possible by observing whether we have ever seen such a navigation in a trace.

3.1.3) Parametrize Update-Actions: Now that we removed mistakes, differences in user inputs reflect machine configuration specific or user specific information. For each widget in each view, WikiDo parses all traces to find all unique values that were given to that widget via Update-Actions that were subsequently committed. Based on these values the associated Update-Actions are marked as either *AutoEnter* if the associated widget is assigned the same value in all traces, or *UserEnter* if the associated widget is assigned a different value in each trace. On play back, AutoEnter updates are performed automatically while WikiDo will stop play back and ask the user for all UserEnter actions. UserEnter is typically used for updates to widgets containing information like username and password which cannot be automated. Note that if the widget is assigned to a few different values, many of which occur in multiple traces (e.g., a printer name), WikiDo will assign it *PossibleAutoEnter*, and on play back let the user select one of the previously entered values or enter a new value.

3.1.4) Parametrize Navigate-Actions:: When two different traces follow different navigation paths, i.e. they visit a different set of views, WikiDo is forced to decide if they differ because the user made a mistake in one of the traces, or simply because differences in the user's configuration require a different set of steps to perform the same action.

WikiDo distinguishes these two cases by recognizing that if the users' underlying configuration causes the difference, then the users' GUI actions will be exactly the same up until the point of divergence, but the resulting views will be different. For example, performing the same set of actions to open Outlook will bring up two different windows depending on whether or not the user has already configured an account. In cases like this, WikiDo will automatically generate a separate branch in its resulting canonical trace with the branch point parametrized by how the underlying system reacts, i.e. which window Outlook displays. This ensures that even when differences in the underlying system create the need for different navigation paths, WikiDo can still automatically execute the solution without needing help from the user. If the users actually perform different actions even though the underlying system reacts exactly the same way, then these are typically mistakes, which would be removed by our filtering algorithm above. If they still exist after filtering, however, WikiDo chooses the path that is the most common among the different traces. Note that if there are multiple ways to complete the same task, this will merely pick the most common way.

3.2. Document Translation via Machine Learning

WikiDo initially builds its database by automatic translating existing natural language (English) documents to a sequence of GUI actions. Given a sentence, e.g. "point to OK", this process translates it into a GUI command, e.g. `LEFT_CLICK`, and an object in the environment on which to perform that command, e.g. `BUTTON:OK`. This problem has been recently addressed in the machine learning community [13]. Our goal is to utilize such an ML translator to bootstrap WikiDo's database by automatically translating existing on-line help databases.

We must address two problems, however, in order to achieve this goal. First, getting experts to correctly translate documents is expensive, and without any examples of correctly translated documents, the state of the art ML translator can correctly translate only 37% of the documents in the Microsoft Knowledge Base [13]. WikiDo addresses this issue by using crowd-sourcing as a proxy for expert annotations. It merges multiple action sequences to filter out idiosyncrasies and mistakes of individual solutions, providing a new inexpensive approach for annotating data for this ML task, in much the same way as the ESP game annotates images for machine vision tasks [21].

The second problem is that even when human experts fully annotate a sizable corpus of randomly selected documents, the ML translator correctly translates only 52% of

the documents in the Knowledge Base [13]. Thus, simply replacing expert annotation with crowd-sourced annotation will not enable WikiDo to translate all documents in the Knowledge Base and similar databases. Thus, instead of randomly choosing documents for human translation, WikiDo asks for crowd-sourced translation of only the sentences on which the ML translator produces incorrect translations. To accurately detect such sentences, WikiDo utilizes the ML classifier described below.

The WikiDo translator operates in an iterative fashion, interleaving two steps. First, starting with the output translations of the baseline ML translator [13], WikiDo's classifier predicts which sentences have been erroneously translated. Next, WikiDo provides humans with a few of the incorrectly translated sentences and a connection to an associated VM snapshot running on WikiDo's servers, and asks them to perform the actions described in the sentence. Multiple such translations are merged into a single canonical translation using the method described in the previous subsection. The second step of this process is to retrain the baseline ML translator with the newly obtained human translations. Thus the human effort is used both to directly translate difficult sentences, and to train the baseline ML translator to improve its accuracy. This iterative process continues until all the action translations are predicted to be correct.

