Tasklets: Enabling End User Programming of Web Widgets

Geetha Manjunath, Hewlett Packard Labs and Indian Institute of Science
M Narasimha Murty, Indian Institute of Science
Dinkar Sitaram, PES Institute of Technology
Bangalore, India

Abstract

Mobile widgets are now popular and form a new paradigm of simplified web. Probably, the best experience of the Web is when a user has a widget for every frequently executed task, and can execute it anytime, anywhere on any device. However, creating a widget today requires knowledge of programming, web technologies and protocols. In this paper, we propose a new method of web simplification that enables an end-user to create simple 'single-click' widgets for a personal task - without any programming. For this, we introduce a new concept called Tasklet to represent a user's personal interaction and model it using an instruction set over websites. These Tasklets can be programmed by demonstration and are executed using a Web Virtual Machine that virtualizes changes to web pages. We believe our approach opens up a different perspective of WWW as not just "web of pages" but "web of tasks".

Key words: Programming-By-Demonstration, Widgets, Tasklets, Web Virtual Machine, End User Programming, Mobile Widgets, Web Interaction Language

1. Introduction

Web being the key source of any content or service/application today, people frequently execute their daily or monthly activities on the web. However, the Web now has grown in multiple ways - content, complexity of navigation, number of web sites, web services and service providers. Performing such regular tasks (bill pay, check horoscope, check bank balance, etc) with current web interfaces require the user to remember website URLs, perform complex navigations, fill multiple forms and essentially figure out a sequence of actions that are needed to accomplish the simple task – often across multiple sites. Even when she succeeds in doing it once, the next time a similar goal arises, she has to recall and manually repeat the same exercise. This problem is even more acute in mobile context, when the device has interface challenges and user is very focused on getting a task done. We need to simplify this web interaction. By radically simplifying the usage of the web and providing familiar means of interacting with it, even non-tech savvy people can start deriving value from the web. Reduced interaction complexity of web can also be a major productivity improvement for an IT-savvy user. Finally, 1-click access to personally valuable web tasks from any mobile device is a compelling user experience of the mobile web.

Providing 1-click access through mobile widgets has become very popular and is a new paradigm of accessing web content today. Probably, the best experience of the web is when a user has a widget for every frequently executed task, and is able to pick it and execute it anytime, anywhere on any device. To make this a reality, either user finds the right app in an app-store or writes it herself. We believe it is very hard to find the widget of one's liking, because different people perform different tasks on their favourite website in their own personal way. There are more than 200 million websites

today [2] and even more number of tasks that one can do on these websites and combination of those. Therefore, we think it will be ideal if the end-user herself will be able to create new widgets that she needs most often. Thus, there is a need for end user programming of mobile widgets.

Today, if a user wants to create a new widget that does what she wants, it requires development of a new web application, which mandates programming knowledge of at least one of Java, JSP, Servlets, JavaScript or Action Script plus an understanding of web protocols and standards. Our goal is to break this limitation and enable the end-users to create simple 'single-click' widgets for their personal tasks.

Some of the technical challenges in achieving the above objective are listed below:

(a) Program synthesis: How does a user create a personal widget without programming? Can we use the simple paradigm of Programming-By-Demonstration (PBD) for this?

(b) Resilience to change: Web pages are not static; their content changes almost on an hourly basis. How do we ensure the widgets created by PBD works over changing websites?

(c) Privacy: When one creates a personal widget and wishes to share it, how do we maintain the privacy of the user data recorded within it?

(d) Disconnection Management: Mobile Devices are not always connected. How will these widgets behave in a disconnected environment?

(e) There are diverse kinds of mobile devices. Which mobile device to support? Can we do something that easily works for multiple devices?

In this paper, we propose an architecture and platform that addresses the above listed problems and supports end-user creation of widgets for any mobile device. We introduce a new concept called Tasklet to represent a user's personal interaction, and bring in a notion of programming over websites using a Web Instruction Set and a Web Virtual Machine. Since these Tasklets can be programmed by demonstration, the target user for our tool is an end-consumer with just simple web-browsing knowledge.

The key contribution of this work is in conceptualizing task-based user-created widgets that can be shared and executed in mobile context. The proposed solution facilitates one to look at the World Wide Web not as a "Web of Pages" but as a "Web of Personal Tasks", simplifying the access to the Web, particularly for first-time users of IT. From a technical aspect, we propose a method for semantic modelling of task-based web interactions through an interaction language called GOPI (Goal Oriented Personal Interaction). A method of handling changes to websites by interpreting this web instruction set with an associated Web Virtual Machine (WVM) is envisaged. We believe our solution for rapid widget authoring coupled with multi-device execution environment potentially opens up the value of the web by delivering personally relevant services to a large section of the non-tech-savvy users in the developing countries, as proved by our live pilot in India.

The rest of the paper is organized as follows. Section 2 introduces the concept of Tasklet and describes the proposed architecture for end user programming of web widgets. Section 3 describes the internals of the Tasklet Authoring Modules with a specific focus on the new web interaction language, GOPI. In Section 4, we present the algorithms used to model web changes and describe our approach to handle the same. Section 5 briefly describes the previous work related to end-user programming for Web. In Section 6 we share the current status of our work and present some results of an initial pilot with end users. We conclude in section 7 along with our thoughts on some possible future work.

2. Proposed Solution

We propose a solution to enable end-user widget creation for personal tasks without programming and to enable cross-device access to the same. We introduce a concept called Tasklet to represent a web task. Tasklets make web tasks as first class web objects on the Internet. In order to cater to diverse needs of end users, we support simple creation of Tasklets by just 'browsing'. This enables a non-tech-savvy user to view the Web as a "Web of personal tasks' as opposed to "Web of Pages". This section details these key concepts and the architecture of the proposed solution.

Concept of Tasklets

We define a concept called **Tasklet** to represent a user's personal web interaction pattern. A Tasklet models a sequence of web interactions needed to perform the web task. It captures a user's preferred way of accomplishing a web task – compressing the sequence of web actions needed to perform a specific task in a user-specified way. To create a new Tasklet, all the user needs to do is to demonstrate the task (web interaction) once using her web browser by giving some sample inputs on all the relevant web sites [1]. These Tasklets can be user-created, shared, customized and composed with other Tasklets and web services (such as language translator, text summarizer).

For instance, if a user frequently goes on 2-day trips from Bangalore to Delhi by Jet Airways, she can create a custom Tasklet that takes the date of travel as input, books her regular flights, and reserves a hotel room at her favourite hotel using her credit card details for payment – without doing any programming! Another example is a Bill-Payment widget with local-language interface that navigates to the right web portal, performs the payment, and translates the results to the user's language. A Tasklet captures a sequence of web actions across multiple sites under a widget. This simplified interface to a personal task spares a lot of non-productive time spent in executing frequent web actions. It can also be used by portal providers and service providers to mobile-enable their businesses.

In order to enable an end user to program a Tasklet, we use the technique of Programming-By-Demonstration. We record and analyze user's browser actions to auto-generate custom personal widgets, which compress all the required actions into a single-click interface. Further, using our cloud-hosted Tasklet services, these widgets can be executed from a mobile device, despite several limitations of the mobile device. They can be accessed from a mobile application, a mobile web browser, SMS or even over just a voice call.

Life Cycle of a Tasklet

The typical life cycle of a Tasklet starts at the Tasklet creation phase (Figure 1). Here the user performs the web interaction for the task of interest using some sample inputs. These user actions are recorded and analyzed along with the corresponding web pages to capture the semantics of the user actions in the form of a script, called **Tasklet Template Script**. This script is registered with a private repository of Tasklets. The Tasklet authoring service parameterizes the script, allows user to customize inputs, and creates a new Tasklet in the Tasklet Repository. The user can remove private data from the Tasklet and share it in a public repository. Once a Tasklet is part of the repository, it is assigned a unique URL with which the task can be repeated. It therefore becomes a first class web object that can be shared, annotated as well as "invoked" from any device.

Since Tasklet execution is hosted as a Cloud service, it can be invoked by different modes. The task can be executed with possibly a different set of inputs than what was used during creation. Access to the Tasklet URL triggers an instantiation of the Tasklet with new parameters and the sequence of web actions is replayed by the **Tasklet Execution Engine**. Result of Tasklet execution is returned back to the invoker either in a single or in multiple phases of output. Depending upon the device and the mode of invocation (SMS, voice, or mobile widget), appropriate transformation of the output may also be needed. We next describe the key phases of this Tasklet Lifecycle.

Tasklet Authoring

The main blocks of the authoring environment are shown in Figure 2. We have developed a browser plug-in to record the user's actions on the web browser. An explicit 'Record' button is used to show the start of recording. After the user goes through a sequence of browser-based interactions with multiple inputs across multiple pages (hyperlink traversals and form filling), the final page of transaction is reached, then the user selects or extracts the content of interest by just a "double click". User can also extract multiple fragments of text from different pages. All the above actions are logged in the recording file (**Browse-Action-Recording, BAR** file).

