Koala: Capture, Share, Automate, Personalize Business Processes on the Web

Greg Little[1], Tessa A. Lau[2], Allen Cypher[2], James Lin[2], Eben M. Haber[2], Eser Kandogan[2]

[1]MIT CSAIL
Building 32
32 Vassar St
Cambridge, MA 02139 USA
glittle@mit.edu

[2]IBM Almaden Research Center
650 Harry Rd
San Jose, CA 95120 USA
{tessalau, acypher, jameslin, ehaber, eser}@us.ibm.com

ABSTRACT

We present Koala, a system that enables users to capture, share, automate, and personalize business processes on the web. Koala is a collaborative programming-by-demonstration system that records, edits, and plays back user interactions as pseudo-natural language scripts that are both human- and machine-interpretable. Unlike previous programming by demonstration systems, Koala leverages *sloppy programming* that interprets pseudo-natural language instructions (as opposed to formal syntactic statements) in the context of a given web page's elements and actions. Koala scripts are automatically stored in the Koalescence wiki, where a community of users can share, run, and collaboratively develop their "how-to" knowledge. Koala also takes advantage of corporate and personal data stores to automatically generalize and instantiate user-specific data, so that scripts created by one user are automatically personalized for others. Our initial experiences suggest that Koala is surprisingly effective at interpreting instructions originally written for people.

Author Keywords

End-User Programming, Programming by Demonstration, Wiki, Automation.

ACM Classification Keywords

H5.2. Information Interfaces and Presentation – User Interfaces; D.2.6 Programming Environments

INTRODUCTION

Modern businesses are full of complex, idiosyncratic processes performed by people of various skill levels and knowledge. In particular, concerns over cost often force companies to adopt a self-service approach where employees perform their own travel arrangements, hiring, and purchasing, despite their lack of expertise in these processes. For example, engineers ordering a new monitor may be confused by arcane procurement terminology such as "Net parts Expense - Purchased from Vendor" versus "Purchased & Leased Equip. - PC/Workstation".

We have created a system called Koala to address this problem by allowing end users to document and record the specifics of performing a business process, and put the process on a wiki to share their experience and allow others to automatically execute the process for themselves.

Koala enables end users to record and automate their interactions within the Mozilla Firefox web browser. As users interact with the browser performing a process, Koala records all the forms filled, links and buttons clicked, and menus selected in a script. Instructions in the script are saved as pseudo-natural language text, which is not only easily readable, understandable, and editable by users but is also interpretable and executable by Koala (Figure 1). The Koala interpreter is sufficiently flexible that it is also surprisingly successful at interpreting and automatically executing instructions originally written for people. Koala scripts are saved to Koala's wiki, called *Koalescence*, so that users can take advantage of working scripts created by their colleagues. By making pseudo-natural language human-readable instructions available on a wiki, Koala makes it easy for co-workers to collaboratively edit and share scripts.

RELATED WORK

Koala builds on a number of emerging technologies in the web space. Greasemonkey [1] enables users to make client-side modifications to the appearance and behavior of web pages on their computer. However, creating a Greasemonkey script requires detailed knowledge of JavaScript programming to alter the DOM of the web page.

Chickenfoot [2] eases client-side customization by providing a higher-level API for accessing and manipulating common web page elements, using information in the rendered DOM. For example, the Chickenfoot instruction `click('search button')` will click a button with the text "search" on it. However, the Chickenfoot interface is

CHI 2007, April 28–May 3, 2007, San Jose, California, USA.

still very much a programming interface, in which users write syntactically correct statements in the Chickenfoot programming language.

The Keyword Commands paper [3] advocates lowering the language barrier even further by removing formal syntax altogether. We call this approach *sloppy programming*. However, practical implementations of this idea require some assumptions. The Keyword Commands algorithm can interpret expressions composed of keywords, such as *click search button*, but it assumes that most keywords will map to some function in the underlying scripting language. This does not scale well to the verbose textual inputs that are likely to appear in how-to documents written for humans. Koala leverages the sloppy programming approach in the web domain by taking advantage of the fact that most web commands are flat: there is one verb, and one or two arguments. This assumption dramatically simplifies the algorithm, and makes it more robust to extraneous words. It can handle long expressions originally intended for humans.

