Using Scenario-Based Requirements to Direct Research on Web Macro Tools

Christopher Scaffidi[1], Allen Cypher[2], Sebastian Elbaum[3], Andhy Koesnandar[3], Brad Myers[1]

[1]*School of Computer Science, Carnegie Mellon University, Pittsburgh, PA*

[2]*Almaden Research Center, IBM, Almaden, CA*

[3]*Computer Science and Engineering Department, University of Nebraska, Lincoln, NE*

Abstract

Web macros automate the interactions of end users with web sites and related information systems. Though web macro recorders and players have grown in sophistication over the past decade, these tools cannot yet meet many tasks that people perform in daily life. Based on observations of browser users, we have compiled ten scenarios describing tasks that users would benefit from automating. Our analysis of these scenarios yields specific requirements that web macro tools should support if those tools are to be applicable to these real-life tasks. Our set of requirements constitutes a benchmark for evaluating tools, which we demonstrate by evaluating the Robofox, CoScripter, and iMacros tools.

1. Introduction

Information workers, webmasters, and other people often need to perform repetitive tasks in a web browser, which is tedious and error-prone. In response, researchers and companies have provided numerous tools intended to help users automate web browser tasks [1][4][5][10][11][12][13][18][22]. Typically, these tools follow the programming-by-example (PBE) paradigm: a recorder watches the user perform operations, determines the user's intent, and generates a "web macro" to represent that intent (generally as a sequence of steps). Later, a macro player exe-

---

Portions of this article previously appeared as C. Scaffidi, A. Cypher, S. Elbaum, A. Koesnandar, and B. Myers. Scenario-Based Requirements for Web Macro Tools. In *VL/HCC'07: Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2007, 197-204.

cutes the macro on new data. Applying PBE to web browser automation has seemed promising, since the paradigm had previously been applied in other contexts, such as user interface design [14] and HyperCard programming [3].

Web macro tools offer several benefits. First, they offer significant time savings to users. In addition, when a procedure in a large web application is complex and hard-to-learn, then users who have mastered that difficult procedure can create a web macro as a teaching tool for other users, in order to encapsulate and communicate the steps required to perform the procedure. Finally, web application developers can use web macros to create automated test suites.

Given these promised benefits, it might seem that web macro tools should be in widespread use, particularly among information workers, whose work in web browsers is highly repetitive [21]. Moreover, users seem to be generally capable of understanding the process of recording and replaying macros, as 42% of information workers report that they or their subordinates recorded spreadsheet macros in the past 3 months, and 33% similarly report recording of word processor macros [20].

Despite these factors, web macro tools do not seem to be in widespread use. One reason is that the web context introduces new challenges that did not apply in traditional PBE. One such challenge is the frequent changes to web sites' structure, which can cause web macros to fail without warning. In addition, whereas many traditional macro tools only need to operate in one environment (such as HyperCard [3]), many office tasks that involve a web browser also involve other applications, such as spreadsheets. Automating these tasks requires a web macro tool to support inter-application integration.

The contribution of this paper is a methodical characterization of the requirements that web macro tools must support in order to be useful for many real-world tasks. An additional contribution is to demonstrate that these requirements serve as a helpful benchmark for evaluating tools and identifying beneficial areas of work. The requirements that we have identified include support for triggering macros, using objects on web pages, adapting to site changes, reading and writing data outside of pages, transforming data, executing control structures, recovering from failure, and supporting macro maintenance.

To help ensure the validity of these requirements, we base them on a range of real-world tasks that should ideally be automatable with web macros. We selected these tasks because automating them would offer clear benefits to end users. For example, automating one time-consuming task was so desirable to one end user that he paid a professional PHP/Perl programmer to automate the task; open source programmers automated two other tasks to help people.

Certain tasks are ones that we regularly perform, and we would like to automate these tasks to save ourselves time, but we have no suitable PBE macro tool. Finally, we have observed co-workers manually performing particular tasks, and automating these tasks would offer significant time savings.

We do not claim that our list of requirements is complete in the sense that satisfying them will necessarily make tools perfect for all imaginable tasks. Instead, by linking requirements to a diverse set of specific tasks, our benchmark indicates the wide range of real-world applicability that a tool would gain by satisfying certain requirements. In addition, the benchmark constitutes a seed that can grow as researchers contribute more scenarios where macros would be beneficial.

Section 2 presents Related Work. Section 3 uses a scenario format to describe tasks, and Section 4 analyzes scenarios to identify tool requirements. Section 5 demonstrates using requirements as a benchmark for evaluating the Robofox, CoScripter, and iMacros tools, thereby identifying areas for future work.

2. Related Work

Many papers use scenarios to motivate and explain a PBE tool's features [1][4][5][10][11][12][13][18][22]. Generally, a paper first presents the scenario in a succinct form to motivate the work; later, the paper describes the scenario in some additional detail and discusses how to use a new tool to automate the scenario.

For example, a paper on the Turquoise web macro tool presents a scenario of combining clippings from web sites into a newspaper, and the paper also discusses a scenario of repeatedly submitting a web form in order to purchase sandwiches [13]. As another example, a paper on the Creo tool describes a scenario of reading a recipe on a web site, then copying each ingredient into a web form on another site to compute the recipe's nutritional value [4].

Such scenarios meet the intended purpose of motivating and demonstrating a new tool. However, they have two limitations.

First, each paper generally only mentions one or two scenarios and rarely describes any scenario details that are unsupported by the tool. Thus, such scenarios rarely highlight opportunities for extending the tool, they provide limited support for evaluating future tools, and they offer no uniform way to compare the capabilities of different tools.

Second, the "pedigree" of scenarios is rarely documented: that is, it is usually unclear whether each scenario was identified by observations of end users or if it is hypothetical. Consequently, it is difficult to determine if supporting the scenario will make the tool useful in practice.

In this paper, we specifically select a variety of scenarios that highlight opportunities for future work. In addition, we have documented the source of each scenario, providing traceability to help ensure that automating each scenario would give real benefit to users.

The identified requirements comprise a benchmark to measure tool improvement. Our intent is similar to that behind the Test Suite for Programming by Demonstration [16], a benchmark for traditional (non-web) PBE tools. Like that Test Suite, our benchmark illustrates the wide range of potential applications for tools and enables researchers to test tools with real world tasks.

Our requirements can also be used to guide the design of future tools. In this sense, our intent is similar to that of user studies that have uncovered requirements for end-user programming tools other than web macro tools. Examples include a survey of programmers that influenced the design of the FlashLight tool for creating web applications [17], as well as field studies of system administrators that guided the design of the A1 tool for creating scripts that read and write data from servers [9]. Empirical studies have also preceded the design of programming tools for other particular populations such as children [15] and females [2]. These studies, like our work, are an application of the standard user-centered design process to the specific domain of end-user programming languages and environments.

3. Scenarios

Each of our scenarios represents a task that users would benefit from automating. Users perform a plethora of repetitive tasks, so selecting tasks for analysis requires applying a few judicious criteria, as follows:

First, our ultimate goal is to enable people to use web macros for real-life tasks. Consequently, we avoided presenting hypothetical scenarios and focused on real situations encountered by users in actual practice. These are not abstract situations, but rather "instantiated" tasks grounded in concrete user experiences.

Second, in order to provide a benchmark to measure improvements to macro tools, we have chosen scenarios that highlight challenges that PBE web macro tools have yet to address. Most scenarios describe repetitive tasks that users still must manually perform, though a few describe repetitive actions that have been automated with hand-coded scripts.

Last, in order to make this benchmark available to other researchers, we selected scenarios that we could publish online without violating privacy or security concerns. The scenarios and the resulting benchmark are posted on a

wiki[1], where other researchers can contribute and comment on scenarios. We hope that this benchmark will grow as macro tools meet existing requirements and other researchers provide new scenarios. The wiki also can serve as a mechanism for documenting any changes (evolution) in the websites involved in scenarios. We began collecting scenarios two years ago and have already seen some changes (as discussed further in Section 4.3).

