Extracting RESTful Services from Web Applications

Bipin Upadhyaya, Foutse Khomh, Ying Zou
Department of Electrical and Computer Engineering
Queen's University
Kingston, Canada
{bipin.upadhyaya, foutse.khomh, ying.zou}@queensu.ca

Abstract— **The Web contains large amount of information and services primarily intended for human users. A Web application offers high user experience and responsiveness. A user performs different task, such as reserving flight tickets from a Web application. A task is a set of activities required for a user to achieve a goal. Similar tasks are often used in different websites. Therefore, facilitating their reuse would improve development productivity and ease maintenance of Web applications. However, designing a reusable Web application component is often neglected by Web developers due to the pressure for the time-to-market. To circumvent this limitation, we propose an approach to interactively identify tasks from Web applications and represent these tasks as services.**

Keywords- RESTful Service, Service Design, Service Extraction

I. INTRODUCTION

A Web application is coded in a browser-supported language such as JavaScript (JS) and combined with a browser-rendered markup language, such as the hypertext markup language (HTML). Web applications are popular due to the ubiquity of Web browsers and the possibility to update and maintain Web applications without distributing the clients. Web pages of Web applications are defined in HTML and represented using the Document Object Model (DOM). All client side interactions are realized with modifying JS of the DOM. The presentation of a Web page is handled by Cascading Style Sheets (CSS). To understand a Web application, a developer must be familiar with HTML, JS and CSS, and the interactions between them. A Web application is accessed through a Web browser running on client's machine whereas a Web service is a system of software that allows different machines to interact with each other through a network. Similar to the problems in early traditional applications, many Web Applications become legacy systems. Web applications are facing new challenges such as the integration of software provided by different organizations and the ability to create combined business scenarios. Web services provide system-to-system interaction and permit the implementation of business constraints using process control primitives to adapt the new challenges.

Web services are self-contained and self-describing. The two most used style architectures in Web services are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer). Compared with SOAP, REST is lightweight, easy to build and frequently favored by developers [2]. However, most of research [1, 3, 7, and 8] in migrating Web applications to SOA uses SOAP based services and require the analysis of the entire source code of Web application. Analyzing source code to extract reusable tasks is complex and error-prone, because there is no trivial mapping between source code and the page displayed in the browser [5]. Our approach focuses on the task that a Web application performs and abstracts the task as a RESTful service. A task is a goal specific functionality, such as searching for a restaurant, and reserving a table in a restaurant. A goal may be defined as a state of affairs that a user wishes to achieve; a task is the course of actions that a user goes through in order to achieve this state. In a Web application, the code responsible for a task is usually scattered between several files. It is often intermixed with code irrelevant for the extracted task. The code is written in different languages, such as PHP, SQL, JS, and HTML using different development paradigms. In this paper, we provide an approach to extract reusable tasks from a Web application by analyzing the client-side representation returned from a Web application. Similar to Insight [6] and FireCrystal [4], our work analyzes client side representation along with the change in URL and request parameters. However our work focuses on extracting task as a service. The extracted tasks are specified in terms of RESTful services and deployed through proxies accessing the original Web server and parsing its responses. The objective of our approach is to discover resources needed to accomplish a particular task. Our contributions in this paper are as follows:

1) *Provide a model to represent a task*. We identify Web pages that are browsed to accomplish a task and represent the functionality of the Web pages as a RESTful service.
2) *Extract logical data from data decorated with HTML for human users*. The extracted data encode semantic information making it easier for machine to invoke the service.
3) *Identify relations between tasks*. We automate the transition from one task to another and migrate such tasks as a RESTful service once the relations are identified.

The remainder of this paper is organized as follows. Section II introduces a meta-model for tasks. Section III presents an overview of our approach. Finally, Section VI concludes the paper and explores some avenues for future work.