Koala exposes the synergy of a new paradigm in end-user programming by combining features from: 1) sloppy programming as a human- and machine-understandable script representation; 2) programming by demonstration (PBD) to record and play back user actions; 3) data stores to automatically personalize scripts; and 4) a wiki for sharing and collaborating on scripts. These features combine synergistically: the power of PBD is multiplied by human-readable recordings of user actions, which can be read and corrected if necessary by the user. It also lends itself well to being shared on wikis, thus extending the reach of PBD systems from single-user personal automation to multi-user collaborative organizational memory. Data stores contribute to this synergy, and address a significant problem in end-user programming, by effortlessly generalizing recorded scripts and personalizing them for subsequent users. Both sides of this process become transparent and intuitive through the use of human-readable scripts and a simple human-readable database.

A USE CASE: ORDERING OFFICE PRODUCTS

One of our first actual uses of Koala occurred when Tina wanted to order a type of pen through our company's online ordering system (BOND), but the pen was not listed in the online catalog. Tina didn't know how to proceed, so her colleague Edward emailed her step-by-step instructions, based on how he had ordered video cassettes which were not in the catalog. The following example is based on this use case.

Tina opens the Koala sidebar, and pastes in the instructions from the email from Edward and runs the script. Koala highlights the first step in red, which says `login to BOND`. The red means that Koala does not know how to perform the step, so Tina edits the script, which puts Koala into Record mode. When she selects the BOND website from her bookmarks, Koala records this in the script as: *go to https://www.buyondemand.com/*. Tina deletes the line *login to BOND* and runs the script again. Koala performs each step of the script automatically, pausing for each web page to load before continuing to the next step. Before executing each step, Koala highlights it in green, and visually flashes the corresponding button or input field on the web page so that the user can see what it is about to do.

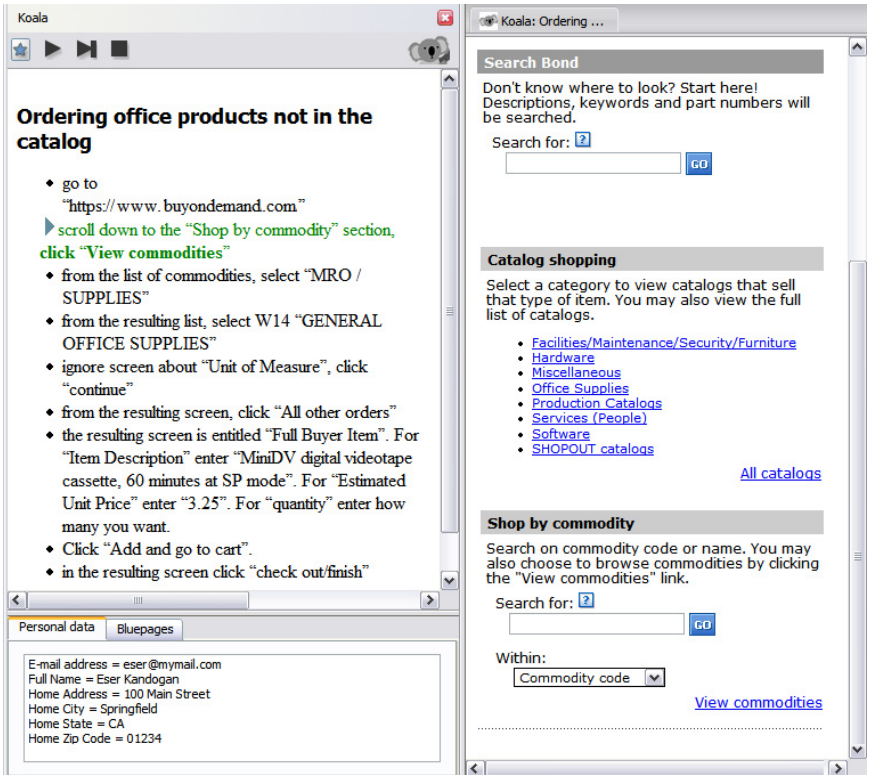

Figure 1. Ordering Office Products using Koala: Tina launches Koala, copies instructions from an email of a colleague, steps through the instructions, fixes steps manually, if necessary, to complete an order not in catalogue.

When Koala gets to the next-to-last step, it highlights that line in red and does not attempt to execute it. There are two reasons for this. Currently, Koala does not split multiple steps listed on a single line. More importantly, Koala does not attempt to execute any line containing the word "you". Instead, it waits for the user to perform the step manually. This is a simple and effective way to enable *mixed initiative* execution of scripts: the computer performs some steps and the user performs others, usually the steps that are

too complex to be worth automating. This is a staple of practical end-user programming [4].

When Tina has finished running the script, she clicks the Save button to store this script in the Koalescence wiki. She or her colleagues can edit it further, rate it, or comment on it. She can also email the script to her colleagues.