A critical component of this process is a classifier which can accurately detect sentences which have been translated incorrectly. To do this with high accuracy, WikiDo takes advantage of the fact that detecting sentences which are difficult to translate is much easier than translating them correctly. For example, the classifier can recognize that long sentences with many uncommon words are much more likely to be translated incorrectly than short sentences with common words like "click" and "OK". Also, if the ML translator had many other alternate translations which also had a reasonable match to the words, then it's more likely the translation was incorrect, and if many words in the sentence are left unmapped to an action, then it's likely an action was missed. These features and others from the sentences, the environment and the translated GUI sequences are combined together using a support vector machine (SVM), a state of the art learning algorithm [17]. WikiDo trains the SVM using the translations obtained by crowd-sourcing.

4. Results

We report two preliminary sets of results showing WikiDo's potential, one set for the merging task, and one for the translation task.

4.1. Trace Merging

We evaluate WikiDo's ability to aggregate multiple solutions using the 5 tasks in Table 1. To perform our evaluation, we ask 12 computer science students to perform each of these 5 tasks based on the instructions from our lab website. We then utilize a prototype of the WikiDo aggregator

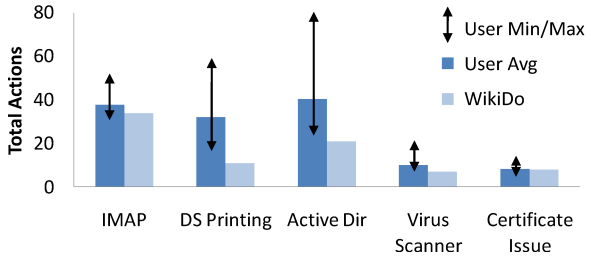


Fig. 2. **Filtering Results:** This shows that the canonicalized WikiDo trace is always on par with the best user trace and occasionally significantly better.

IMAP	Add a new IMAP account to Outlook
DS Printing	Configure the local printer to print double sided
Active Dir	Join the computer to the lab's Active Directory domain
Virus Scanner	Install a virus scanner
Certificate	Change a Firefox configuration setting related to certificates

Table 1
Aggregation Tasks

described in §3.1 to merge the recorded GUI traces from the 12 users into a single canonical trace.

We find three important results. First and foremost, we find that for all 5 tasks, WikiDo's canonical trace plays back successfully, performing the required task on a set of 6 different personal machines from other members of our lab. Second, the quality of WikiDo's filtering is shown in Figure 2. The figure shows that WikiDo's removal of many unnecessary actions results in traces that are much shorter than those from the average user, and about as short or shorter than those from even the most efficient users. Despite removing many unnecessary actions, WikiDo's playback success shows that it never removed any *necessary* actions. Lastly, we also find that WikiDo's parametrization also works quite well. Approximately 90% of the actions in the canonical traces are marked as AutoEnter and thus require no user interaction. This reduced the average number of actions a user needs to perform from 25 down to 2. Additionally, 92% of the remaining UserEnter actions require entering personal information such as usernames or passwords which cannot be generically automated. Thus, WikiDo's parametrization ensures that as much as possible, the resulting traces can be replayed automatically with little input from the end user.

4.2. Document Translation

We evaluate WikiDo's ability to translate text documents into GUI actions by utilizing 120 documents pulled from the Microsoft Knowledge Base. Representative of typical Knowledge Base articles, these documents contain an average of 8-9 actions each. We implemented both the iterative translator, and the classifier that identifies incorrectly translated actions, but we have not yet fully integrated them. Therefore, we report the performance of each component in isolation.

Our first evaluation effort considers WikiDo's iterative translation when selection of which actions to give to human translators is driven by an oracle. As additional

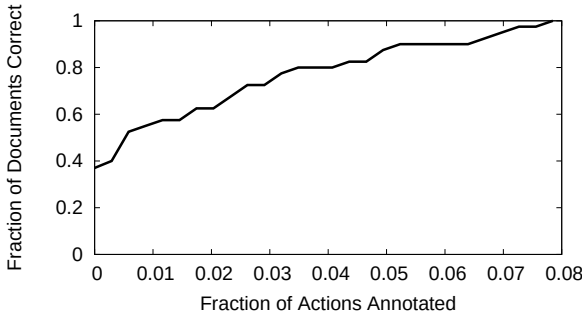


Fig. 3. **WikiDo's Translation Performance as a Function of the Annotated Actions:** After asking users to translate only 7.5% of actions, WikiDo can successfully translate all documents.

	Actual Correct	Actual Wrong
Predicted Corr.	24354 (94%)	1459 (6%)
Predicted Wrong	783 (12%)	5661 (88%)

Table 2
Classifier Performance

human translations are provided, we measure the fraction of correctly translated documents. Figure 3 shows that WikiDo's translator is quite effective, and can successfully translate all the documents correctly by requesting human translation of less than 8% of the total actions.