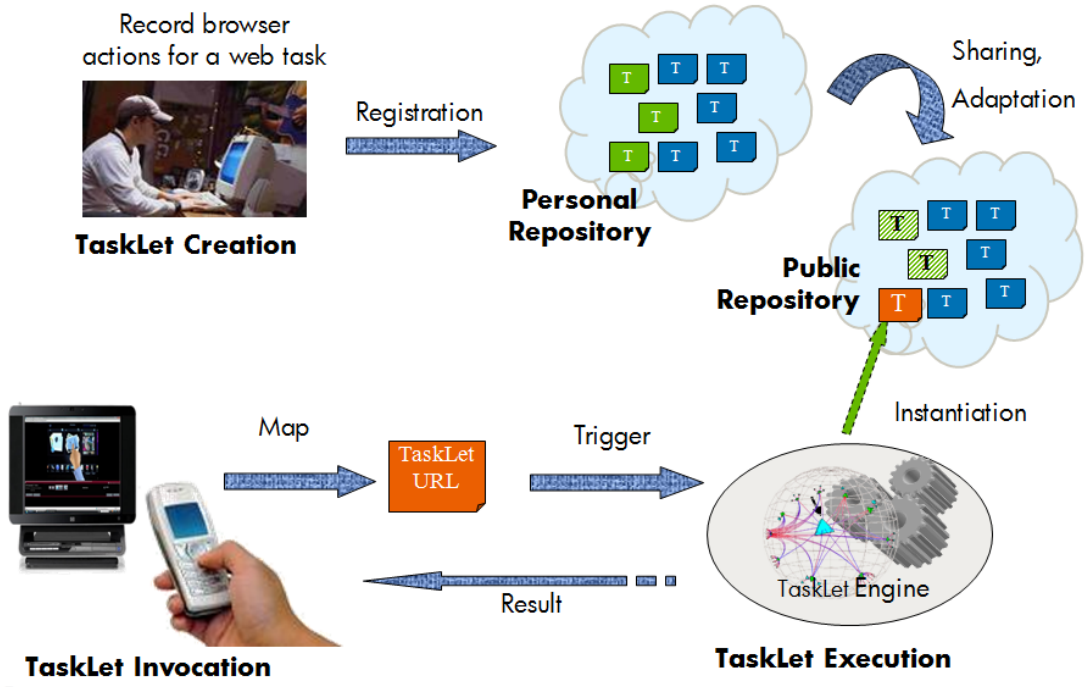

Figure 1: TaskLet Life Cycle

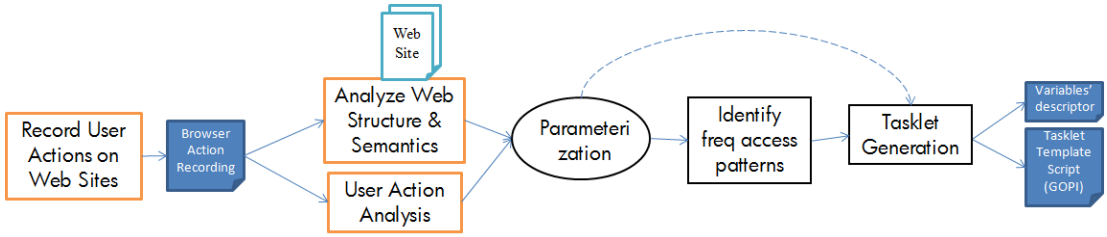

Figure 2: Key modules of Tasklet authoring

This Browser Action recording (BAR) file is analyzed along with the content and structure of source web sites to create a generalized version of the web task. This process of generalization is also called **parameterization**. One of the results of this generalization is to identify the potential input parameters for the task and create the **Tasklet Template Script**, a new script which contains browser actions with variable references replacing sample inputs used during recording. This newly generated Tasklet Template script (TTS) has the sequence of web interactions modelled using a special web instruction set. We call this scripting language as **GOPI (Goal Oriented Personal Interaction)**, more details of which is given in section 3. The technique of using web recording as a means of programming is known as Programming-by-Demonstration or Programming-by-Example and has been used in multiple domains [8]. We have adapted this technology for web interactions and in the process derived a new way of simplifying personal web interactions.

The authoring tool interface supports explicit Record and Stop buttons for the user to specify the beginning and end of the interaction needed to perform the web task. Additionally, an implicit recording mode is also supported wherein, the browser plug-in continuously captures the web interactions done by the user and analyses it to determine potential candidates for a Tasklet. The modelling of the task with generalized web instructions enables us to detect frequent web interactions using frequent pattern mining techniques using standard techniques [31].

Tasklet Execution

As mentioned earlier, the Tasklet Execution Engine is responsible for executing the Tasklet every time the task is replayed. When a Tasklet is invoked from user's own device (widget invocation for example), a version of Tasklet Template Script is instantiated with the new inputs given by the user and is interpreted on the Tasklet Execution Engine. The Tasklet Execution Engine contains an execution sand box (Web Virtual Machine) that interprets the different GOPI instructions in the TTS

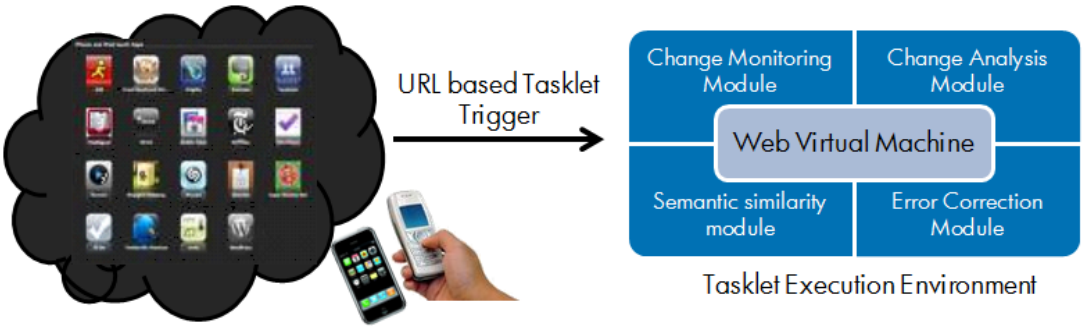

Figure 3: Tasklet Execution Engine

and provides the browser-less behaviour of the web interaction. Figure 3 gives the key modules of the Execution Engine. Every Tasklet in the repository is addressable with a URL and accessing this URL will trigger the execution of the corresponding Tasklet script on the execution engine. Since web sites change very often, the web change analysis is an important module that compares two versions of the website, one version that was available at record time versus the second available at replay time. The Web Virtual Machine interprets the GOPI instructions and takes appropriate actions on changed web sites to ensure correct behaviour of the Tasklet.

Tasklet Repository

The Tasklet Repository is a store of all the Tasklets created by users. As mentioned earlier, a recorded browser action is registered with a repository service to auto generate a Tasklet (using the process mentioned under Tasklet Authoring). Further, a unique Tasklet URL is allocated to enable simplified triggering of Tasklets from multiple/diverse client devices. When the Tasklet is invoked from a mobile device, the actual execution of the Tasklet is done by a special cloud service, and only the result of the Tasklet is sent across in a simple XML file. A generic mobile client that understands this Tasklet execution protocol can be provided for any device, hence enabling Tasklet invocation.

We have developed different sandbox applications for MS Windows desktop and multiple mobile device to support this protocol. The Windows application is a standalone version that takes the input specification of a Tasklet and the associated GOPI script and interprets the script locally as the Tasklet execution engine is embedded within this standalone application. In effect, one can 'download' an auto-generated widget application for every Tasklet in the repository for desktop-based execution. We have created similar sandbox applications for multiple mobile platforms (Google Android, Windows ME and WebOS). In the mobile versions however, the application leverages the Tasklet execution engine as a cloud service for small footprint. Figure 4 shows a screenshot of our Tasklet Repository that can be used to download different device-specific widgets corresponding to every Tasklet.

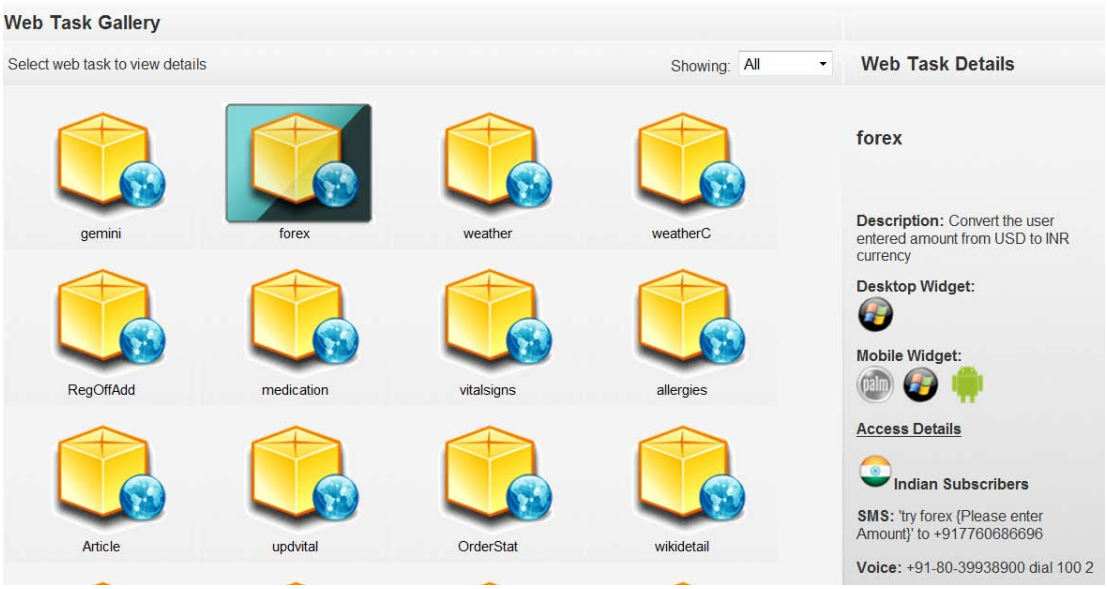

Figure 4: Screenshot of Tasklet Gallery showing downloadable widgets

We now look at more details of the authoring and execution environment, with a particular focus on the proposed Tasklet scripting Language used during Tasklet Authoring and Web Change Handling during Tasklet Execution in the following two sections.