3.1. Scenario Sources

To explore the breadth of macros' applicability, we have selected scenarios involving several types of users, including office workers, online shoppers, financial analysts, and IT staff. Moreover, these scenarios come from a variety of sources: contextual inquiry, co-workers, online sources, and our own experience.

Three scenarios were uncovered by our contextual inquiry of office workers [21]. We observed the work of three administrative assistants, four office managers, and three webmasters/graphic designers at Carnegie Mellon University. We watched each for one to three hours, in some cases spread over two days, and used a tape recorder and notebook to record information.

Two scenarios were performed by co-workers. We observed these tasks during the course of work and later realized that they were suitable for automation.

Three scenarios, which involved screen-scraping, were automated by people with scripts. In two cases, the programmer publicly posted the script (one PHP and one JavaScript); we have reverse-engineered these scripts into macro scenarios. In the third case, a financial analyst publicly posted a specification for the scenario (which probably was implemented by a professional programmer in Python, PHP or Perl); we have converted this specification into a scenario.

Two scenarios were performed by us in our role as end users. Like all researchers, we lead double lives. On one hand, we can program in various languages when needed. On the other hand, we are also end users. Constrained by time and interest, we often live within the confines of existing applications rather than write our own.

3.2. Structure of Scenarios

Each scenario describes not only a task, but also pre-conditions that must hold prior to the task as well as post-conditions that must hold after the task. To achieve the post-conditions, different tools may take different implemen-

[1] http://softwaresurvey.cs.cmu.edu/wmcorpus.html

tation approaches. For example, some macro players are agents that emulate a browser, while others are toolbars that manipulate the browser like a puppet.

Therefore, while we specify *what* scenario post-conditions must be satisfied, we do not specify *how* they must be satisfied. Indeed, we do not even stipulate what examples the macro recorders may request from users: if a recorder can do better by requiring more input, then that innovation represents a tradeoff worth considering. For example, some recorders use a pure PBE approach, while others allow users to augment the macro with procedural code [5].

As shown in Table 1, each scenario contains a number of sections in addition to pre-conditions and post-conditions. If other researchers follow this section structure when augmenting our wiki with new scenarios, then readers may be able to easily locate information. Extra sections might be added for specific scenarios to contain information that does not readily fit into any standard section.

Table 1: Sections contained by scenarios; required sections are asterisked.

*	*Title:*	This makes it easy for researchers to refer to scenarios.
*	*Typical User:*	This makes it easy for readers to find scenarios that might be performed by specific populations.
*	*Scenario Source:*	This summarizes the observations or other empirical data that generated this scenario. This information helps communicate the extent to which this scenario represents the real world rather than a hypothetical situation.
*	*Overview:*	This explains the context and motivations that would prompt a *Typical User* to perform the scenario.
*	*Starting Conditions:*	This identifies pre-conditions that hold true before the scenario.
*	*Result:*	This identifies post-conditions that should hold true after the scenario. This specifies the goal that a user or macro must achieve in order to be judged capable of performing this scenario.
*	*Actions:*	This describes the algorithm, process, or steps that the user performs (or that the user performs with the help of a tool). This is *not* to suggest that macros must perform the same algorithm. A macro need only achieve the *Result* in order to be considered successful.
	Action Details:	This could include screenshots, snippets of HTML, video, or other pieces of information that make the A*ctions* clearer.
	Variations:	This discusses ways in which users might tweak the *Actions, Starting Conditions,* or *Results* into a slightly different but similar scenario.
	Macro Maintenance:	This examines how the scenario might evolve over time, prompting changes to macros that attempt to automate the scenario.

3.3. Sample Scenario

The following scenario illustrates the structure and content of our scenarios.

Scenario Name: Per Diem Lookup

Typical User: office worker

Scenario Source: Several administrative assistants performed this during a contextual inquiry.

Overview

To file travel expense reports, office workers use an intranet application with fields for the date and locality of the travel (city and state). Using the date and locality, they must enter the federally-approved per diem allowance into another field.

They generally achieve this by popping open a new window (leaving the expense report form running in a window in the background), then navigating to a government web site to look up per diem rates. After using the site to look up the rate for that date and locality, workers copy and paste the result back into the expense report and close the popup window.

Starting Conditions

The user already has open an expense report web form containing the following data in text widgets:

- A date, in MM/DD/YYYY format
- A locality, in City, ST format

Result

The clipboard contains the per diem rate for the date/locality.

Actions

1. Open the per diem web site.
2. Select the year and click the image map link for the state.
3. If the desired city and date appear, then
    3.1. Retrieve the per diem rate and terminate.
4. Open the county lookup web site.
5. Submit the city name.
6. If the city's county appears, then
    6.1. If the county also appears on the per diem web site, then
        6.1.1. Retrieve the per diem rate and terminate.
7. Use the default value from the per diem web site (if Action 6 does not generate a rate).

Action Details

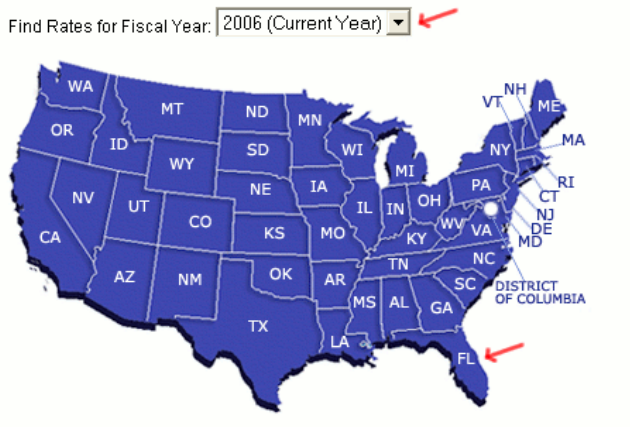
1 When we recorded this scenario, the web site was publicly accessible at

http://www.gsa.gov/Portal/gsa/ep/contentView.do?programId=9704&channelId=-15943&ooid=16365&contentId=17943&pageTypeId=8203&contentType=GSA_BASIC&programPage=%2Fep%2Fprogram%2FgsaBasic.jsp&P=MTT

The site has subsequently moved to

http://www.gsa.gov/Portal/gsa/ep/contentView.do?contentId=17943&contentType=GSA_BASIC

2 Select the year and click the image map link for the state.



The map has an image map; each link has an href like `javascript:setAction('Florida')`

3 Look for the appropriate per diem rate based on the city and the travel date. If the desired city and date appear, then retrieve the per diem rate and terminate. Otherwise, proceed to Action 4 to try looking up a value based on county name.

Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate
ALTAMONTE SPRINGS	SEMINOLE	67		43		110
BRADENTON	MANATEE	64		35		99
COCOA BEACH (October 1 - December 26)	BREVARD	83		39		122
COCOA BEACH (December 27 - September 30)	BREVARD	93		39		132
DAYTONA BEACH (February 1 - March 31)	VOLUSIA	134		43		177
DAYTONA BEACH (April 1 - January 31)	VOLUSIA	79		43		122

4 Sometimes the user can find a per diem rate for the county, even when the specific city is unavailable. However, the user rarely knows the county corresponding to the city. Fortunately, the per diem web site has a link to the "NACO" site, which helps users find the city's county.