II. MODELING A TASK

A task is a set of resource grouped in a meaningful way to accomplish a goal. Basically, identifying a task is centered

on the question: ***What will a user do with a Web application?*** A task is a course of actions that a user might want to accomplish on the Web application. A task is identified on the basis of three major characteristics: 1) be reusable; 2) perform a goal; and 3) be state independent. The resource interaction may be performed by a user when clicking a link or filling a Web Form or by the Web browser without the knowledge of the user (e.g., a Web redirection). We have identified three types of resources used in Web applications: Type I resources[1] have fixed URLs; Type II resources[2] take URL parameters or payload as input; and Type III resources[3] take input from a user and then a client side code executes something locally. A user event, such as a button click, calls the JS function. The HTTP protocol is invoked from the JS function. A resource interaction may execute a client-side script before issuing a request to a resource. Basically each resource interaction performs a HTTP-method on a URL with some parameters. We model a task as a series of resource interaction with a Web server.

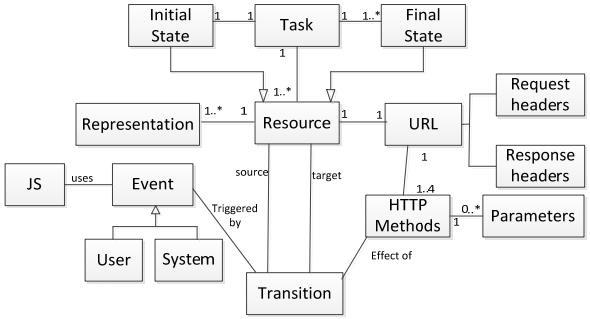


Figure 1: Meta-model for users' tasks

Figure 1 show the meta-model to model users' tasks. A task can be accomplished by one or more resources. A task starts with an initial resource (*i.e.,* initial state) and ends with one or more final resource. Each resource has a URL and one or more representation. Each URL has request and response headers. The request and response headers are components of the message header in HTTP. They define the operating parameters of an HTTP transaction. While completing a task, a resource undergoes a series of transitions. A transition occurs by user action or by system events

III. OUR APPROACH

Figure 2 shows the overall process of our approach to represent a task as a RESTful service. Our approach consists of two steps as shown in Figure 2. The first step is to select and execute a task to migrate. A user chooses a task to migrate as a RESTful service in a Web application. A user does not necessarily need to be an expert in the technologies used to develop the Web application. We provide a tool to denote the start and completion of a task. A user runs a Web application multiple times denoting the start and the completion of tasks to capture all scenarios involved in completion of a task.

Figure 3(a) shows a menu to denote the start and completion of a task in our Firefox plugin. A user denotes the start and the completion by clicking the menu. Figure 3(b) shows a task completion process for a login task. In a login task, a user clicks the login link and fills a login form. Based on the data entered, this task can have one of the two final states. We instrumented the browser to record all events generated by a Web application in a client-side. The second step is the analysis of the annotation logs and the execution logs to identify input, output and HTTP methods of a task.

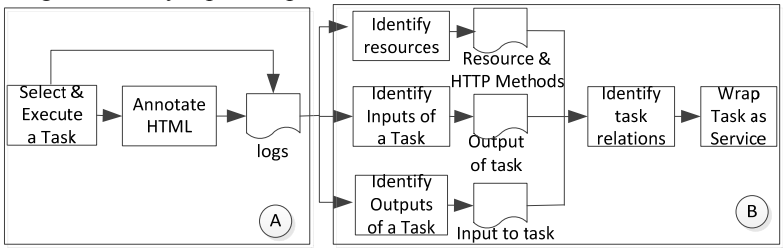


Figure 2: Overview of our approach to Identify service from Web pages

A. *Identifying Inputs of a Task*

The meta-model in Figure 1 shows that a resource transition can be either a user event or a system event. Web forms and hyperlinks are used to provide input to a Web application. A Web form submission doesn't always invoke a resource. Web forms generally generate a number of events. These events are handled by client side JS functions. JS programs are executed by a client's Web browser and have access, via a document object model, to the resources of the browser, in particular, to the HTML document shown in the browser.