3. Tasklet Scripting Language

We propose a Tasklet Scripting language called GOPI (Goal Oriented Personal Interaction). GOPI is meant to capture multiple models of a web interaction in order to enable the resilient Tasklet execution engine despite website changes. Since Tasklets are programmed by browsing, the GOPI script is actually generated by the Tasklet authoring module. An expert user can however edit it, as needed. In this section, we describe the scripting language and the steps involved in producing the same using examples.

The GOPI Script

GOPI script represents the web actions similar to processor assembly instructions, but for web sites (which are treated as web service processors). This **instruction set over websites** consists of typical actions supported by web browsers today, such as clicking on hyperlink, entering a form element, submitting a form and so on. The instructions in the script have an **opcode** and one or more **operands** – with a format of <opcode> <operand>*. The opcode represents a web action (HYPERLINK, INPUT, SUBMIT). An operand represents the specific instance of the action (e.g., details of the hyperlink clicked) and is captured in multiple modes (such as, *what is the URL of the hyperlink, where was the anchor tag in the DOM tree, what is the text that appears on the hyperlink*). Operands represent several attributes of the instance of the action in the form of name-value pairs. The operands of instructions can have variables – and these variables are identified by the authoring service during the parameterization step.

As briefly mentioned in the earlier section, all the browser actions during Tasklet authoring are logged in a recording file (Browse-Action-Recording, BAR). We have used an off-the-shelf browser plug-in called iMacro for this, as iMacro tool captured the browser recording in a form that closely resembled the format that we wanted to use. String constants have spaces replaced with <SP>.

Let us consider an example web task of extracting the horoscope of Gemini Sun Sign from msn.com website. The browser interaction for this horoscope is (a) to go to msn.com, (b) select the hyperlink named “Horoscope” and (b) on the next web page, select “Gemini” among all other Sun Signs and then on the third page, (c) to highlight the content of interest and extracting it with just a double click. The corresponding browser recording logged by the off-the-shelf macro recorder is shown below. Clearly, this is very crude and describes the task in the form of simple html tags.

```
1. URL GOTO=http://www.msn.com/index.html
2. TAG POS=1 TYPE=A
3. 'New page loaded
4. TAG POS=2 TYPE=A
5. 'New page loaded
6. TAG POS=7 TYPE=TD ATTR=CLASS:standard11&&TXT:* EXTRACT=TXT
```

The authoring module then converts this raw browser recording into semantic instructions consisting of an opcode followed by one or more operands. As any processor instruction set, the opcode determines the type of operation to be done and the operand gives enough information about the specific action. The GOPI script for the above msn horoscope example is as follows.

```
1. HOME URL http://www.msn.com/index.html
2. HYPERLINK txt=Horoscope POS=1 TYPE=A
3. 'New page loaded
4. HYPERLINK txt={{VAR1}} POS=2 TYPE=A
5. 'New page loaded
6. EXTRACT REGEX=*Today*" POS=7 TYPE=TD ATTR=CLASS:standard11&&TXT:*
```

As seen, in the modified version, the opcode signifies the type of operation (HOME, HYPERLINK, etc) and comes with additional semantic information given in the operands such as txt=Horoscope on line 2. Line 3 and 5 are comment lines. Line 4 shows the use of a variable, VAR1 instead of the hyperlink named with text Gemini. Finally, line 6 describes the content to be extracted with opcode EXTRACT. The POS and ATTR fields describe the structural position of the selected content, whereas the regular expression (*Today*) gives additional information about the content to expect. As we describe in the next section, these multiple models of the opcodes help us in handling changes. The regular expression is used only as a guideline to identify the content to be extracted while interpreting the EXTRACT opcode. Usually, the regular expression does not describe the content itself but some static headings nearby and the content to be extracted is then described relative to these headings. For this, a visual hierarchy analysis of the web page is done using VIPs algorithm [Cai03].

```
1. HOME URL=http://lib.hpl.hp.com/
2. HYPERLINK txt=ACM<SP>Digital<SP>Library POS=1 TYPE=A
3. INPUT POS=1 TYPE=TEXT FORM=NAME:emp ATTR=NAME:UID CONTENT={{EMP_ID}}
4. SUBMIT POS=1 FORM=NAME:emp ATTR=NAME:ACTION
5. INPUT POS=1 TYPE=TEXT FORM=NAME:qiksearch ATTR=NAME:query CONTENT={{TITLE}}
6. SUBMIT POS=1 TYPE=IMAGE FORM=NAME:qiksearch ATTR=NAME:Go
7. EXTRACT HREF regex="*pdf*" POS=1 TYPE=A
8. SAVE {{EXTRACT}} FOLDER=C:\  FILE=ACM_paper.pdf
9. SET !EXTRACT Successful<SP>Download.<SP>Check<SP>C:\ACM_paper.pdf
```

Another example script to elaborate the support for variables is shown above. This Tasklet takes the name of an ACM research paper and returns the URL of the paper, after required authentication. On line 1, the user goes to HP Labs library page, on line 2 clicks on a hyperlink named ACM Digital Library, and on line 3 she gives the authentication information (employee id) needed for accessing the digital library, and submits a form. Next on Line 5 is the interaction with ACM web page, where she inputs the title of the paper in a form and submits (on line 6). On Line 7, she extracts the URL of the research paper and requests for saving it in a local folder. Line 8 is the SAVE instruction. Line 9 sets a return value for the Tasklet that shows successful download of the file. In the above example, the variables EMP_ID and TITLE represent the employee id (used for authentication) and title of the paper to be searched, respectively. The lines 8 and 9 are inserted after the browsing session wherein the user would have indicated that the extracted content is an URL (the user uses Shift-Cntrl-Click instead of Double-Click).

Variables for Web Task Generalization

Typically, many tasks require input parameters and several of these input parameters are usually same across runs of the task, while some need to be different in different runs. We introduce a concept of free and bound variables for a Tasklet. A **free variable** is one that changes in every run of the task and needs the user to provide inputs (for example date of travel). A **bound variable** is same for every run of the task (username). A bound variable takes a predefined value that was given during Tasklet execution and a free variable appears in the final widget interface as an input box. This classification of variables is utilized when generating the corresponding widget. Each Tasklet variable is marked as either bound or free by the Tasklet author. To honour user's privacy needs, towards the end of the authoring session, the user is asked to identify those variables that she prefers not to include with the Tasklet while sharing (for e.g., passwd), which become free variables of the Tasklet.

In the second example of ACM research paper, the variables are EMP_ID and TITLE representing employee id (used for authentication) and title of the paper being searched, respectively. We chose to make EMP_ID variable to be bound with the value of a valid HP employee id and TITLE to be free. So, the generated widget asks just for the title of the paper to be searched in every run. Please note that the binding status of these variables can be changed even after the Tasklet is published on the gallery. A variable can, for example, be free at the time of authoring and be bound the first time a user uses the Tasklet as well. Such late binding of variables enables Tasklets to be shared by masking private data.

Parameterization

The operands of instructions can have variables and these variables are automatically extracted by the authoring service during the parameterization stage. During the action recording for the web task, the user can give multiple inputs such as name, date, address, etc. As mentioned earlier, the BAR file is parsed and analyzed by the Tasklet authoring module to remove these instance-level detail to create a Tasklet template. In this Tasklet generalization step, all values in the web task that are potential inputs to the task are marked as variables. This parameterization for variables is done for explicit inputs provided by the user as well as for alternate hyperlinks. For explicit inputs, the HTML tags on each of the web pages traversed during Tasklet recording are used to identify the potential variable. To determine if hyperlinks can be inputs, an analysis of the web pages (called as *sibling analysis)* is done to recognize hyperlink tag names such as Gemini, Aries, etc., as potential user inputs in the instruction.

After the parameterization step, the Tasklet author is given a list of possible input parameters for a specific Tasklet. For example, during this Tasklet creation, the user will be shown a list of possible input parameters to select from (do you want to change the flight 'date', 'destination' in every run?), and asked to provide the privacy settings of each of the input parameters. Based on the author's privacy preferences, the Tasklet can be configured with free or bound parameters.

Once the GOPI instructions corresponding to a Tasklet is created and registered with the Tasklet repository, the key problem is about handling changes to web sites, which is described next.

4. Web Change Handling

One of the interesting challenges of executing Tasklets in an environment that is different from the creation environment is that the execution may happen at a time instance that can be several days or

months after the creation time. Meanwhile, the websites on which the Tasklet was authored could have changed and just naïve replay at the time of its re-execution will not sufficient. Studies show that on an average 50% of the web pages of any website change within 50 days [32]. Therefore, unless this ephemeral property of the web is efficiently handled by Tasklets, they may fail to perform or replay the user's task. We now describe our novel approach to handle website changes.