KOALA SLOPPY INTERPRETER

Koala utilizes a *sloppy programming* approach that interprets pseudo-natural language instructions, as opposed to formal programming language statements that must be syntactically correct. Each step in a sloppy program is processed by an interpreter that tries to evaluate the step in the context of a given web page's content, elements, and available actions. For instance, given the Google home page, and the slop *click search button,* the interpreter would propose to programmatically click the search button.

We describe the algorithm in three basic steps, using the example instruction *type Danny into first name field* on a simple web form (Figure 2).

First, the interpreter enumerates all the possible actions associated with various HTML objects in the document, such as links, buttons, text boxes, combo-box options, check boxes, and radio buttons. For each of these objects, the interpreter associates a variety of keywords, including the object's label, synonyms for the object's type, and action verbs commonly associated with the object. For instance, the *First name* field of a web form (Ⓐ in Figure 2) would be associated with the words *first* and *name*, because they are part of the label. It would also associate *textbox* and *field* as synonyms of the field's type. Finally, the interpreter would include verbs commonly associated with text fields such as *enter*, *type*, and *write*.

Next, the interpreter searches for the object that matches the greatest number of keywords with the slop. In the example, textbox Ⓐ would match the keywords *type*, *first*, *name* and *field*, with a score of 4. In contrast, textbox Ⓑ would only match the keywords *type* and *field*, with a score of 2.

Finally, the system may need to do some post-processing on the slop to extract arguments (e.g. *Danny)*. The key idea of this process is to split the slop into two parts, such that one part is still a good match to the keywords associated with the web object, and the second part is a good match for a string parameter using various heuristics (e.g. has quotes around it, or is followed/preceded by a preposition). In this example, the parts are: *type into first name field* (with 4 keyword matches), and *Danny* (which gets points for being followed by the preposition "into" in the original slop).

Note that our scripts consist only of human-readable text. We do not save interpretations or any other additional information with the script. This is a radical departure from previous PBD techniques but is surprisingly effective, since

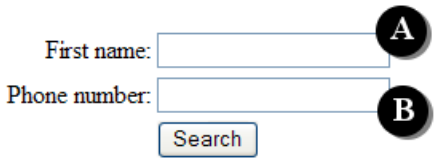

Figure 2. A simple web form.

a web page provides a very small search space. As with Chickenfoot [2], the use of textual labels to identify web page elements makes Koala scripts more robust to changes in the page, since we expect that developers will keep constant this proximate text-labeling, to avoid confusing their users. Furthermore, this representation facilitates wiki-style collaboration: users can simply edit the text document to change the script. Finally, users can also utilize existing how-to documents to import scripts into Koala, and have reasonably good success executing them.

Understanding and Correcting Koala

Because Koala's interpreter can sometimes be wrong, we have implemented several techniques to help the user know what the interpreter is doing and make corrections. We'll illustrate these techniques with the following example from Figure 1.

Suppose the user has selected the highlighted line: *scroll down to the "Shop by commodity" section, click "View commodities"*. Koala interprets this line as an instruction to click a link labeled *View commodities*. At this point, we want to make two things clear to the user: what is the interpreter planning to do, and why?

We show the user what the system is planning to do by placing a transparent green rectangle around the *View commodities* link, which is also scrolled into view.

We address the question of "why" by letting the user know which words in their slop lead the interpreter to its selection. In this case, the words *click*, *view* and *commodities* were associated with the link, so we make these words bold: *scroll down to the "Shop by commodity" section,* ***click "View commodities"***.

If the interpretation was wrong, the user can click the triangle to the left of the line, which expands a list of alternate interpretations. These interpretations are relatively unambiguous instructions generated by the interpreter:

- *click the "View commodities" link*
- *click the "View contracts" link*
- *click the "Skip to navigation" link*

When the user clicks on any of these lines, the system places a green rectangle over the corresponding HTML control. If the line is the correct interpretation, they can click the Run or Step button to execute it. If not, they may need to edit the line. Failing that, they can add the keyword *you* (e.g., ***you*** *click the "View commodities" link*) so that the interpreter leaves execution to the user.

DATA STORE — PERSONALIZING SCRIPTS

Business work frequently involves filling in forms with user-specific data. While Koala scripts are posted to Koalescence, so that users can take advantage of scripts created by their colleagues, it would be of little value to run them with the original author's data. Koala uses several sources of user-specific data to overcome this problem.