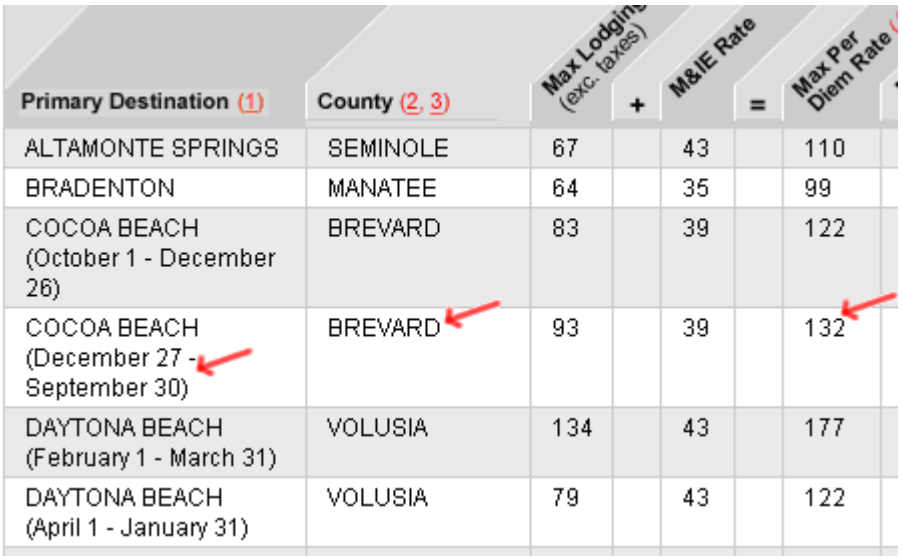
Cities not appearing below may be located within a county for which rates are listed. To determine what county a city is located in, click here for the National Association of Counties (NACO) website (a non-federal website).

5 The NACO web site offers several ways to look up a city's county. Perhaps the easiest is to type the city's name into the fourth form on the web page, shown below.

6.1 See if the city (with the correct state) appears on the NACO results. If so, then retrieve the county name and proceed to Action 6.2. Otherwise, go to Action 7 to compute a default per diem rate.

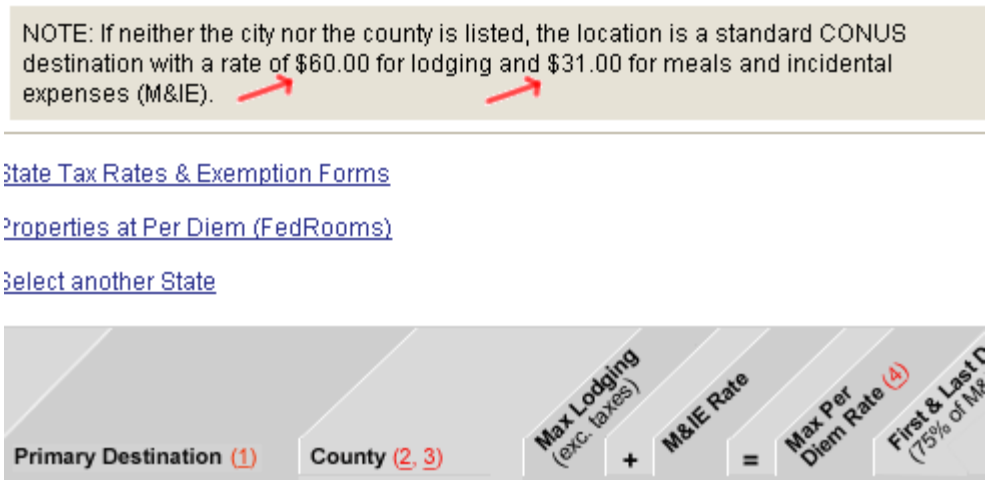
State	Place			County
AL	Rockledge			Etowah County
FL	Rockledge	city	Incorporated Area	Brevard County
GA	Rockledge			Laurens County
PA	Rockledge	borough	Incorporated Area	Montgomery County

6.1.1 If the county name was available, then the user can try to look up a per diem rate based on the travel date and county. If this is successful, then the scenario terminates. Otherwise, the user proceeds to Action 7.



Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate
ALTAMONTE SPRINGS	SEMINOLE	67		43		110
BRADENTON	MANATEE	64		35		99
COCOA BEACH (October 1 - December 26)	BREVARD	83		39		122
COCOA BEACH (December 27 - September 30)	BREVARD	93		39		132
DAYTONA BEACH (February 1 - March 31)	VOLUSIA	134		43		177
DAYTONA BEACH (April 1 - January 31)	VOLUSIA	79		43		122

7 The government site specifies a per diem rate to use as a default when the Actions above fail to generate a result. This appears in a grey background text box near the top of the page.



NOTE: If neither the city nor the county is listed, the location is a standard CONUS destination with a rate of $60.00 for lodging and $31.00 for meals and incidental expenses (M&IE).

State Tax Rates & Exemption Forms

Properties at Per Diem (FedRooms)

Select another State

Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate (4)	First & Last D (75% of M&

The default per diem rate actually is the sum of two numbers ($60.00 and $31.00 in the screenshot). These values vary from year to year, depending on what year selected in Action 2. Thus, the default per diem is not a constant, and determining the right value involves picking these two currency amounts out of the text and summing them.

Variations

Automating this with a macro might copy this value directly back to a widget in the expense report form rather than leaving it in the clipboard.

If the city is located outside of the United States, then these actions will not return a result. Internationally, entirely different rules apply (involving different web sites), so the macro player might show an alert and let the user take alternate actions.

Macro Maintenance

Although the government currently publicizes per diem rates at the URL shown above, they previously appeared at http://policyworks.gov/org/main/mt/homepage/mtt/perdiem/travel.htm

The HTML for the current table of rates actually contains hidden (commented) code for an additional column called "Properties at Per Diem". If it was not commented out, then it would appear on the right side of the table and would contain a link to a list of hotels that honor the per diem rates. It appears that this column used to be visible but has since been commented out. Although this would not affect any macro automating the scenario above, it could affect any macro that attempted to operate on this list of hotels.

The government occasionally posts alerts on the per diem web site, and these may force the user to make modifications. For example, since we first documented this scenario, the government has posted a document with special rules for traveling to areas affected by Hurricane Katrina. The user might want to add special rules to the macro to handle these new government instructions.

The Internet Archive's Wayback Machine has an archive of the NACO county lookup site from April 2003. This reveals that the city widget's name has not changed; however, the text on the submit button (which has no name) did change, from "Search for City" to "Search for County." Any macro that targeted this button based on human-readable label would have required maintenance activities.

In addition, the current version of this page provides four different ways of looking up the county, whereas the old version only provided two ways. Although the output page (Action 6.1) is not available in the Wayback Machine, it seems possible that the addition of new functionality may have resulted in modifications to the output page. This could have prompted maintenance activities.

3.4. Catalog of Scenarios

The following is a brief summary of the scenarios, roughly sorted in increasing order of complexity. We give the scenario name, typical user, scenario source, pre-conditions, post-conditions, and task overview. Full descriptions of each scenario, with screenshots, are available on our wiki.

Currency Converter – office worker (contextual inquiry)

Pre: A spreadsheet has a row for each expense incurred on a trip (showing each expense's date and amount in a foreign currency).

Post: Each row must also contain the amount in US dollars.

Overview: Use converter at www.oanda.com to do currency conversion, then copy results to the spreadsheet.

Package Tracker – online shopper (own experience)

Pre: A spreadsheet has a row for each tracking number.

Post: Each row must be updated to contain the package's status.

Overview: Use www.dhl-usa.com to look up each package's current status, then copy results to spreadsheet.

Path to Procurement – office worker (co-workers)

Pre: A spreadsheet has a row for each item that a worker would like the purchasing department to buy (with item description, quantity, and price).

Post: An order must be placed for each item.

Overview: Use a web form (which is buried deep within a labyrinthine intranet site) to add each item to the shopping cart, then submit the cart; this emails the cart's contents to the purchasing department.

Peoplesoft Scraper – IT staff (co-workers)

Pre: A Peoplesoft system contains a list of workers.

Post: A spreadsheet must be created, with one row per worker of interest (with each worker's name, phone number, office code, and job title).

Overview: Submit a web form to query for a list of workers. For each result, follow a link to access a page with the worker's details; copy these to the spreadsheet.