Our key approach is to handle web site changes based on the semantics of the interaction. This is the reason that we capture the interaction as instructions over websites and define a virtual machine to interpret individual instructions in the context of the changed website. As described in the previous section, different actions use different instruction opcodes and the Web VM (Tasklet execution engine) knows how to handle changes to the web page with respect to the specific opcode of the instruction. The operands of the instructions either can be syntactically bound (like the absolute URL in HOME instruction) or semantically bound (HYPERLINK with text on the hyperlink in addition to than the hyperlink URL). Since we model each operand of the instruction in multiple representations, the Web VM can pick and compare the required model for a specific type of web change. For example, it can handle the HYPERLINK instruction by giving priority to the matching text (label) on the hyperlink as opposed to the URL in the anchor tag, if the URL in the new version of the web site points to a different location.

It is also many times possible to take care of semantic changes to the text of the hyperlink as well - where the text on the hyperlink changes to a semantically equivalent text. For example the hyperlink 'horoscopes' may be changed to the hyperlink 'astrology' and we will detect the same using ontology-based semantic similarity matching techniques that we had earlier developed [15,17]. Thus the execution engine virtualizes the changes to the websites and behaves like a Web Virtual Machine that interprets Web Interaction Instructions defined in GOPI script.

Next thing to note is that not all changes to websites will affect the task and it is not possible to exactly predict when a website changes [16,18]. So one way to handle this unpredictable nature is to check of there is change in websites before the Tasklet execution, and take corrective action immediately (whenever possible). The Web Virtual Machine shown in Figure 3 takes care of such changes to web pages on which a Tasklet is authored. Detecting and representing these web site changes is non-trivial and is described in the next subsection.

Modelling Web Site Changes

A website may change in multiple ways. In order to detect and characterize the type of change that occurs between two different versions of a website, we first classify these different types of web changes into structural, content, attribute, and semantic changes. Chawathe et al [24] propose that one could use edit strings to represent changes between structured documents (like strings). They define edit operators such as insert and delete for XML data. We extend their approach, and include additional operators such as MOVE, SHIFT as well as a semantic similarity operator. We then use a novel difference finding algorithm to detect the type and form of web change. Our algorithm computes the edit distance and edit string of web-change operators with respect to interesting fragments of the HTML page as opposed to complete HTML tree as done in the prior solutions. We detail this technique in the rest of this section.

The HTML DOM representation of the two versions of the web pages, the one that existed at the time of Tasklet creation and the second that is seen at the time of Tasklet execution, are used to compute website changes. In order to derive this edit string, we define the following seven edit operators; six for structural changes plus one semantic change operator. They are Insert, Delete, attributeChange, Update, Move, Shift and Semantic. Insert and Delete operator insert and delete a sub-tree in the original DOM tree, while AttributeChange modifies or adds a new attribute for a node of the original DOM tree. Update operator modifies a value of a leaf node in a sub tree of the DOM tree and Move represents a change in position of a sub tree (moved from one parent node to another). The Shift operator is a special type of Move in which a sub tree changes its position but the path from the root node to its parent remains the same. For example, if two rows in a table are interchanged, it will be a Shift operator. The Semantic operator gives the semantic similarity between two words chosen to be compared.

The operators Insertion, Deletion, Update and AttributeChange are **syntactic** in nature and can be computed by an analysis which treats DOM trees as simple labeled unordered trees. We use a lexical similarity metric (that measures the extent of overlap of words between the two strings) to determine the extent of content change in a node, so as to distinguish between an update and an insert. If the lexical similarity is more than a threshold (say, 50%), then the node is considered to be the result of an Update operation. Otherwise, the edit string would contain an Insert and a Delete operator to represent the change.

The Move and Shift operators represent **structural** changes. The Move operator signifies a position change of a node, where the whole subtree of interest is moved to be rooted at a different parent in the current copy of the web page. Shift is a special type of Move instruction where the subtree moves to a sibling's position; and in this case, the signature to the parents of the two matching nodes will be same i.e., the path from the root to the two parents will be the same. The Semantic operator is identified when the visual text of the content being compared are lexically dissimilar but, the two words may be synonyms or of similar meaning (eg., ``horoscope" and ``astrology"). This involves computation of text similarity measures [15] that consider the semantics of the content in the two nodes being compared.

Rather than looking for an overall difference between two DOM trees of different versions of a HTML page, we identify the DOM element of interest in the first tree and search for the best matching similar node in the target HTML DOM, that way, we will be able detect the fragment even if it has moved to a different location. Since we model the fragment in multiple modes (as exemplified for Hyperlink instruction earlier), if at least one of the models has not changed, the Web VM will be able to detect and correct the change. A detailed algorithm for detecting these change operators is given in Appendix A.

Handling changes to web sites

Since Tasklet uses an instruction set for execution, there are two possible ways of implementing the corrections. The correction for a detected web change can be performed by either modifying the recorded Tasklet instruction or by ensuring that the Web Virtual Machine interprets the recorded instruction correctly. In our design, the Web Virtual Machine interprets the instruction in the changed context and additionally modifies the Tasklet instruction to work on the changed web page to take care

of future executions of the Tasklets. As a final design point, this change detection module can be used in two ways - either as a part of Web Virtual Machine (reactive behaviour to web changes) or can

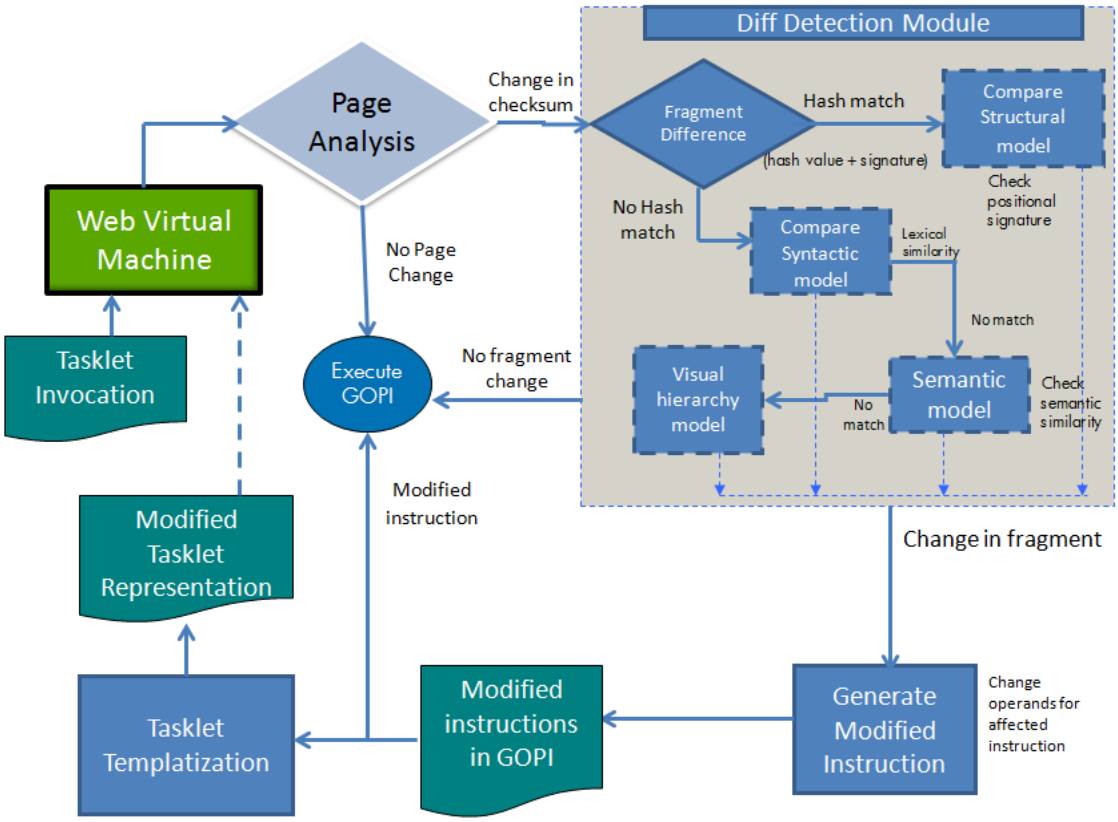

Figure 5: An abstract flow between modules for Change Detection

execute independently within a website monitoring framework so that the instruction set is updated as soon as the corresponding website changes (proactively).

Figure 5 shows an abstract flow diagram highlighting the key modules of the Tasklet execution environment participating in the change handling aspects. When a Tasklet is invoked, the web virtual machine is instantiated with the GOPI script corresponding to the Tasklet. The execution engine also has access to a copy of the original web pages on which the Tasklet actions were recorded. A simple checksum on the content of the web page is done first to see if there are any textual changes to the webpage. If there is a change in the HTML text, then the Diff Detection Module is triggered. This module uses the algorithm described in the Appendix to detect changes with respect to a fragment. The fragment of interest is identified in the form of an XPath expression, based on the operands of the instruction. If a match for the fragment is found in the new page, then its positional signature is used to detect moves or shifts within the tree. Once such a structural change is identified, the operand of the instruction is modified to reflect the new position of the fragment.

If there is no exact match for the fragment in the DOM of the new web page, then lexical similarity is used to determine if there was an update in the content of the fragment of interest. This

update is revalidated using the regular expressions that represent visual hierarchy of the content of interest. In effect, if a change in the fragment is detected, the instructions for the Tasklet are modified appropriately and the execution of the Tasklet continues. The modified script is also saved in the repository for future use.