Our second evaluation focuses on the accuracy of the SVM action classifier described in §3.2 which will be used to replace the oracle once we integrate it into our iterative translator. To train the classifier, we run our iterative translator on the 120 Knowledge Base documents. We then manually compare these translations with the correct translations to mark each one as correct or incorrect. We then split the translations such that those for 35 of the documents are placed in a training set and the translations for the other 85 are placed in a test set. The performance of the classifier on the translations for the 85 test documents is shown in Table 2. The figure shows that if the classifier predicts that a given action was translated correctly, then 94% of the time the action actually is correct. This ensures that WikiDo's resulting translations are usually correct. Secondly, if WikiDo's classifier predicts that an action was translated incorrectly then only 12% of the time was the action actually translated correctly. This ensures that WikiDo does not request too many unnecessary translations. Specifically, we can assume that this inaccuracy will increase the percentage of human translations needed from 8% to more like 9-10% of actions.

5. Related Work

WikiDo is motivated by prior work on crowd-sourcing [10], [21], [5]. Prior work on crowd-sourcing of how-to computer tasks, however, has limited users' contributions to text and graphics. In contrast, the key idea underlying WikiDo is to allow users to contribute by simply performing a computer task rather than documenting in text how one could potentially perform the task.

WikiDo is also inspired by the concept of Programming by Example (PBE) [14], [12], [18], [11], however these approaches rely on application-specific APIs, and do not take advantage of crowd-sourcing.

While there are many tools to help automate computer tasks, most either do not support recording and must be scripted by programmers [3], [2], or allow recording only by relying on application specific APIs, or low-level screen coordinates and thus cannot be used to robustly automate generic computer tasks. Apple's Automator is the only tool which provides both, as far as we know, but it provides no mechanism to collaborate and automatically produce a canonical GUI trace which works under multiple different machine configurations.

Lastly, there are also tools that exploit persistent state information (e.g. file-system state) from a large user population [15], [22], [16], [20]. By nature of their design, they cannot handle how-to type tasks, configuration tasks that involve a server machine, or any task with user specific information. In contrast, since WikiDo works at the GUI level it can handle not just local configuration tasks, but any general task the user can perform.

In contrast to all of this work, WikiDo is the only application-independent tool which allows crowd-sourced automation of how-to type computer tasks.

6. Acknowledgments

This work was supported by NSF grant IIS-0835652.

References

[1] Accesibility Toolkit (ATK). `http://accessibility.kde.org/developer/atk.php`.
[2] AutoHotkey. `http://www.autohotkey.com/`.
[3] AutoIt, a freeware Windows automation language. `http://www.autoitscript.com/`.
[4] Mac OS X Accessibility Framework. "`http://developer.apple.com/documentation/Accessibility/Conceptual/AccessibilityMacOSX/AccessibilityMacOSX.pdf`".
[5] Mechanical Turk. `http://www.mturk.com`.
[6] Microsoft Active Accessibility. `http://en.wikipedia.org/wiki/Microsoft_Active_Accessibility`.
[7] Microsoft Fixit. `http://support.microsoft.com/fixit`.
[8] Microsoft Knowledge Base. `http://support.microsoft.com`.
[9] Security Garden Blog. `http://securitygarden.blogspot.com/2009/04/microsoft-fix-it-gadget.html`.
[10] Wikipedia. `http://www.wikipedia.org`.
[11] L. D. Bergman, V.C., T.A.L., and D. O. DocWizards: A System For Authoring Follow-me Documentation Wizards. In *UIST*, 2005.
[12] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. Miller. Automation and customization of rendered web pages. In *UIST*, 2005.
[13] S. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.
[14] A. Cypher. *Watch What I Do: Programming by Demonstration*. MIT Press.
[15] Y.-M. W. et. al. Strider: A black-box, state-based approach to change and configuration management and support. In *LISA*, 2003.
[16] H.W., Y.H., C.Y., Z.Z., and Y.W. FTN: Towards Privacy-Preserving, Automatic Troubleshooting. In *IPTPS*, 2004.
[17] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, 1998.
[18] G. Little, T.L., A.C., J.L., E.H., and E.K. Koala: Capture, share, automate, personalize business processes on the web. In *CHI'07*.
[19] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *ACM TOCS*, 1992.
[20] Y. Su, M.A., and J. Flinn. Autobash: improving configuration management with operating system causality analysis. *SOSP*, 2007.
[21] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI*, 2004.
[22] H. J. Wang, J.P., Y.C., R.Z., and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI*, 2004.