Per Diem Lookup – office worker (contextual inquiry)

Pre: A user is editing an expense report in a web form; form fields include a date and city/state.

Post: A form field must be populated with the government-approved per diem rate for that date and locality.

Overview: Navigate image map at www.gsa.gov to choose the state, select the year, then find the city and date in a table to locate the result. If the city is not shown, then look up the city's county and try finding per diem based on county. If the county is not shown, add two numbers on the page to compute a default per diem.

Person Locator Scraper – volunteer developer (online)

Pre: A web site displays a multi-page list of people and their status after Hurricane Katrina.

Post: An XML file must be created, with one node for each person (with that person's name, location, etc.)

Overview: Page through the list, performing minor transformations on the data before storing as XML.

Scraper for CMS (Content Management System) – webmaster (own experience)

Pre: A site contains a multi-page list of training events.

Post: Each event's data must be copied from the source site to a web form that adds the event to a CMS on another site.

Overview: Page through the list, performing minor transformations on the data, and then submitting through the CMS web form.

Staff Lookup – office worker (contextual inquiry)

Pre: A spreadsheet has a list of worker names, one per row.

Post: Each row must also contain the employee's phone number, email address, and job title.

Overview: Use form at people.cs.cmu.edu to look up each person's data, do minor reformatting, then paste results into appropriate columns of the spreadsheet.

Stock Analysis – financial analyst (online)

Pre: A spreadsheet has a row for each stock (with the ticker symbol and a date).

Post: Each row must contain a variety of statistics on that stock (including averaged volume, price, ratios, etc.)

Overview: Use forms at finance.yahoo.com and moneycentral.msn.com to retrieve the data, which are in tables. Date calculations are required to retrieve the right data.

Watcher for eBay – online shopper (own experience)

Pre: User has the tracking number for an item on eBay.

Post: The item's name, image, and various statistics must be displayed in a "pretty-printed" format.

Overview: Use www.ebay.com to retrieve data, then concatenate with HTML to form the pretty-printed format.

3.5. From Scenarios to Requirements

We began to generate requirements from scenarios by performing functional decomposition of the scenarios. This involved documenting the key web pages, spreadsheets, and other data sources that a web macro would need to access in order to perform the scenario. Next, we identified the inputs and outputs that a macro would need to read and write. This clarified the operations that a macro would need to support in order to compute appropriate outputs from inputs, as well as steps necessary for identifying inputs on the page.

We then augmented our set of functional requirements with non-functional requirements by considering what would be required to ensure macro maintainability, robustness, and other essential quality attributes. To assess maintainability requirements, we used the Internet Archive [8] to see how web sites had evolved in the past, and we considered what tool features would be highly beneficial for debugging and changing macros in response. By considering how web site evolution might cause web macros to malfunction, we also identified robustness requirements.

Finally, we collected the resulting requirements and organized them into eight groups. The first six of these groups relate to the timeline of a macro as it executes: the macro is *triggered*, the macro *uses objects on web pages* to read data (which may require *adapting to web page* evolution), the macro may *read/write data outside web pages*, and the macro may *transform data* from inputs to new outputs (perhaps through *control flow* constructs). Two additional groups contain requirements that apply outside the normal execution of a macro: the macro may need to *recover from failure*, and the end-user programmer may need make changes to the macro during *maintenance*.

To summarize the linkage between the scenarios discussed in this section and the eight groups of requirements that Section 4 describes in detail, we have provided Table 2, which indicates the scenarios that led us to each requirement. An additional benefit of this table is that it highlights requirements that are required to support many scenarios. For example, using *Text snippets* is part of every scenario and is therefore an essential feature of any macro tool. Because there are so many ways in which each scenario can break and require maintenance, we have marked each scenario's box in this table for the rows that deal with exception handling and maintenance.

Table 2: Cross-reference between scenarios and requirements

Scenario Title / Typical user **Category -** Requirements	Currency Converter Office worker	Package Tracker Online shopper	Path to Procurement Office worker	Peoplesoft Scraper IT staff	Per Diem Lookup Office worker	Person Locator Scraper Volunteer developer	Scraper for CMS Webmaster	Staff Lookup Office worker	Stock Analysis Financial analyst	Watcher for eBay Online shopper
4.1 Triggering macros										
- On-demand execution	X	X	X	X	X	X	X	X	X	
- Event-based triggers						X				
- Scheduled execution	X									X
- Subroutines	X					X	X			
4.2 Using objects on web pages										
- Text snippets	X	X	X	X	X	X	X	X	X	X
- Tabular information		X	X	X	X	X	X	X	X	
- Web form widgets	X	X	X	X	X		X	X	X	X
- Other HTML structures										X
4.3 Adapting to site changes										
- Adaptation to changing page layout	X	X			X		X		X	X
- Adaptation to changing form fields	X	X	X						X	X
- Adaptation to changing URLs					X					
4.4 Reading/writing outside pages										
- Browser APIs	X				X					
- Spreadsheets and other files	X	X	X	X		X		X	X	
- Parameters containing user input	X		X	X				X		
4.5 Transforming data										
- Reformat to equivalent value	X				X	X		X	X	
- Extracting values' parts					X				X	
- Combining values					X				X	X
4.6 Executing control structures										
- Looping operations		X	X	X	X	X	X	X	X	X
- Conditional operations		X	X		X	X	X	X	X	
4.7 Recovering from failure										
- Partial restarts			X			X	X		X	
- Exception handlers	X	X	X	X	X	X	X	X	X	X
4.8 Supporting macro maintenance										
- User-understandable representation	X	X	X	X	X	X	X	X	X	X
- Editable macros	X	X	X	X	X	X	X	X	X	X
- Features for debugging	X	X	X	X	X	X	X	X	X	X
- Maintenance at runtime	X	X	X	X	X	X	X	X	X	X

4. Requirements for Web Macro Tools

In this section, we present the 25 requirements, grouped into eight subsections. These groups range from the straightforward (Triggering macros and Using objects on web pages) to more sophisticated (Adapting to site changes and Recovering from failure). We illustrate each requirement with specific examples from the scenarios introduced in Section 3.

4.1. Triggering macros

All scenarios involve some pre-conditions, so the corresponding macros should not begin to execute until those conditions are met.

On-demand execution

Most scenarios begin when a user consciously decides that it is time to begin performing a task. These include scenarios that read data from spreadsheets (e.g.: Currency Conversion) and those that perform lookup operations to help the user fill out a web form (e.g.: Per Diem Lookup). When a macro uses a spreadsheet as input, and then writes results back to the spreadsheet, it would be helpful if the macro player provided buttons in Excel so that the user could open up the spreadsheet and play the macro.

Event-based triggers

The Watcher for eBay scenario begins when the user visits his iGoogle homepage (a portal). The macro triggers on page load, and the macro's output is formatted as HTML and inserted into the page's HTML structure, yielding the "portlet" user interface shown in Figure 1. (This portlet was implemented with script, rather than PBD.) Of course, the user may not be visiting the iGoogle homepage in order to check on the eBay auction, yet the portlet still executes. That is, the scenario is implicitly triggered by the event of visiting the browser homepage, rather than by a conscious "on-demand" command by the user.

Figure 1: The Watcher for eBay scenario involves using retrieved data from eBay to generate an HTML table that is injected into an iGoogle portlet.

Scheduled execution

In scraping scenarios, the input data come from a web site, and fresh data could arrive at any time. Consequently, these scenarios might benefit from macros that "poll" web sites for data. To achieve this, the macro tool might provide a user interface so that users could schedule playbacks. Alternatively, it could offer a command-line interface so users could schedule playbacks using operating system facilities.

Subroutines

Some organizations have multiple staff directories, so a macro might call several Peoplesoft Scraper or Staff Lookup macros and then merge the results. In such cases, the macro tool should support triggering a macro through a subroutine call.

4.2. Using objects on web pages

Macros are built from primitive operations that use a variety of objects on web pages.