In addition, one needs to handle HTTP errors and exception in web pages (user authentication related, session related etc) intelligently and also, provide support for captcha. For this, we have a simple classifier for detecting whether a web page is an error page and if so, we extract the content of the landing page and display to the user. In order to deal with captcha, we have developed an interaction flow protocol which can actually display the image and ask the user to break the captcha, as opposed to the usual approach used by bots to illegal way of hacking th captcha image.

NP Completeness

Analysis of website changes and handling of those changes is a hard problem. We now briefly look at a formal problem formulation of this Tasklet Resilience problem to appreciate the complexity. As described earlier, Tasklet is a sequence of web interactions required to perform a given task on the web. A Tasklet is recorded initially by the user while performing the task, which can be replayed later.

Let the Tasklet when recorded be $T = \{< P_1, a_1>, <P_2, a_2>, ..., <P_n, a_n>\}$, Where, P_i is the web page on which the action ai is performed. Each P_i may not be distinct as there can be multiple actions on a single page. For example, the action of filling up a form with three fields takes four actions (three actions for filling each field and one for submitting the form) on the same page. At some later stage when the Tasklet T is replayed, due to the changes in the web, it may be transformed to $T' = \{< P_1', a_1' >; < P_2', a_2' >; ...; < P_n', a_n' >\}$ where P_i' is the changed web page on which the modified action a_i' is to be performed so that the Tasklet works as expected. We say that the Tasklet T is resilient to changes in the web, if while replaying the Tasklet, each action a_i can be performed on the modified page successfully.

We have proven that this Tasklet resilience problem is NP Complete by reducing it to the problem of computing the editing distance between two unordered, labelled trees that is known to be NP Complete [14]. The editing distance problem asks the question of whether there exists a Map between two trees such that the edit string satisfies a cost bound. The key intuition for proving the equivalence of the two problems, is to define the Tasklet resilience problem as a sub-tree finding problem as below.

For this, we represent each action a_i with two parts. Say, $a_i = (s_i, ac_i)$, where, s_i is the unique segment within the DOM representation of page P_i on which action ac_i is performed. In the modified page, the segment on which the action a_i is acting may have been modified, shifted or removed completely. So the main challenge in Tasklet resilience is to be able to find the new segment in the modified page, and then identify the corresponding action ac_i'. We can say that the Tasklet T is non resilient if for any action a_i, we cannot frame such a modified action a_i'.

We too define a cost function cost for edit operators: e -> R+ such that the total cost of Tasklet resilience is directly proportional to the number of steps required for modifying the action that counters the change. Therefore, changes that cannot be handled take up an infinite cost and changes that do not affect the action get a zero cost.

Let diff (s_1; s_2) denote the total cost of converting segment s_1 to s_2 through a sequence of edit operations. Then if E = e_1; e_2; ...; e_k is the sequence of edit operations that convert segment s_1 to s_2 on the sequence of web pages, diff(s_1, s_2) = Σ cost(e_i).

Formally, we define the problem of Tasklet Resilience as:

T = {< P_1, a_1 >; < P_2, a_2 >; ...; < P_n, a_n >} where a_i = (s_i; ac_i), is resilient *iff*, for each segment s_i associated with action a_i, there exists a segment s_i' on the modified page P_i' such that diff (s_i, s_i') ≤ k

Therefore, the Tasklet resilience problem is equivalent to finding a web page segment in the modified page such that the edit distance between the two segments is less than a given bound k. The bound k is assigned a value such that it incorporates only the changes that can be handled and is less than the cost of editing all leaf nodes of the segment. The complete proof of the equivalence of the Tasklet Resilience problem to Editing Distance problem is however, beyond the scope of this paper.

Now that Tasklet Resilience is an NP-Complete problem, we have to take care of that in our design. One possibility is to adapt the approximation algorithm for computing the best Editing Distance that was proposed by the same authors [14] to find the best sequence of Tasklet changes needed to enable Tasklet Resilience. However, we found that the most common types of web changes seen in practice could be characterized by a single change; and so, we decided to design our system to handle only these typical changes initially.

5. Related Work

This section gives a broad survey of the research literature on end user programming for web-based applications and details approaches that use programming-by-example model for the web.

End User Programming

End user programming has evolved as an interesting research topic for both programming language experts and human computer interaction (HCI) specialists. The goal is to move from 'easy to use' to 'easy to develop' paradigm in order to enable common people to customize information technology for their personal use. Given that different persons may need to perform different tasks at different points in time, it is important to be able to give personalized computer experiences. The belief is that having a group of programmers to develop custom applications for each and every need of an end user does not scale. So, the main goal of End User Development is to empower end users to develop and adapt systems themselves.

One of the earliest known works that used the term 'end user programming' is a book by Bonnie A Nardi from MIT [25] where the author analyzes technical as well as social and cognitive issues of end user programming. She stresses the need for task-specific programming languages and visual application frameworks and describes the need in the context of spreadsheets and AutoCAD. The end users she considers are people such as chemists, teachers, librarians, architects, accountants who are interested in making serious use of computers but are not interested in becoming professional programmers. Interestingly these users still remain as targets of end user programming even today. In addition, a class of enthusiastic students and rapid application developers will potentially be very interested in non-professional programming.

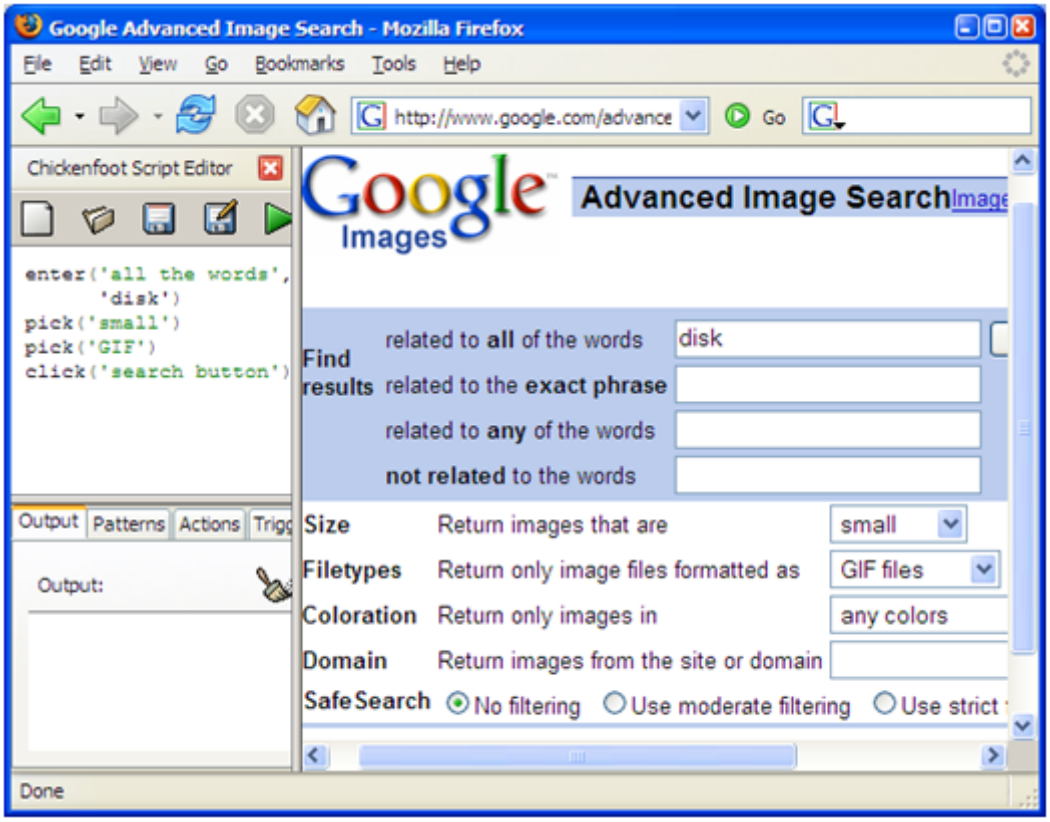

Figure 6: Screenshot of ChickenFoot Browser Plugin from CSAIL

Programming By Example is a popular means of supporting end user programming, as exemplified by the use of 'macro programming' in Microsoft Excel. Macro recording and replay is a well-known technique used in editors, spreadsheets, program automation and like. Web testing tools (like WebVCR, iMacro, HP Mercury QTP), also provide a record and replay facility over websites. Though we use this familiar metaphor of record-and-replay to create a program, the techniques used are somewhat different. These macro recorders have a very tight coupling between the recording and replay modules – and hence cannot work on mobile devices. Further, a lot more processing is needed to make a browser recording into a program and to accommodate changes in the inputs the next time the same task is repeated – bringing the need for generalization and parameterization. Though some web testing tools do handle website changes by recording both the visual co-ordinates as well as DOM information, our treatment of website changes is somewhat unique as we define an instruction set for the web and try to handle most common type of changes in a virtual machine.

