Scenario-Based Requirements for 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
cscaffid,bam @cs.cmu.edu

[2]*Almaden Research Center*
IBM
Almaden, CA
acypher@us.ibm.com

[3]*Computer Science and Engineering Department*
University of Nebraska-Lincoln
elbaum,akoesnan @cse.unl.edu

Abstract

Web macros automate interactions 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 needs of users 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.

1. Introduction

Researchers have successfully applied the programming-by-example (PBE) paradigm in many environments ranging from user interface design [11] to HyperCard [3]. In PBE, a macro recorder watches the user perform operations, determines the user's intent, and generates a macro to represent that intent (generally as a sequence of steps). Later, a macro player executes the macro on new data.

For over a decade, researchers have provided numerous tools that implement the PBE paradigm in the web context, automating actions that users take in a web browser [1][4][5][7][8][9][10][13][16]. Such tools provide several benefits. First, like traditional PBE tools, 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 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 [15]. 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 [14].

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 these requirements that web macro tools must support in order to be useful for many real-world tasks. We anticipate that this set of requirements will serve as a helpful benchmark to researchers for evaluating tools and identifying beneficial areas of work.

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 benefit people. Some tasks were performed repetitively by us, and we would like to automate these tasks to save ourselves time, but we have no suitable PBE macro tool. Finally, we have observed coworkers manually performing certain 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 automatically make tools perfect for all imaginable tasks. Instead, by linking requirements to a diverse set of spe-

cific 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 3 uses a scenario format to describe tasks. Section 4 analyzes scenarios to identify tool requirements, which include support for triggering macros, using objects on web pages, adapting to site changes, reading and writing data outside pages, transforming data, executing control structures, recovering from failure, and supporting macro maintenance. Section 5 demonstrates using requirements as a benchmark for evaluating the Robofox tool, thereby identifying areas for future work.

2. Comparison to Related Work

Many papers use scenarios to motivate and explain a PBE tool's features [1][4][5][7][8][9][10][13][16]. 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, [10] 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. As another example, [4] describes a scenario of reading a recipe on a web site, then using the ingredients list and another site to compute the recipe's nutritional value.

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 and provide limited support for evaluating future tools.

Second, the "pedigree" of scenarios is rarely documented: that is, it is usually unclear if 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 [12], 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.

3. Scenarios

Each scenario 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 shied away from 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, we want to provide a benchmark to measure improvements to macro tools. Therefore, we have chosen scenarios that highlight the ground that PBE web macro tools have yet to cover. 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, to explore the breadth of macros' applicability, we have selected scenarios involving several types of users. These include office workers, online shoppers, financial analysts, IT staff, and others. Moreover, these scenarios come from a variety of sources:

Contextual inquiry

Three scenarios were uncovered by our recent study of office workers [15]. 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.

Co-workers

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.

Online sources

Three scenarios involving 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 web macro scenario.

Our own experience

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.

3.1. Structure of Scenarios

Users currently perform many tasks manually or by writing a script. We hope to foster innovative development of PBE tools that will someday be able to achieve the same tasks.

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 implementation 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 by tools, we do not specify *how* post-conditions 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 valid tradeoff worth considering. For example, some recorders use a pure PBE approach, while others allow users to augment the macro with procedural code [5].

3.2. Catalog of Scenarios

The following is a brief summary of the scenarios, roughly sorted in order of complexity. We give the scenario name, typical user, scenario source, pre-conditions, post-conditions, and task overview. Full details on each scenario, with screenshots, are available on our wiki[1]. We hope that over time, this benchmark will grow as macro tools meet existing requirements and other researchers contribute new scenarios.

Currency Converter – office worker (contextual inq.)
Pre: A spreadsheet has a row for each expense incurred on a trip (showing each expense's date and amount).
Post: Each row must also show 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 show 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 hidden 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, follow a link to access a page with the worker's details; copy these to the spreadsheet.