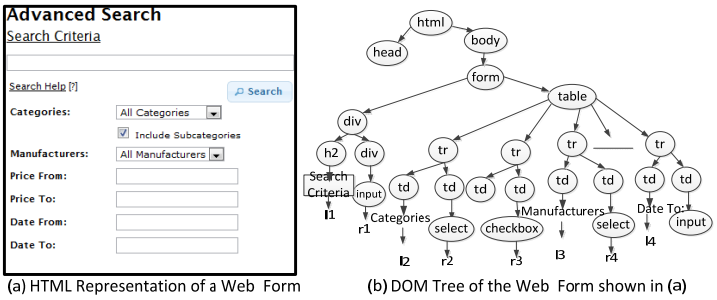
Our plugin keeps track of all the events generated during the completion of a task. Web forms and hyperlinks contain semantic information (*i.e.*, labels). The positions of labels in a Web form depend on the designer of the Web page. Labels can be placed above, below, to the left, or to the right of an input element. To identify the label representing an input element, we analyze the content of a Web page delimited by the opening and closing tags of a HTML partitioning element

(a) Menu to denote Start and Completion of a task

(b) Task involving multiple Web page navigation

Figure 3: Different phase of task Identificatication process

1. Weather Forecast http://www.theweathernetwork.com/weather/caon0349
2. EBay http://www.ebay.ca/sch/i.html?_nkw=IPhone
3. Google Accounts https://accounts.google.com/ServiceLogin



(a) HTML Representation of a Web Form (b) DOM Tree of the Web Form shown in (a)

Figure 4: HTML and DOM representation of Web query interface

that separates different sections of a Web page. For example, paragraph tag (*i.e.,* <p>) separates a paragraph in HTML. The text nodes under the partitioning element are part of the same blob (i.e., a text contained within a partitioning element). However, style tags, such as the italic tag (*i.e.,* <i>) and the bold tag (*i.e.,* <b>) are generally used to add styles within a section of text. Therefore, styling tags are not considered as partitioning elements.

Web form labels and input elements are hierarchically nested in a DOM tree. Hierarchical proximity between the elements helps to associate the input elements with the text blob. Figure 4(a) shows a screenshot of a Web query interface. Figure 4(b) shows a fragment of the DOM tree of the query Web form shown in Figure 4(a). In Figure 4(b), the input field *r1* is in closer hierarchical proximity with the label l1 (*i.e.,* "Search Criteria") than the label l2 (*i.e.,* "Categories").Therefore, the label l1 should be associated with the input r1. To identify the association between input elements and labels, we traverse and analyze the DOM tree to find the text nodes that constitute a label. When a partitioning element (such as *<p>,
,* and *< hr>)* is reached, we create a new label. The text node under the partitioning element is added to the label. If the partitioning element contains another partitioning element as a child, then the text nodes that appear under the sub-partitioning child belong to the text blob of the sub-partitioning child. For each input element, we compare the hierarchical proximity between the input element and the text blob. The label with the least distance is considered a candidate of an input description tag for the input element. The distance between a text blob and an input element is given by the number of nodes visited from the text blob to reach the input element. For example in Figure 4(b), the distance between the nodes r1 and l2 is 2; the distance between the nodes, r1 and l3, is 4; and the distance between the nodes, r1 and l4, is 5. The node r1 has the least distance with the text blob l2; and hence the node l2 is selected as the description tag for the node r1. If more than one candidate is identified, we calculate the edit distance [9] between the candidates and the "name" attribute of the input element to choose the candidate for the input description tag. We track changes in cookies and HTTP header fields and consider them as input parameters.