Programming by Example for the Web

One of the early systems that used Programming By Example on hypermedia systems was Eager [26] which automated repeated graphical user interface tasks within a user's session on a HyperCard

GUI application on Apple Macintosh systems. HyperCard system was a hypermedia system similar to the Web and was very popular for writing rapid applications before the World Wide Web was proposed. In fact, the first web browser Mosiac (Netscape) was inspired by the HyperCard system. Eager system enabled repeated use of sequence of HyperCards, and was written in Lisp and ran as a background application watching and recording every interaction of the user with any application. These actions of button click and selection were recorded and replayed. It also had a capability of generalizing the actions to some extent, by measuring the 'similarity' of the action to the previous action (such as a button selection, followed by another button selection). The good part of Eager was that it had access to higher level structures such as buttons, lists as opposed to screen co-ordinates, as it was exposed to the structure of Hypercard GUI on which the action was performed, which helped in its generalization.

For the present day Web, there are many Web Application testing tools (like WebVCR, iOpus iMacro[1], HP Mercury QTP[2] that provide a record and replay facility over websites. Since these macro recorders are used to test web applications developed by the same users, they rarely handle any changes to the UI of the web application. When the user interface of the web application changes, the users' simply record new macros for testing the new interface. These systems also have a very tight coupling between the recording and the replay modules, wherein the recorded action is stored in an internal data format which is understood by the replay module. There is no support for generalization during recording as well, though some systems (like iMacro) do allow people to write Java programs that can substitute parts of the macro scripts with different inputs. These systems also do not generate applications as such. So, these systems cannot be directly used to develop web applications or web widgets (which is the intent of our proposed framework).

There are also browser plugins that help users automate some common tasks on web pages. A feature in Windows 7 called WebSlices provides segmented book marking facility where the user can bookmark a portion of a page and access it through Internet Explorer. A similar tool SnipIt[3] was developed under the guidance of Prof Alan Dix from Lancaster University. One of the early efforts at creating such a system with PbE support was Internet Scrapbook [27] which allowed creation of user dashboards from snippets of information on web sites. Their system automatically updated these snippets for live view. A more recent tool along these lines is OpenKapow RobotMaker[4] software which uses PbE for enabling its users to select interesting text snippets and makes the dynamic version of that content available as an RSS feed. Tools that enable users to personalize web page views through browser-side manipulation also exist.

For example, Greasemonkey[5] is a Firefox plugin that enables users to make easy modifications to the web browser so that the appearance of web pages is as per their taste. However, Greasemonkey scripts need to be written in Javascript for modifying the HTML structure (DOM).

[1] http://www.iopus.com/download/imacros
[2] http://en.wikipedia.org/wiki/QuickTesProfessional
[3] http://www.snipit.org/
[4] http://kapowsoftware.com
[5] https://addons.mozilla.org/en-US/firefox/addon/greasemonkey

An award winning MS Thesis by Michael Bolin from MIT, a student of Prof Rob Miller [28] describes an interesting system called ChickenFoot that enables End User Programming on web pages. ChickenFoot[6] was written as a browser plugin and provided easy-to-use APIs with semantics of web interaction. It enabled programmers to execute simple commands on the web page, using a scripting language called ChickenScript, that could be embedded in Javascript. An example ChickenScript code and the interface provided to the user is shown in figure 6, a screenshot taken from the author's website [CSAIL]. ChickenScript had some easy-to-use API commands like click("Google Search"), uncheck("Remember Me"), etc., that could be called from a Javascript program. An interesting aspect of this work is that it was able to handle some changes to the web pages as well. For this, the script runtime required that the user use visual labels on the web page for programming an action. Some most common variants of the visual labels were supported using syntactic rules for transposing words[29]. An example mentioned in their paper shows that the users could use the label "websearch" when the caption on the web page read "Search the Web".

Another interesting work that extends the above described ChickenFoot system with Programming-By-Example is Co-Scripter (formerly called Koala [10]). It is an end-user programming environment for the Web, which records, replays user interactions within a web browser, and uses a nice natural language-like script that is editable by humans as well interpreted by machines. An example script for checking flight arrival times and status from American Airlines is given below:

go to "www.aa.com"

enter your "Flight Number" (e.g. 144) into the "Flight Number:" textbox

click the fifth "GO" button

The focus of Co-Scripter is on human understandable natural language like scripts that can be contributed within communities (wiki's) to share the how-to knowledge [5] of working with web pages. It has a Firefox plug-in that enables users to record scripts and further modify them, and replay the scripts within the browser. They use the word sloppy programming approach to interpret these English like scripts employing the syntactic similarity technique used by ChickenFoot[29] described earlier. However, other types of web changes were not handled by this system. Moreover, the recording and replaying systems were through a web browser and so these scripts could not be accessed from mobile devices, more so from low-end phones.

Another body of work that does in-depth analysis of Programming by Example is under Prof Henry Lieberman, the creator of Logo! In his book on PbE [Lieberman01], Henry points out the following three key issues in PbE:

- The generalization problem: How to describe actions and data ? How to record? How to represent the recorded program? How similar can the actions be during replay?
- Feedback: How does the system show the user what it has learnt? Should the user see the recorded program?
- Advice: How can the user modify what the tool has learnt? Should the user interact directly with the recorded program?

[6] http://groups.csail.mit.edu/uid/chickenfoot

An interesting tool developed by this team is Creo [7] that the authors describe as a *goal oriented web browser*. The tool automatically tries to understand the goal or intent of the user while she is browsing based on the web content. Once the goal is known, it is used for generalization. The tool also includes a technology called Miro, that is a semantically enriched data collector, that learns the type of data based on an example (*is the string an email, phone number, or date?)* which helps in understanding the goal further. To use the tool, users explicitly hit the Record button before doing a web action. Now, when the user selects `Diet Coke', Creo automatically detects that it is a food item – and the generalized program is stored. They use semantic information from MIT's ConceptNet and Stanford's TAP to enable this generalization (or parameterization). Creo was implemented as an extension to Internet Explorer 6. However, there is not much information on how they handle changes to websites during replay. That work is also described in the context of a single page and for information retrieval purposes only.

In general, end user programming for the web is usually supported in two different forms: either by Programming-by-Example or by simple scripting that captures the semantics of web browsing in some form. Our approach is unique in that we use a combination of these two approaches but target the semantic instructions more at the machine so that we can handle website changes. We also decouple the recording and replay sessions that is typical of macro recorders to make it more flexible so that we can enable mobile as well as cross-device simplified web experience.

The key difference between our system and simple PbE systems is that they assume that the 'replay' environment to be very similar to the environment in which the program was 'recorded'. While the web content on which our system works, changes most often – probably due to dynamic content, advertisements, personalization, and other factors. So, the proposed end-user programming system for developing web-based applications using PbE addresses the difficulty of replay despite the changing behavior of the websites (and web pages). Additionally, our system handles the generalization of the newly created web application to capture the intent of the user in the form of a Web Instruction Set. Our Web Instruction Set (called GOPI) is interpreted by our Web Virtual Machine which enables the action recording to be executed not only on modified websites but also on a different set of websites with equivalent services.

From a system design perspective, like ChickenFoot, Highlight and Co-Scripter, our system also uses a browser plug-in to enable recording of user actions. The action representation used by the above systems is human-friendly, whereas the representation used by our system (GOPI) is machine-friendly, but can be edited by humans, if necessary. This enables us to handle website changes more effectively with multiple models of a web site (syntactic, semantic and visual models) as opposed to their syntactic similarity technique or sloppy programming approach. Creo/Miro had the best generalization technique by use of sophisticated data detectors through semantic analysis for capturing the intent of the user. We however take a programming language approach for generalization, and replace potential user parameterizable inputs with variables, which can be either free or bound at the time of replay. We too use a semantic-web based approach for generalization (rather for error handling through generalization), but unlike Miro that uses semantic facts from ConceptNet, we use ontologies that help us understand topic hierarchy.

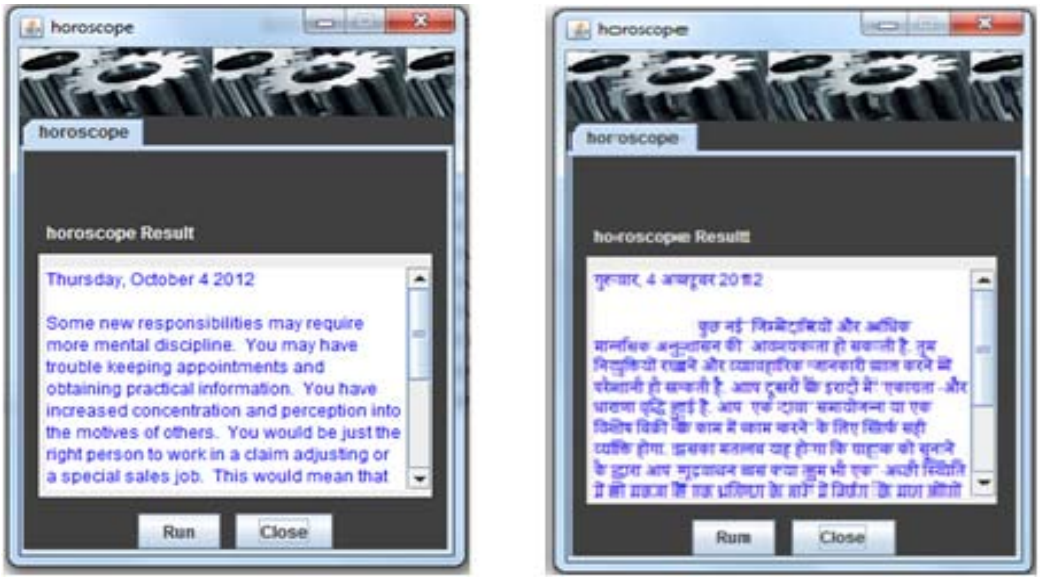

Figure 7: Generated Horoscope Widgets in English and Hindi

6. Implementation and Results

We have a working prototype of the Tasklet authoring and execution environment that can be used to generate widgets or client applications for Windows PCs, Android, Palm WebOS, and Windows Mobile. Additionally, we generate SMS and voice interfaces and provide smart gateways that can convert SMS and voice-based invocation to Tasklet invocation so that the solution can be used with low-end phones. We have hosted the authoring solution at http://siteonmobile.com and video demos showing the Tasklet authoring and invocation are available there. The Tasklet authoring and execution services are hosted on Amazon EC2 Cloud server.