Koala includes a personal data store for each user, which is simply a text box containing name-value pairs (bottom left corner of Figure 1). Users can enter any data field they choose, and supply their personal value. During script recording, if the user fills in a form with a value that appears in the database, that step is automatically generalized to refer to the named attribute, rather than the current user's literal value. For example, a script step might be generalized to: *enter your home street address (e.g., 100 Main Street) into the "Address:" textbox*. Note that Koala also includes a sample value as part of the step, since we have found that specific examples help future users of the script understand the format to be used for that value. During playback, references to the personal data store are automatically filled in with the current user's values, thereby personalizing the script for this user.

Koala can also incorporate other data sources, such as a corporate directory containing data about employees, such as phone number, office number, and e-mail address. Koala automatically extracts all of this data about its current user, and makes it available to recorded scripts.

KOALESCENCE WIKI — SHARING "HOW TO"

The pseudo-natural language instructions in Koala scripts make wikis a natural choice for sharing "how to" knowledge. As scripts are readable even without the Koala extension, they are useful to many users.

The Koalescence wiki allows users to view existing scripts, along with ratings and information on the scripts (Figure 3).

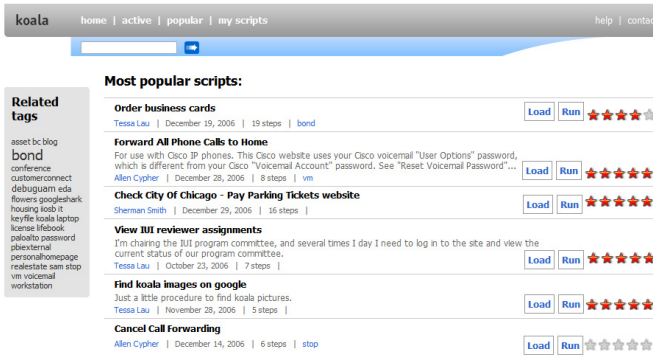

Figure 3. The Koalescence wiki allows users to search and view existing scripts with ratings and information, as well as editing scripts to provide additional information.

The wiki provides various ways to find a script, including a search utility, a popular script list, and tag clouds. Users can also view their scripts under "my scripts".

When viewed in the Koalescence wiki, a script is essentially a list of human-readable instructions. They can be duplicated to create another version or deleted, if obsolete. Users can also add tags, rate, or comment on the scripts to facilitate an effective sharing environment.

CONCLUSION AND FUTURE WORK

Koala allows users to capture, share, automate, and personalize business processes on the web. With Koala, the barriers to capturing "how to" knowledge are significantly lowered. Future work includes leveraging past contexts, adding data detectors, enriching the interface to support control flow, and performing extensive user studies to determine its value in a real-world setting.

REFERENCES

1. McFarlane, N. 2005. Fixing web sites with Greasemonkey. *Linux J.* 2005, 138 (Oct. 2005), 1.
2. Bolin, M., Webber, et. al. Automation and customization of rendered web pages. In *Proc. UIST 2005*, ACM Press (2005), 163–172.
3. Little, G., and Miller, R. C. Translating Keyword Commands into Executable Code. In *Proc. UIST 2006*, ACM Press (2006).
4. Horvitz, E. Principles of Mixed-Initiative User Interfaces. In *Proc. CHI '99*, ACM Press (1999). 159–166.

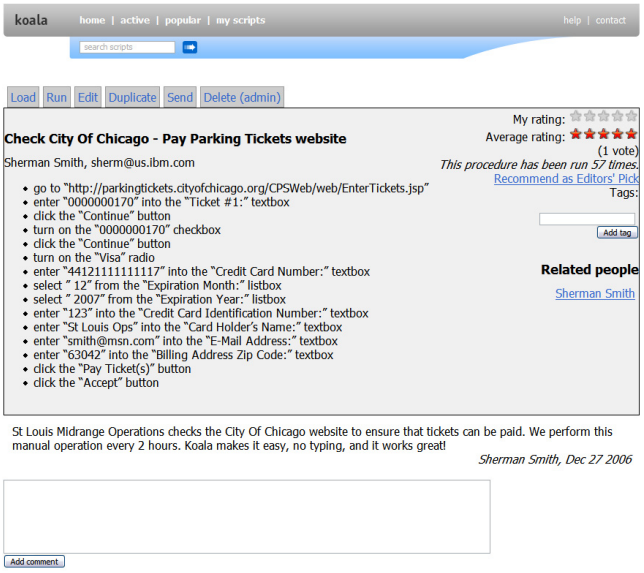

Figure 4. A script in the Koalescence Wiki displays human readable instructions. Users can load, edit, rate, and comment on the scripts.