B. *Identifying Output of a task*

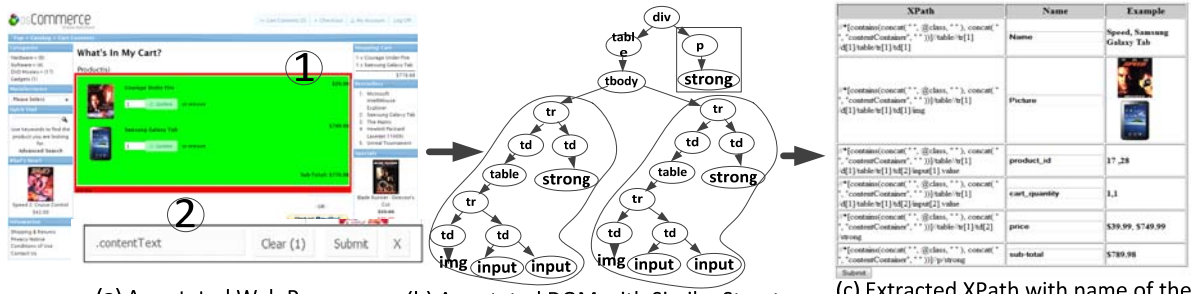
The result of a Web form submission is generated in a template. The template generated content contains advertisements, navigational panels and so on. Although these parts of a Web page may be helpful for user browsing, they can be considered as "noisy data" that may complicate the process of extracting data objects from Web pages. When dealing with Web pages containing data objects and "noisy data", the "noisy data" could be wrongly matched as correct data resulting in either inefficient or even incorrect wrappers. Consequently, given a Web page, the first task is to identify which part of the page is the data rich section, *i.e.,* the section or frame that contains the data objects of interest to the user. The annotation tool helps to select data record that is used as output of a task. Whenever a user submits a form, there are basically two kinds of data send by the Web server header information containing a status code and a resource representation. We keep track of all the changes in the header fields. To identify the output of a task, a user can select the region in an HTML representation that represents the output using our annotation tool. The following steps are performed on the selected region of representation.

1) Select a portion of an HTML representation. A user selects a segment of HTML representation. Figure 5(a) shows the example that a user selects specific part of a HTML page.
2) Parse the HTML of the source document and find the starting (SP) and ending (EP) positions of the selected region.
3) Identify regions with similar DOM structures between SP and EP. Our approach identifies segments of DOM regions with similar DOM structures. Similar DOM structures indicate similar types of data. Figure 5(b) shows an example of similar DOM structures. We use the following heuristic to identify the semantics of the extracted elements. To apply the heuristic, we proceed in two steps: First, we match Web form labels with responses. We examine if any labels discovered from a Web form are presented in the response page. Second, we search for labels in table headers. HTML specifications define tags, such as header cells and header contents in HTML tables. We list the columns of HTML tables; and search for voluntary labels encoded in the response pages. For example, if a page contains a column with the symbol '$', we consider the data item represents currency related fields such as price.

Our approach identifies and refines the semantics of the extracted data template. A user can import an available ontology or define her own ontology if there is no available ontology. Figure 5(c) shows a screenshot of the GUI of our tool to help users to refine the extracted data templates. Based on selections, we identify different parts that can be named by a user. XPath and ontology mapping are described in a single file for each task. These description files can be modified and reused easily, without affecting other parts of the generated services.

C. *Identifying Resources and HTTP Methods for a Task*

In this step we identify resources required to accomplish a task and the execution sequence between the resources. A task uses one or more resource with different HTTP-methods. We select unsafe methods over safe methods and



(a) Annotated Web Page (b) Annotated DOM with Similar Structure (c) Extracted XPath with name of the elements

Figure 5: Identifying the data segment in the HTML Representation

un-idempotent methods over idempotent methods. For example, if a task uses two resources one using HTTP method GET and the other using POST, the HTTP method for that task is POST. If an intermediate resource changes the parameters of cookies during the completion of a task, the most recent change in the cookie is propagated to the client.

D. *Identifying Task Relations*

Web developers embed links in a HTML representation that guides a user from one state to another. We analyze next-state elements (*i.e.,* links and Web forms) to determine the transition sequence. We propose the following four rules to extract task relations from a client-side representation.