The overall solution for rapid authoring of web tasks and accessing it from multiple mobile devices has received a lot of positive feedback from end users, business owners and even developers since it radically simplifies the way applications are created today. We are currently running a live pilot where the solution is being used by ten business owners to create Tasklets on their own websites and release the auto-generated widgets to their consumers [22]. As we support even SMS and voice interfaces to the Tasklet, the business owners find it a very valuable way of reaching their consumers who do not yet have access to the web (particularly in emerging market). The Tasklet repository hosts over 200 Tasklets that range from common information retrieval tasks to ecommerce tasks. Some example Tasklets we have are for fetching daily horoscope, performing currency conversion from a specific unit (USD) to another (INR), fetching latest gold price, stock price, uploading book reviews, checking status of an order placed, as well some health related retrieval and upload tasks. These services are being used by real consumers in a controlled fashion for the last 3 months. So far, we have not found a case where our Tasklet execution service has broken the execution despite changes to websites.

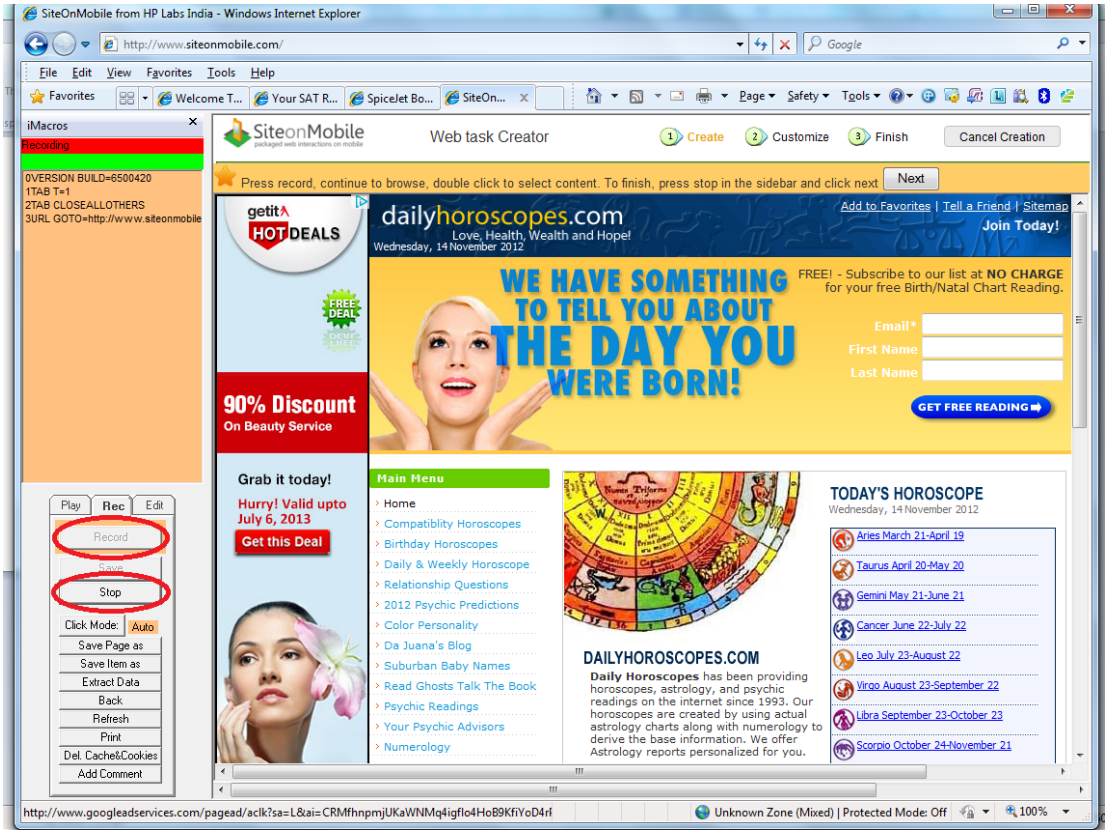

Figure 8: Tasklet Authoring Tool

For the authoring environment, we have leveraged the browser recording IE plug-in from iMacro (from iOpus) software which records in its own format (BAR). We have built services to register this BAR; analyze it, process it and to generate a TTS (a sample of the same was shown before). The sharing service can be used to share this URL over SMS or email; clicking this URL results in cloud-based execution of the Tasklet. The corresponding Tasklet-based application can be downloaded onto the client devices, upon which the standalone application with embedded execution engine for Desktops and cloud-hosted execution engine for mobile devices can provide a browser-less experience of the web task. On widget execution, user-given inputs are used for the unbound variables in the TTS and results are displayed. To support Always Responsiveness when the client is disconnected, the Tasklet output will be cached in the client application and refreshed at user-specified intervals.

Tasklet Authoring Experience

We now briefly describe the authoring experience. To show how one can easily author new widgets by just ‘doing it once on the browser’, we will choose the task of retrieving a person’s horoscope from a website she trusts. The user needs to first install our browser plug-in within her web browser. When she hits the “create new Tasklet” button on the SiteOnMobile portal, she is asked to

Figure 9: Changed MSN website over the years – works with Tasklets Engine

give the URL of the website. The website is loaded within an iframe of the portal and has a record button. She presses the record button explicitly and starts using the browser just like normal. She will navigate to MSN page, go to the Horoscope page, select her Sun Sign and extract content for my Sun sign. TO extract content, all the user needs to do is to double click on the content. That is it, the user "stops' recording.

The tool then takes the user to the parameter configuration page, where she will be an option to select which inputs needs to be used in every run or which are bound. After that, the system takes care of rest of the work of generating the application for multiple client devices that the user can download and execute. On execution, these widgets will show that day's Horoscope. Optionally, the user can select a language of preference (say a local language like Hindi). Then the output is automatically translated (using Google translation engine) to the language selected. Figure 7 shows a view of the Tasklet-based application that retrieves daily horoscope. Figure 8 shows a view of our authoring

environment showing MSN page loaded for recording with a side-record bar. The Tasklet repository to download widget was shown in Figure 5 earlier.

Figure 9 shows the changed MSN page between last year and now. Our Tasklet web change handling is so robust that a Tasklet recorded on the old MSN website works nicely even on the new MSN website. The horoscope link has been moved around, and in fact is under a drop down menu of "Lifestyles" now as opposed to direct hyperlink on the top banner earlier. There are also multiple layout changes to the web page but those do not affect the task of interest.

Live Trials in India

A Beta release of this solution was made on 8th July 2010 and the service received a lot of media coverage and excellent response from bloggers and tweeters, as an Internet solution for users with low-end phones. Over 200 trial users have been provided access to the service in just 2 months. Over 110 website owners (SMB) from India, US, UK, Germany, Singapore, Sri Lanka, and other countries, have shown interest to partner to deliver their services through this solution.

We are currently running a live pilot where the solution is being used by 10 website owners to create Tasklets on their own websites and release the auto-generated widgets to their consumers. Working on these customer trials with small-medium web-based business in India, we have created several interesting mobile applications and got a lot of technology insights and enhanced the solution for personalized content delivery, privacy and security. The feedback from the website owners has been very positive. They have found it a very valuable way of reaching their consumers who do not yet have access to the web, particularly in emerging market. Some Tasklets available for common use are to check the weather of different cities, price of gold (or other commodity from a specific website), currency conversion, accessing horoscope and so on. Among others, the solution has been applied to multiple public utility services as well [30]. These services are being used by real consumers in a controlled fashion for the last few months.

Interested readers are requested to visit http://www.siteonmobile.com for some demo videos. User can also request for a login access by sending an email to siteonmobile@gmail.com.

7. Conclusions and Future Work

In this paper, we presented a new way of simplifying access to web content and services. We proposed a concept called Tasklet that represents a user's specific personal web interaction. These Tasklets could be generalized to support variable inputs and hence could be shared with other users, enabling a viral propagation of web tasks. The method of rapid creation of Tasklets using programming-by-example enables even end-users to author their personal web tasks, enabling the scale needed to have the next generation "Web of Personal Tasks". The concept of Tasklets enables multi-device access to one's personal task. We hosted the Tasklet model in the cloud and enabled multiple clients to access it and hence multi-device access was possible.

One of the key problems that had to be solved to make the concept of Tasklets feasible was to handle changes to web sites without impacting the user experience. For this, we proposed a concept of Web Virtual Machine that would ensure correct execution of the Tasklet by interpreting the Tasklet script in Web Instruction Set, GOPI. Since GOPI instructions were semantic in nature, it was possible

to use action-specific heuristics to handle changes as long as we understand the exact change that has happened.