Text snippets

All scenarios demonstrate that web macro tools should be capable of retrieving web pages from servers and extracting portions of the pages' text. The text is sometimes delimited with an HTML tag of its own. However, the text may be buried in a larger section of text with no HTML tags to delimit the target text.

Tabular information

Several scenarios involve interpreting tabular information and retrieving data from one or more rows or columns. For example, in Per Diem Lookup and Stock Analysis, the macro should retrieve data from specific rows that have an appropriate date in the leftmost cell. (See steps 3 and 6.1 of the sample scenario in Section 3.3.) Achieving this

requires identifying the table within the HTML, parsing it into keyed records, filtering records based on whether their respective keys match certain criteria, and then retrieving fields within those records for use in computations.

Web form widgets

Most scenarios involve getting or setting values of web form widgets, which include textboxes, dropdowns, and radio buttons, as shown in Figure 2. In many cases, the tool could compose http operations directly (rather than contacting the server indirectly by rendering pages, filling widget values, and clicking a submit button), which would reduce the need for manipulating widgets. However, the macro player will still need to support widget get/set operations since scenarios like Per Diem Lookup require reading inputs and writing outputs to a form that the user has opened in another browser window.

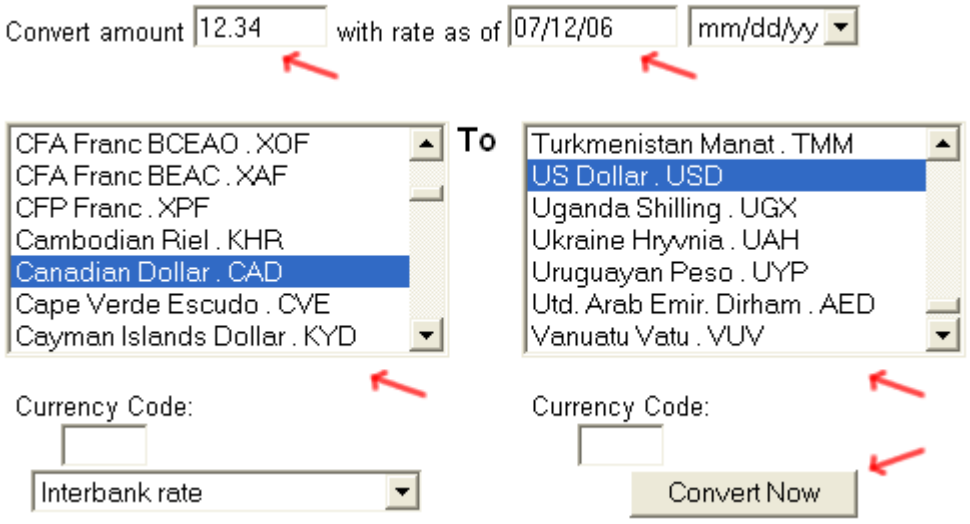


Figure 2: Like many scenarios, the Currency Converter scenario requires reading and writing form widgets, as well as clicking on buttons.

Other HTML structures

As shown in Figure 1, Watcher for eBay demonstrates display of HTML. The ideal macro tool will allow users to reformat macro output into a textual or HTML format, possibly using a template that the player fills in at runtime, and then display the result.

4.3. Adapting to site changes

Web pages might change between the recording of a macro and its playback, which could cause unintended effects at runtime. Such page evolution in scenario sites is documented by the Internet Archive's Wayback Machine [8] and by comments in sites' HTML.

Adaptation to changing page layout

Most existing tools find text on a page using one of two approaches, each of which has limitations.

If macro players only find values based on one or two visual characteristics of the text, then changes in the font, color, and other visual attributes could break a macro. For example, if a Currency Converter macro tries to find the second red text on the page (which is the output value in US dollars, as shown in Figure 3), and the site evolves so this text changes color (as it has in the past), then the macro will be unable to find the value.

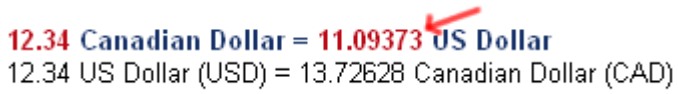


Figure 3: The top two numbers in the Currency Converter's output, above, are red. A macro would need to read the second of these numbers and store it in a spreadsheet.

If macro players find values based on structural characteristics of the HTML, then evolution in page layout could break macros. For example, if a Package Tracker macro tries to find the result table based on nesting of HTML tags, and the site evolves so the results are moved inside of another table (as has happened in the past), then the macro will be unable to find the values.

A successful web macro tool may need to combine the two approaches above with additional heuristics. For example, Creo can recognize text based on the semantic category of the text (e.g.: a food item) [4].

Adaptation to changing form fields

Macro tools directly or indirectly transmit a list of variable names and corresponding values. Variable names should match the names that the server is expecting; in particular, the names should match the names of widgets on the web form.

Therefore, evolution in the names of form widgets can break macro players. For example, HTML comments indicate that in 2005, a programmer added a new hidden field to the Path to Procurement web form; presumably the server software was also modified so that it now uses this new variable. Any macro recorded prior to the addition of

this field would not contain any instructions for transmitting a variable with that name. Consequently, if the new version of the server software requires the presence of this hidden field, then the server might not perform as anticipated during playback.

Evolution in the internal *values* of form widgets (versus their human-readable text) can also break macros. For example, in the Currency Converter (Figure 2), the code for a Bulgarian Lev has changed from "BGL" to "BGN." If a user recorded a macro using "BGL", then the tool would still keep sending this old value, which the server later might not understand. Therefore, tools should be resilient to changes in widget values as well as changes in widget names.

Adaptation to changing URLs

Most scenarios start with "go to this URL," but like page structure, page locations change. For example, the government's Per Diem Lookup was located on the www.policyworks.gov server until it moved to www.gsa.gov. Macros that use the old URL would fail to locate the new page. Fortunately, the webmaster of the old server put up a web page informing users that the old content has moved and providing a link to the new location. Just as a human is capable of following this new link, a web macro tool should be capable of automatically doing likewise.

4.4. Reading and writing data outside of pages

All scenarios include reading and writing data from the browser, but some also involve reading and writing from other locations such as spreadsheets.

Even though our scenarios did not uncover them, we are aware that there are a number of other systems where web-related data often are located. These include databases, word processors, RSS feeds, web services, and email servers. Another simple but likely possibility is the operating system clipboard.

Browser APIs

It may be desirable to display output within the browser, but outside of the web page. For example, in a variation of the Currency Conversion scenario (documented on our wiki), the user would highlight an amount of foreign money on a web page and command the macro tool to begin executing an existing Currency Conversion macro, using the highlighted money amount as an input. The tool would infer the correct source currency from the source page's URL (e.g.: Euros), then feed the amount and the source currency into the converter to calculate the equivalent

number of US dollars, which the tool would display in a popup window. To support this scenario variation, the macro tool should be able to read highlighted text and the current URL at runtime, then display results in a popup.

Spreadsheets and other files

Several scenarios involve reading data items from a spreadsheet, using each data item to perform lookups on the web, and then writing the results back to the spreadsheet. In addition, the Person Locator Scraper writes an XML document; to support this scenario, the macro recorder might allow the user to define a template that the macro player would instantiate and fill at runtime.

Parameters containing user input

Although most macro input comes from the sources described above, the user may want to parameterize the macro and explicitly provide values at runtime.

For example, several scenarios require authentication. When the user demonstrates the example and types a user-name and password, the tool could record the username and password, essentially hard-coding these as part of the macro, which could inhibit sharing the macro with other users. Or the tool could represent the username and pass-word as parameters that are undetermined until runtime, which could be a hassle when executing the macro. Since each option has trade-offs, the tool should allow the user to choose.

4.5. Transforming data

Our scenarios demonstrate that using data from the web necessitates more sophisticated transformations than simply unescaping HTML (e.g.: from `&` to `&`).