Identify state changes without requests and responses: A client state may change without requests and responses of URLs. In this case, the URL, HTTP-methods and parameters remain the same, whereas there is a change in the representation. This kind of change is due to the client slide scripts, such as client validations of Web forms. For example shown in Figure 3(b), when a user submits a form without username and password, the representation displays a validation error. This rule relates the client side script to a Web user interface control.

Identify related tasks: This rule helps to identify dependent resources. The resources may have one to many relationships with other resources. We cluster URLs with similar parameters and resource paths. For example, if the URL of a product resource is http://../product?pid=xx and the URL of the review resource is http://../reveiw?pid=xx, the parameter names in the URLs of the product info task and the product review task are similar and belong to the same cluster. Hence the two tasks are related.

Identify the next task to perform: A Web developer embeds a link or a Web form that helps a user to decide what to do next. This rule identifies embedded next-state elements and the tasks associated with the next-state elements. We extract the next-state elements from all the resources used to accomplish a task and choose non-reoccurring elements. A non-reoccurring element is a symmetric difference between the next-state elements of two resources. For each resource, we identify non-reoccurring elements. We identify tasks whose initial states are present in the non-reoccurring elements list. For example shown in Figure 3(c), after a user logs in, the logoff link appears in response representation.

Identify dependent task: Dependent tasks require an authorization from another task. These relations are identified by finding subset relations among tasks. For example the checkout of a shopping cart resource needs the login task to be invoked first.

IV. CONCLUSION

Our work addresses the problem of migrating reusable tasks of Web applications towards service oriented architecture. Our approach considers a Web application as a special type of form-based system containing one or more Web pages. The processes of RESTful service extraction run at client-side and do not depend on server side code. Our approach extracts reusable tasks by analyzing client side Web user interface controls and fragments of HTML representation developed with a combination of JS, HTML and CSS code. However, our approach currently does not support the extraction of Silverlight nor Flash. Our initial experiment shows that our approach can extract task and task relations as RESTful services. In future, we plan to perform a detailed case study on different real-life Web sites to extract services.

REFERENCES

[1] A. Almonaies, J.R. Cordy and T.R. Dean. Legacy System Evolution towards Service-Oriented Architecture. In International Workshop on SOA Migration and Evolution, pages 53–62, 2010.

[2] R.T. Fielding. "Architectural Styles and The Design of Network-based Software Architectures". PhD thesis, University of California, Irvine (2000)

[3] G. Lewis, E. Morris, L OBrien, D. Smith, and L. Wrage. Smart: The service oriented migration and reuse technique. In Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pages 222–229,2005.

[4] Oney S., Myers. B., FireCrystal: Understanding interactive behaviors in dynamic Web pages , IEEE Symposium on Visual Languages and Human-Centric Computing, 2009.

[5] R. Holmes, T. Ratchford, M. Robillard, and R. J. Walker. Automatically Recommending Triage Decisions for Pragmatic Reuse Tasks. In Proc. Of 24th IEEE International Conference on Automated Software Engineering, 2009.

[6] Li P., Wohlstadter E., Script InSight: Using Models to Explore JavaScript Code from the Browser View, Proceedings of the 9th International Conference on Web Engineering Pages 260 - 274

[7] Almonaies A., Alalfi M., Cordy J.R. and Dean T.R. , Towards a Framework for Migrating Web Applications to Web Services, Proc. CASCON'11, Toronto, November 2011, pp. 229-241.

[8] Ajlan A. and Zedan H,. E-learning (MOODLE) Based on Service Oriented Architecture. In the EADTU's 20th Anniversary Conference, Lisbon, Portugal, 8-9 November pages 62–700, 2007.

[9] Baeza-Yates, R. Algorithms for string matching: A survey. ACM SIGIR Forum, 23(3-4):34-58, 1989.

All URLs are last accessed on 20th October, 2012