Our approach of using Programming-By-Example and an Instruction Set (and an Execution Engine) in the context of Web can trigger newer technologies for web-site agnostic programs (aka Java=processor-agnostic). Further, ability to address Tasklets as a first-class web objects (URL), separates the program from its interaction enabling familiar interfaces to be used for web transactions (SMS, voice, widgets, gestures,...). Further, these web tasks can be shared and annotated just like web pages today. Just like semantic web protocols have evolved to support the ``Web of Data", there is a potential to create new standards and protocols for the ``Web of Tasks". Overall, we believe that this solution can be potentially disruptive in how web can evolve to be in future – with user-created services, ability to capture web-flows (as first-class web objects) and browser-less personalized web consumption.

Going further, we can think about using natural language to invoke Tasklets. A system that understands natural language speech commands from the user, determines which is the best Tasklet to search in the repository and if none exists creates one on the fly and adds to the repository. Since GOPI script enables one to think about a web task in terms of sequence of interactions on different web pages, the problem of task creation if simplified. It only requires one to solve the problem for one a web interaction on just one web page, and we can extend the same to a sequence of interactions.

One of the primary problems with internet-enabled mobile devices is the disconnections to the Internet. Offline availability of web content is very important for a good mobile web experience. In order to cater to intermittent disconnections, we would like to implement a notion of ‘live Tasklets’ that use smart caching techniques on the client-side to perform appropriate synchronization and hence pre-execution of the Live Tasklets while the device is connected. Another future work is to enable composition of Tasklets. From a technical perspective, since the GOPI language has support for variables, flow of data between composed Tasklets can be done by considering variables as ‘slots’ filled by outputs of other Tasklets. It will be interesting to explore an appropriate user interface that enables an end user to do this composition. We are also exploring use of additional instructions in GOPI script to support conditional execution, loops as well as interactive behaviour to support complex web tasks.

8. References

[1] Creating Personal Mobile Widgets without Programming, Geetha Manjunath, S Thara, Hitesh B, Santhi Guntupalli, et al, Developer Track, 18th Intl Conference on World Wide Web, April 2009, Madrid, Spain.

[2] NetCraft Home Page, http://news.netcraft.com/

[3] Fixing web sites with Greasemonkey, McFarlane, N. Linux J. 2005, 138 (Oct. 2005).

[4] Automation and customization of rendered web pages. Bolin, M., Webber, et. al. In Proc. UIST ’05, MIT CSAIL

[5] CoScripter: Sharing ‘How-to’ Knowledge in the Enterprise, Gilly Leshed, Eben Haber, Tessa Lau, Allen Cypher, GROUP’07, November 4–7, 2007

[6] Koala: capture, share, automate, personalize business processes on the web. Little, G., Lau, T.A., Cypher, A., Lin, J., Haber, E.M., and Kandogan, E. 2007. CHI’07.

[7] A Goal-Oriented Web Browser, Alexander Faaborg, Henry Lieberman, MIT Media Laboratory

[8] Your Wish is My Command: Programming by Example. Lieberman, H. Morgan Kaufmann. California. (2001).

[9] Instructable and Adaptive Web Agents that Learn to Retrieve and Extract Information Tina Eliassi-Rad and Jude Shavlik University of Wisconsin-Madison, Computer Sciences Department, 2000

[10] Koala: Capture, Share, Automate, Personalize Business Processes on the Web, Greg Little1, Tessa A. Lau2, Allen Cypher2, et al, CHI 2007

[11] Efficient and effective web change detection, S. Flesca, E. Masciari, Data and Knowledge Engineering 2003.

[12] Webvigil:An approach to just-in-time information propagation in large network-centric environments,S. Chakravarthy, et al, 2nd Intl Workshop on Web Dynamics, Honolulu, Hawaii, 2002.

[13] WebCQ – Detecting and Delivering Information Changes on the Web, Ling Liu, et al, Georgia Institute of Technology, CIKM ’00

[14] On the editing distance between unordered labelled trees, K. Zhang, R. Statman and D. Shasha, Information Processing Letters. Volume 42, 1992.

[15] Computing Semantic Similarity using Ontologies, Rajesh T, Geetha Manjunath, M Stumpner, HP Labs Technical Report HPL-2008-87 (2008)

[16] B. Brewington, G. Cybenko, How dynamic is the web?, Proceedings of WWW2000, March 2000.

[17] Finding experts by semantic matching of User profiles, Rajesh T, Geetha M, M Stumpner, PICKME’08, Germany, Oct 2008

[18] Monitoring the Dynamic Web to respond to Continuous Queries, Sandeep Pandey, Kriti Ramamritham, Soumen Chakrabarti, WWW 2003

[19] Eager: Programming repetitive tasks by example, Allen Cypher, Apple Computer Inc, ACM 1991

[20] Intel MashMaker Home Page, http://mashmaker.intel.com

[21] OpenKapow Home Page, http://www.openkapow.com

[22] SiteOnMobile Home Page, http://www.hpl.hp.com/india/research/siteonmobile.html

[23] Semantic Analysis of Web Site Changes, Manjunath, G; Gupta, D, HP Laboratories, HPL-2010-123

[24] Meaningful change detection in structured data, S. Chawathe, H. Garcia-Molina, in: ACM SIGMOD 1997.

[25] A Small Matter of Programming Perspectives on End User Computing, Bonnie A. Nardi, 1993.

[26] Eager: Programming Repetitive Tasks by Example, Allen Cypher, Advanced Technology Group, Apple Computer, Inc, In Proceedings of CHI '91, New Orleans, Apr 28-May2, ACM, New York, 1991

[27] Internet Scrapbook: Web browsing by programming-by-demonstration, Sugiura, A. and Koseki, Y., (in Japanese), in: Proceedings of WISS'97, 1997.

[28] End-user Programming for the Web. Masters in Engineering Thesis, Massachusetts Institute of Technology, Michael Bolin, June 2005.

[29] Naming Page Elements in End-User Web Automation, Michael Bolin and Rob Miller, Workshop on End-User Software Engineering, ICSE 2005.

[30] Delivering Mobile eGovernance on Low-End-Phones, Demo Track, Thara S, Geetha Manjunath, et al, 13th International Conference on Mobile Data Management, IEEE MDM 2012, July 23-26, 2012

[31] Mining Frequent Patterns without Candidate Generation, J. Han, J. Pei, and Y. Yin, Proceedings of 2000 ACM-SIGMOD, Intl Conf. Management of Data (SIGMOD 00), pp. 1-12, May 2000

[32] Resonance on the Web: Web Dynamics and Revisitation Patterns, Eytan Adar, et al, CHI09.

9. Appendix A

Algorithm for Change Detection

Input: Tree T1, Tree T2, Mark Tag m

Output: Change Operators for interesting fragments of the tree

Step 0: *Initialize.*

- *N1 = { all the sub trees of T1 with tag m as a root }*
- *N2 = { all the sub trees of T2 with tag m as a root }*
- $N1' = N2' = N3' = \emptyset$

Step 1: *Mapping of the sub trees*

- *For ∀ sub tree (x,y) :* $x \in N1, y \in N2$

  if (hash(root(x) == hash(root(y)))

  / Both the sub trees are identical so Map x with y */*

  Then M' = M' + Map(x,y)

- *Compute N1' = { sub trees of N1 not mapped with any subtree of N2*

  $N1' \subseteq N1$; $N1' = \{x \in N1 : \nexists(x,z) \in M'\}$

- *Compute N2' = { sub trees of N2 not mapped with any subtree of N1*

  $N2' \subseteq N2$; $N2' = \{y \in N2 : \nexists(z,y) \in M'\}$

Step 2: *Model the structural changes for all mapped trees*

- *For ∀ pairs* $(x,y) \in M'$

  if (signature(x) != signature(y))

  Mark (x,y) as "Move"

  $M' = M' - \{(x,y)\}$

Step 3: *Match the sub trees in N1' and N2' and model the changes*

- *For ∀ (x,y) where* $x \in N1'$ *and* $y \in N2'$

  if (lexicalSimilarity(x,y) $\geq$ *configuredThreshold)*

  if (lexicalSimilarity(x,y) == 1)

  Mark (x,y) as "Other Changes"

  else

  Mark (x,y) as "Update"

  M"= M" + Map(x,y)

  checkAttributeChange(x, y)

  / Detecting "Move". */*

if (signature(x) != signature(y))

Mark (x,y) as "Move"

N2' = N2' \ {y}

else

Mark x as "Delete"

/ Mark the rest of elements in N2 as an "Insertion" */*

For $\forall y \in N2'$, Mark (x,y) as "Insert"

***Step 4**: /* Detect shifts */*

- *Remove all nodes that are inserted or deleted from the tree before checking for shifts*

  For $\forall x \in N1$, if (x is marked as deleted or moved) then T1 = T1 - x

  For $\forall y \in N2$, if (y is marked as inserted or moved) then T2 = T1 - y

- *For $\forall$ element $(x, y) \in (M' \cup M")$*

  Px = parent node of subtree (x)

  Py = parent node of subtree (y)

  while (hash(Px) == hash(Py))

  Px = Parent(Px) and Py = Parent(Py);

  if (Px and (PosSignature(Px) == PosSignature(Py)))

  Then mark as "No Change"

  Else Mark as "Shifts"

end.