Reformat to equivalent value

The details of the Per Diem Lookup scenario involve a significant amount of reformatting. For example, matching up choices in the image map with values in the expense report requires reformatting between state names and state abbreviations. In addition, the scenario includes reformatting dates from `MM/DD/YYYY` to `Month D`. Finally, it involves capitalizing the county name for comparison to other county names.

Other scenarios also require small reformatting operations based on the semantics of the data. Examples:

- The Staff Lookup reformats phone numbers from `###-### ####` to `###-###-####` and strips spaces from email addresses (Figure 4).
- The Stock Analysis reformats dates from `MM/DD/YYYY` to `DD-Mon-YY`.

- The Person Locator Scraper interprets status data for each person record to set a Boolean flag indicating if the person was found after the hurricane. For example, if the person is "Hospitalized" or "Deceased", then the Boolean is set to true. This essentially entails passing the value through a lookup table.

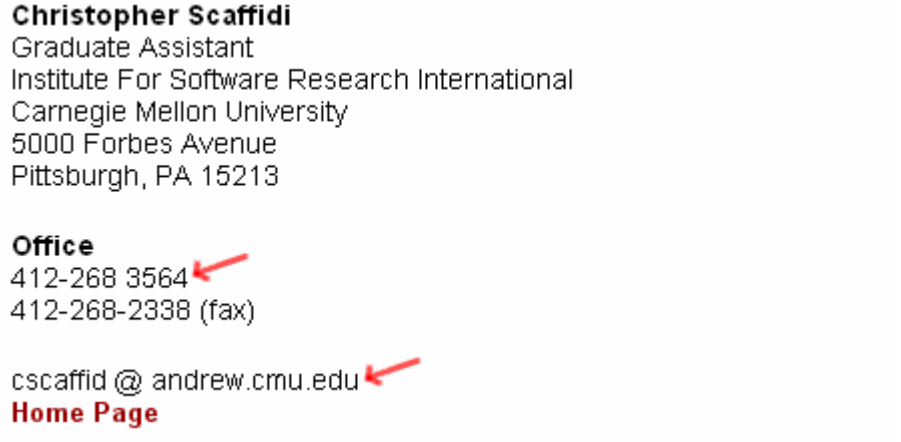


Figure 4: In the Staff Lookup scenario, phone numbers and email addresses contain space characters that need to be removed.

Operations like these transform a data value to another that is semantically "equivalent" for the purposes of the scenario. Macro tools could provide a way for users to specify transformations like these. In addition, the tools could intelligently perform commonly occurring transformations, such as those involving dates.

Extracting values' parts

Various scenarios require extracting part of a value. For example, Per Diem Lookup extracts the year from a `MM/DD/YYYY` value. It would also extract the city and state from a `City, ST` value. Thus, tools should enable users to extract parts of strings.

Combining values

Some scenarios involve combining data. The mode of combination depends on values' types. Examples include arithmetic with numbers (in Per Diem Lookup), date range comparisons (in Per Diem Lookup), and string concatenation (in Watcher for eBay).

4.6. Executing control structures

Macro recorders should support three types of operations: primitive, looping, and conditional. As discussed above, primitive operations include those required for manipulating the web browser (such as reading tables). We consider looping and conditional operations here.

Looping operations

Sometimes an operation's target is a set of strings or numbers. For example, the Per Diem Lookup picks two numbers out of the text and adds them together to generate a default per diem rate.

Scenarios demonstrate other repetitions of an operation on each record in a set. For example, several scenarios repeat actions for each row in a spreadsheet. In addition, the scraper scenarios repeat read operations for each page in a list of pages. Finally, many scenarios perform a read operation on each HTML table row while paging through a web site.

Support for a general `while(condition)` construct might be useful for polling web sites until a condition is met, such as polling the Hurricane Katrina web site in the Person Locator Scraper to watch for new data.

Conditional operations

Sometimes a scenario requires different actions depending on conditions at runtime. For example, Staff Lookup picks text differently from the page, depending on whether zero, one, or more people have the same name. A single demonstration can only exemplify one of these three conditions, so the web macro recorder may need to incorporate multiple examples, just as non-web macro recorders such as Eager have done [3].

4.7. Recovering from failure

In some cases, the macro tool will be unable to prevent failure. For example, the computer might lose its network connection, the server might crash, or the page might have evolved so much that the macro tool cannot automatically determine how to use the new page. In these cases, the macro player should help the user recover from the failure as gracefully as possible.

Partial restarts

If a macro fails halfway through a scenario, it may be safe to restart the macro from the beginning. This is typical with scraping and lookup scenarios. For example, if the Staff Lookup successfully retrieves data for 50 of 100 co-workers, but then the server crashes, then it is safe to restart the macro.

Of course, repeating work is wasteful. Moreover, some operations are not safe to repeat, due to side-effects. For example, the Scraper for CMS scenario inserts records into the target site. Repeating these operations would probably result in duplicates.

Consequently, the macro tool should track how far macros proceed. That way, if a macro fails, then the user has the option of doing a partial restart—that is, restarting the macro from where it left off.

Exception handlers

The macro tool should allow the user to specify how to handle exceptions. In addition, the macro tool should help users add exception handlers as the user adds new examples, as these examples will uncover new response patterns by the server. As described above, several scenarios include conditionals that cope with differences in how the servers respond to different inputs.

For example, tools could enable users to create an assertion that fires at runtime if data looks out of the ordinary or if the web page's structure seems to have changed in a way that the tool cannot automatically handle. The tool could alert the user and ask for guidance. If users could attach assertions and exception handlers to existing macros, then they could reuse another person's macro and add assertions to help ensure that the macro would behave as desired.

4.8. Supporting macro maintenance

Records in the Internet Archive demonstrate that many of the sites involved in our scenarios have evolved significantly over the years [8]. In some cases, site evolution might have broken macros automating the scenarios. Therefore, macro tools should support the maintenance of macros by end user programmers.

User-understandable representation

Before a user can perform maintenance, it is first necessary to understand the macro's structure. In addition, a user-understandable representation of macros may prove extremely valuable for other activities. For example, if one user offers to share a macro with another user, the recipient can examine the macro before executing it, in order to determine whether to trust the macro. To support these activities, tools should provide a user-understandable representation of macros.

Editable macros

Another basic requirement for maintenance is the ability to make changes to existing macros. Desirable edit operations include deleting operations, adding operations, changing operations, wrapping operations in loops, and many of the other types of edits that are currently supported in textual editing environments.

Features for debugging

Many professional programmers have come to rely on various sophisticated debugging services within the development environment. Tools could include features for traces, breakpoints, step-by-step execution, and runtime variable inspection. Macro tools have only recently begun to provide similar features, as we discuss below.

Maintenance at runtime

A macro might break because site evolution prevents the tool from finding text, getting or setting widget values, or following URLs. However, the changes leading to the broken macro might have been minor, such as a change of font or a renaming of a widget. In such cases, it would be desirable if the macro tool provided a way for the user to modify the macro to fix it at runtime. For example, the user could highlight the data or widget so the tool could relearn how to find the data or widget.

For larger changes, the tool may need to provide mechanisms to add new operations. For example, the government site in the Per Diem Lookup sometimes displays new regulations on how to use the site. The macro tool could let the user specify that the macro should check at runtime if these regulations changed—and, if so, to enter a maintenance mode so the user could incorporate the new regulations into the macro.

5. Example Benchmark Uses

To illustrate using the requirements as a benchmark, we analyze support for requirements by three web macro tools. We are developing two of these tools, Robofox and CoScripter [10][12], while iOpus has produced the iMacros tool [7].