Per Diem Lookup – office worker (contextual inq.)
Pre: A user is editing an expense report in a web form; form fields show expense 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 – webmaster (own experience)
Pre: A site shows 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 another site.
Overview: Page through the list, performing minor transformations on the data, then submitting thru form.

Staff Lookup – office worker (contextual inq.)
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 save.

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 show 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 pretty-printed format.

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

4. Requirements for Web Macro Tools

Our scenarios reveal requirements for effectively enabling users to record and replay web macros. Some requirements are partially satisfied by existing tools. Many other requirements remain to be addressed.

Some web macro tool limitations could be addressed through Semantic Web markup [2]. However, many other requirements are unrelated to how web sites are marked up (particularly in Sections 4.1, and 4.4 through 4.8), and semantic web markup will not address these requirements.

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 after a conscious demand by the user. These include scenarios that are driven by 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.

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 scheduled operation of macros, in a sort of "polling" process. 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.

Event-based triggers

The Watcher for eBay demonstrates the possibility of triggering a macro based on an event that is unscheduled and does not represent a conscious demand by a user at runtime. In this scenario, the macro triggers on a page load, and then its output is formatted as HTML and appended to the page's HTML structure.

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 must 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 must retrieve data from specific rows that have an appropriate date in the leftmost cell. 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, including textboxes, dropdowns, and radio buttons. 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.

Other HTML structures

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 [6] 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), 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.

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 must match the names that the server is expecting; in particular, the names must 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 valid *values* of form widgets can also break macros. For example, in the Currency Converter, 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 must 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 telling users that the old content has moved and providing a link to the new location. While some sites post pages like this when content moves, others use an HTML META refresh tag or an http header to redirect browsers.

In any case, the macro player should detect that the page has changed significantly. If a macro player detects that the page has changed considerably, it could examine the page to find a new URL. With the user's permission, it could then retrieve the content at that new 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.

4.4. Reading and writing data outside pages

All scenarios involve 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 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 tell 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 must 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 involve authentication. When the user demonstrates the example and types a username 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 password 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.

It may be beneficial to support "sticky" input parameters. For these, when a value is set for one execution of the macro, the macro tool would record that value and use it as a default value during the next execution. The user could always override the default for a given execution, thereby changing the default value for subsequent executions. Such "sticky" behavior could be used to store authentication data or complex inputs. For example, a variation of the Peoplesoft Scraper (documented on our wiki) would accept regular expressions that filter which employee records are retrieved. System administrators using such a tool might appreciate not having to retype the regular expressions unless there was a need to change the macro's behavior.

4.5. Transforming data

Our scenarios show that using data from the web involves 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 involves reformatting between state names and state abbreviations. In addition, the scenario involves 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 involve small reformatting operations based on the semantics of the data. Examples:

- The Staff Lookup repairs phone numbers from `###-### ####` to `###-###-####` format and strips spaces from email addresses.
- 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 involves passing the value through a lookup table.

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 involve 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 must 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 must 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 involves certain 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, or 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 must 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 there is no harm in restarting the macro later.

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 involve 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 show that many of the sites involved in our scenarios have evolved significantly over the years. 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.

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 Use: Robofox

To illustrate using the requirements as a benchmark, we analyze support for requirements by Robofox, a web macro tool that two of us are developing [7].

As shown in Table 1, Robofox lacks support for seven requirements and only partially supports 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.

By referring to Section 4, we see that because of these 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.

In short, our scenario-based requirements enabled us to identify two highly beneficial areas of future tool work. We anticipate that other researchers could likewise benefit from evaluating tools using these requirements.

Table 1. Robofox's support for scenario requirements

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

6. Acknowledgements

We thank Mary Shaw and other EUSES Consortium members for helpful discussions. 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.

7. 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] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. *Scientific American*, Vol. 284, No. 5, 2001, 34-43.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] B. Myers. *Creating User Interfaces by Demonstration*. PhD Thesis. Technical Report CSRI-196, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, M5S 1A1, 1997.

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

[13] 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.

[14] 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.

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

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