All three tools are available as Firefox plug-ins, and iMacros is also available as an Internet Explorer plug-in. All three tools are intended for use by ordinary end users who lack formal training in programming; example users include information workers, e-commerce shoppers, and the other users typified in our scenarios. (Of course, the tools are also intended to be usable by more sophisticated users such as software engineering researchers.) In addition, iMacros is available in a Scripting Edition that offers powerful integration with the VBScript programming language

so that professional programmers can gain additional benefit from iMacros. While this integration provides the ability to call iMacros web macros from VBScript (and vice versa), our paper has focused on the needs of end-user programmers who typically are unfamiliar with VBScript. Consequently, we will focus below on whether requirements be met with iMacros without resorting to VBScript.

The three tools differ somewhat in their level of adoption. At present, Robofox serves as a testbed for evaluating advanced web macro tool features and is not widely deployed. CoScripter is used semi-regularly in daily work by several dozen administrative assistants, researchers and other workers within IBM, and it has been downloaded for free by hundreds of non-IBM users. iMacros has thousands of paying users worldwide.

Table 3. Support by Robofox, CoScripter, and iMacros for scenario requirements

Requirements	**Robofox**	**CoScripter**	**iMacros**
Triggering macros			
On-demand execution	Yes	Yes	Yes
Event-based triggers	Yes	Yes	No
Scheduled execution	Yes	Yes	Yes
Subroutines	No	No	No
Using objects on web pages			
Text snippets	Yes	Limited	Yes
Tabular information	No	Limited	Limited
Web form widgets	Yes	Yes	Yes
Other HTML structures	No	No	No
Adapting to site changes			
Adaptation to changing page layout	Limited	Limited	No
Adaptation to changing form fields	Yes	Limited	No
Adaptation to changing URLs	No	No	No
Reading and writing data outside of pages			
Browser APIs	Limited	Limited	Limited
Spreadsheets and other files	Limited	Limited	Limited
Parameters containing user input	Limited	Yes	Yes
Transforming data			
Reformat to equivalent value	No	No	No
Extracting values' parts	Yes	No	No
Combining values	No	Limited	Limited
Executing control structures			
Looping operations	Limited	Limited	Limited
Conditional operations	Yes	No	No
Recovering from failure			
Partial restarts	No	No	No
Exception handlers	Yes	No	No
Supporting macro maintenance			
User-understandable representation	Yes	Yes	Limited
Editable macros	Yes	Yes	Yes
Features for debugging	Yes	Yes	No
Maintenance at runtime	Yes	Yes	No

Robofox

As shown in Table 3, Robofox lacks support for seven requirements and only partially supports another five. For example, although Robofox does not automatically perform *adaptation to changing page layout*, it uses visual heuristics to find objects on pages and inserts "sanity check" assertions after operations to test if the page's structure matches the tool's expectations. If page layout changes so dramatically that the heuristics cannot find an object, then the tool brings the changes to the user's attention so the user can do maintenance. Similarly, although Robofox partially supports accessing *spreadsheets and other files*, users have limited control over files' structure. Robofox supports *looping operations* over sets, but not arbitrary `while(condition)` loops.

On the other hand, Robofox provides a wide variety of features to support macro maintenance. For example, Robofox's *debugging features* include "anticipation highlighting," which uses green highlighting to indicate what action the tool will perform if the currently selected line of a macro is executed. In addition, Robofox automatically supports a wide variety of automatically-generated assertions as well as assertions manually added by users. When assertions are violated, the tool enters a *maintenance at runtime* mode so that the user can fix the macro if needed.

Because of unsupported requirements, Robofox cannot support at least five scenarios: Per Diem Lookup, Person Locator Scraper, Staff Lookup, Stock Analysis, and Watcher for eBay. Variations of two scenarios are unsupported: Currency Conversion and Peoplesoft Scraper. Ongoing site changes would have caused macros for many scenarios to break, due to Robofox's limited support for automatically adapting to site changes.

The list of unsupported scenarios would be reduced considerably by adding support for two requirements: using *tabular information*, and *reformat to equivalent value*. With these additions, Robofox would support Per Diem Lookup, Staff Lookup, and Stock Analysis fairly well (with limited automatic adaptation to site changes), leaving two scenarios and two variations unsupported.

CoScripter

CoScripter completely supports nine requirements and partially supports another eight. For example, to provide a *user-understandable representation*, it presents macros as a series of English-like statements such as "click the Log On button". This is not only the representation displayed to the user, but actually the representation in which macros are stored internally. To robustly handle slight deviations from this syntax, CoScripter uses a "sloppy parser" that in-

fers a likely interpretation of the macro's instructions. CoScripter's *debugging features* include anticipation highlighting.

Although the tool cannot automatically reason about sophisticated *tabular information* (as in steps 3 and 6.1 of the Per Diem Lookup), but the CoScripter extension called "Vegemite" allows users to organize *text snippets* into spreadsheet-like "scratch spaces" [23]. Users can create macros that compute cell values by posting other cell values through a web form and retrieving data from the web server. CoScripter currently provides only minimal support for *combining data values*, in that data in a scratch space can be manipulated with arithmetic formulas demonstrated by the user with a calculator. Once a macro is demonstrated for one row of a scratch space, Vegemite supports *looping operations* that execute the macro on other rows in the table.

Because of its limitations, CoScripter can only completely support the PeopleSoft Scraper scenario. The primary problem is its lack of support for *conditional operations*, which are required by nearly all scenarios, as well as *reformat to equivalent value*. Adding support for these requirements would enable CoScripter to support three additional scenarios fairly well, though without support for *partial restarts* or *exception handling*, which would be required to help ensure macro robustness.

iMacros

iMacros supports six requirements and partially supports another six (without resorting to VBScript). iMacros was primarily designed as a screen-scraping tool, and it can scrape columns or rows of *tabular information* or other pieces of web pages and export them to plain text files. However, it is not possible to perform the complex row selection or row filtering described in Section 4.2. In addition, like Robofox, iMacros can export strings to *spreadsheet*-like comma-separated files, but the user does not have much control over the layout of the resulting files. iMacros can display popup windows but is not integrated with other *browser APIs* such as clipboard support. Users can *combine values* with numeric addition, but iMacros does not support other arithmetic operators. It is debatable whether macros have a *user-understandable representation*: they often contain lengthy chunks of HTML (for use in locating widgets on the page), and many end-user programmers cannot read HTML.

Because of unsupported requirements, iMacros can only support the Peoplesoft Scraper scenario. In addition, macros are probably fairly brittle with respect to page evolution, since they use literal pieces of HTML in order to reference widgets and data on pages. The lack of any debugging tools—even the step-by-step execution present in

the Robofox and CoScripter tools—could make it difficult for an end-user programmer to diagnose macro failures. However, judging from user testimonials on the iOpus web site, iMacros has been successful among professional programmers (rather than end-user programmers), largely because scraping web pages with iMacros and VBScript requires much simpler code than scraping web pages with VBScript alone.

6. Discussion

Web macro tools have evolved significantly over the past decade, but they have not yet reached their full potential. In particular, our scenarios have highlighted the need to support 25 requirements, which we have classified into 8 groups. These groups of requirements can be further categorized into the following three clusters:

Cluster #1: Technical requirements tightly coupled to current technology: Two groups of requirements are purely based on the need for a web macro to interact with today's browsers, spreadsheets, and other software. Section 4.2 involved reading/writing objects on web pages (such as *web form widgets* and *text snippets*), and Section 4.4 dealt with reading/writing outside pages (such as through *browser APIs* and *spreadsheets*). As new technology arises, these two groups of requirements will evolve, providing new opportunities for web macros to save users time when transferring data among web pages and pieces of software.

Cluster #2: Requirements tied to the web macro paradigm: Three groups of requirements largely relate to making web macros convenient to create, use, and maintain. Section 4.1 described requirements for triggering web macros (such as *on-demand* or through *scheduled execution*), Section 4.7 dealt with recovering from failure (including supporting *partial restarts* of web macros), and Section 4.8 discussed the need to support macro maintenance (including providing *editable macros*). These requirements in Cluster #2 are not tightly coupled to any particular browser or spreadsheet API. Therefore, to keep pace with changes in the underlying technology, solutions to requirements in Cluster #2 will require less ongoing modification than solutions to requirements in Cluster #1.

Cluster #3: Requirements that generalize beyond web macros: An additional three groups of requirements not only deal with making web macros convenient and useful, but solving requirements may yield significant applications in other areas of computer science. Section 4.3 discussed the need for macros to adapt to web site changes (such as *changing page layout* and *changing URLs*), which is also a challenge in search engines and product recommendation engines [6] that automatically read data from particular areas of web pages for data mining purposes. Sec-

tion 4.5 described the need to support data transformations (such as *reformatting data to equivalent values* and *extracting values' parts*), and these data operations are currently difficult and time-consuming to code in other end-user programming tools, such as spreadsheets and tools for creating web applications [19]. Section 4.6 dealt with inference of control structures (such as *loops*) from examples, which is a general problem in PBE tools [11][16]. Meeting the requirements in Cluster #3 may provide insights, models, and algorithms that can be reused to solve these other problems in other domains (and vice versa).

Finally, we want to note three requirements in particular that are unsupported by all existing web macro tools (to our knowledge) but that might be possible to address in the near future: *partial restarts*, *adaptation to changing URLs*, and *output to HTML structures*.

It might be possible to support *partial restarts* by extending the notion of transactions to PBD web macro tools. Operations that can be safely repeated are generally performed with http GET rather than http POST requests.[2] One approach to achieving partial restarts would be for the macro player to log operations (and cache http results) as they occur. If a failure occurs, then the player could break the log into transaction-like segments, with POST operations identifying the boundaries between segments. All POST operations prior the failure should not be repeated, and any necessary data from GET operations prior to those POST operations should be served from the cache. Any GET operations after the last POST operation could be re-executed if desired. In effect, the partial restart would begin from the moment after the last POST operation. Of course, this approach would not work perfectly in every circumstance, such as when authentication should be repeated if a macro restarts. Further research might identify cases like these and address them with appropriate algorithms and user interfaces.

The *adaptation to changing URLs* requirement appeared in situations when a web page moves to another server. New heuristics, perhaps combined with existing screen-reader technology, might be able to detect that a webmaster has posted a human-readable message indicating a page's new URL. Sometimes, detecting that a page has moved is relatively easy, since some sites use an HTML META refresh tag or an http header to redirect browsers. In any case, if a macro player is able to find a new URL, then with the user's permission, it could retrieve the content at that new

---

[2] http specification, `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

URL and see if the structure matches the expected structure at the old URL. If so, the macro player may be able to update the macro and continue.

A final noteworthy requirement is to *output HTML structures*. While older web macro tools enabled users to define simple web pages as web macro output [14][18], the Watcher for eBay scenario (Figure 1) highlights the need to help users generate more sophisticated HTML structures, such as iGoogle portlets. Other possible output forms might include geographical map visualizations and DHTML animations. Clearly, research teams developing web macro tools cannot directly support every conceivable output visualization. Consequently, it may be better to view web macro recorders and players as reusable engines rather than finished products, and to design them accordingly. That way, other researchers and industry partners could reuse web macro tools by attaching them to novel output visualizations. Equally importantly, intentionally designing web macro tools as reusable modules would make it easier for researchers to use features from existing tools in order to meet more sophisticated requirements as they are identified in new scenarios.

7. Acknowledgements

We thank our paper reviewers, as well as Mary Shaw and other EUSES Consortium members, for helpful suggestions. This work was funded in part under ITR grant CCF-0325273 (via EUSES) and by NSF under ITR grants CCF-0438929 and CCF-0324861. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8. References

[1] M. Apperley, D. Fletcher, B. Rogers. Breaking the Copy/paste Cycle: The Stretchable Selection Tool. *AUIC'00: First Australasian User Interface Conference*, 2000, 3-10.

[2] L. Beckwith, S. Sorte, M. Burnett, and S. Wiedenbeck. Designing Features for Both Genders in End-User Programming Environments. *VL/HCC'05: Proc. Visual Lang. and Human-Centric Computing*, 2005, 153-160.

[3] A. Cypher. EAGER: Programming Repetitive Tasks by Example. *CHI'91: Proc. Conf. Human Factors in Computing Systems*, 1991, 33-39.

[4] A. Faaborg, H. Lieberman. A Goal-Oriented Web Browser. *CHI'06: Proc. Conf. Human Factors in Computing Systems*, 2006, 751-760.

[5] J. Fujima, A. Lunzer, K. Hornbæk, Y. Tanaka. Clip, Connect, Clone: Combining Application Elements to Build Custom In-

terfaces for Information Access. *UIST'04: Proc. 17th Symp. User Interface Software and Technology*, 2004, 175-184.

[6] M. Hu and B. Liu. Mining and Summarizing Customer Reviews. *Proc. 10th SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 2004, 168-177.

[7] *iOpus* corporate website, www.iopus.com

[8] *Internet Archive Wayback Machine*, www.archive.org

[9] E. Kandogan, E. Haber, R. Barrett, and A. Cypher. A1: End-User Programming for Web-Based System Administration. *UIST '05: Proc. 18th Symp. User Interface Software and Technology*, 2005, 211-220.

[10] A. Koesnandar, S. Elbaum, G. Rothermel. *Building Dependable Web Macros with Robofox*. Technical Report TR-UNL-CSE-2006-0010, Dept. Computer Science and Engineering, University of Nebraska—Lincoln, 2006.

[11] H. Lieberman (Ed.). *Your Wish is My Command: Giving Users the Power to Instruct their Software*. San Francisco: Morgan Kaufmann, 2000.

[12] G. Little, T. Lau, J. Lin, E. Kandogan, E. Haber, A. Cypher. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *CHI'07: Proc. Conf. Human Factors in Computing Systems*, 2007, 943-946.

[13] R. Miller, B. Myers. *Creating Dynamic World Wide Web Pages by Demonstration*. Technical Report CMU-CS-97-131, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997.

[14] B. Myers. *Creating User Interfaces by Demonstration*, Perspectives in Computing, Vol 22, Academic Press, 1988.

[15] J. Pane, B. Myers, and L. Miller. Using HCI Techniques to Design a More Usable Programming System. *HCC'02: Proc. Human Centric Computing Lang. and Environments*, 2002, 198-206

[16] R. Potter, D. Maulsby. A Test Suite for Programming by Demonstration. In *Watch What I Do: Programming by Demonstration*, 1993, 539-592.

[17] J. Rode and M. Rosson. Programming at Runtime: Requirements and Paradigms for Nonprogrammer Web Application Development. *VL/HCC'03: Proc. 2003 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2003, 23-30.

[18] A. Safonov, J. Konstan, J. Carlis. Towards Web Macros: A Model and a Prototype System for Automating Common Tasks on the Web. *Proc. Conf. Human Factors and the Web*, 1999.

[19] C. Scaffidi, et al. Using Topes to Validate and Reformat Data in End-User Programming Tools. *Fourth Workshop on End-User Software Engineering*, Leipzig, Germany, 2008, to appear.

[20] C. Scaffidi, A. Ko, B. Myers, M. Shaw. Dimensions Characterizing Programming Feature Usage by Information Workers. *VL/HCC'06: Proc. 2006 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2006, 59-62.

[21] C. Scaffidi, M. Shaw, B. Myers. Games Programs Play: Obstacles to Data Reuse, *2nd Workshop on End User Software Engineering*, 2006.

[22] A. Sugiura, Y. Koseki. Internet Scrapbook: Automating Web Browsing Tasks by Demonstration. *Proc. 11th Symp. User Interface Software and Technology*, 1998, 9-18.

[23] J. Wong, J. Lin, T. Lau, and A. Cypher. Making Ad-hoc Mashups with Vegemite. Submitted for publication.