KIRKE
Re-engineering of Web Applications to Mobile Apps

INDRAPRASTHA INSTITUTE *of* INFORMATION TECHNOLOGY **DELHI**

Rohit Mehra

Computer Science

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

A Thesis Report submitted in partial fulfilment for the degree of

MTech, Computer Science

24 November 2015

Thesis Committee

1. Vinayak Naik (Thesis Adviser)

2. Rahul Purandare (Internal Examiner)

2. Vasudeva Varma (External Examiner)

Day of the defense: 24 November 2015

Signature from Post-Graduate Committee (PGC) Chair:

Abstract

A large number of web applications are written using server-side scripting languages. Although web browsers allow clients to run these applications, it is often cumbersome to depend on the desktops for the services provided by the applications. Given the popularity and convenience of mobile devices, there is an obvious need to have native mobile apps driving the applications along with web browsers.

In this thesis, we present a solution to re-engineer web applications, developed using server-side scripting languages, into native mobile apps. The solution takes source code and test cases of the web application as input and produces corresponding cross platform mobile apps. The entire re-engineering process is fully automatic requiring no manual intervention at any stage. Our solution is generic enough not only to handle popular server-side scripting languages, but also to output mobile apps that support diverse popular platforms including Android, iOS, and Windows Mobile. To showcase the generality of our solution, we have developed a prototype tool Kirke to handle applications developed using Java(JSP), ASP.NET, and PHP. We present a case study based on a real-life codebase in JSP to evaluate the correctness, coverage, usability, and performance of our solution.

Acknowledgements

"When you talk, you are only repeating what you already know. But if you listen, you may learn something new."

- Dalai Lama

This is the motto of my advisor Dr. Vinayak Naik whose help, stimulating suggestions and encouragement helped me at all the times of research and writing of this thesis. His unparalled guidance and motivation has always been the source of inspiration whenever I got stuck in the work. Without his guidance this work would not have been possible. I would like to take this opportunity to express gratitude to him for inspiring me to undertake research as a career. The amount of time that he has taken out from his busy schedule is quite commendable and worth mentioning. I thank him from the bottom of my heart.

My heartfelt thanks to Dr. Rahul Purandare for his valuable time, suggestions and inputs while writing of this thesis. It is his exceptional teaching style and knowledge that has prevailed my interest in this domain.

Next, I would like to thank Mr. Vikrant Kaulgud and Mr. Saurabh Bhadkaria for laying out the foundation stone for this work. They have always been there to support me and to share their expertise in the software engineering domain.

I would also like to thank Dr. Vasudeva Varma for his valuable time and suggestions.

A special thanks to all my friends for critically reviewing and commenting on the work and helping to make it what it is. I would like to thank my parents for always supporting me and providing me an environment where I could work dedicatedly. Last but not the least, I would like to thank IIIT Delhi for providing a research oriented atmosphere and facilities required for driving the work, uninterrupted.

Certificate

This is to certify that the MTech Thesis Report titled **KIRKE: Re-engineering of Web Applications to Mobile Apps** submitted by **Rohit Mehra** for the partial fulfillment of the requirements for the degree of *MTech* in *Computer Science* is a record of the bonafide work carried out by him under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Vinayak Naik
Indraprastha Institute of Information Technology, New Delhi

Contents

List of Figures

List of Tables

List of Code Listings

1

Introduction

1.1 Research Motivation

According to Cisco "By 2019, there will be 8.2 Billion handheld and personal mobile-ready devices", which approximates to one device per person on this Earth [1]. This is why, 77% of companies now rank mobility as their top business functionality and priority [2]. Mobile phone is now the primary source of consuming online media with 60% of the average global mobile phone users using it every day to surf the web, leading to more than 60% of the total online traffic [3, 4]. For Facebook, 69% of their total advertisement revenue comes from mobile apps rather than their desktop counterparts [5].

The impact of mobile phones is more prominent in developing countries like India, as the first and primary access to the Internet for most users in these countries is from mobile phones. India ranks second globally by the number of Internet users, with 60% accessing the Internet from their mobile phones only. These numbers are expected to grow at a rate of 50% by 2017 [6]. This unprecedented growth in mobile Internet user base has shifted the focus of the Indian Internet companies from web towards mobile. Myntra being one of the largest Indian online fashion stores has gone mobile only, backed by Flipkart, India's largest online retail store by sales which is also expected to go mobile only by the end of 2015 [7]. According to Flipkart, 70-75% of their total traffic is already coming from their mobile app.

Sudden surge of portability and ubiquity has forced the developers to lean towards a mobile-first approach and provide mechanisms to allow access to their existing web application via mobile devices. Two possible strategies to move towards a mobile-first landscape are:

- Develop server and mobile application from scratch. This involves high development cost and duration.
- Re-engineer the existing web applications to create mobile apps and compatible server side code. This would save development cost and duration. However, a challenge is to design a re-engineering solution that can transform the existing code with correctness and complete coverage, without compromising much with its usability. Further, it would be useful to have a solution that is generic enough to handle multiple server-side scripting languages, and target a wide range of mobile devices.

1.2 Research Contribution

Extensive research in the domain of re-engineering has contributed towards various strategies that involve static analysis of the legacy application's source code and Unified Modeling Language (UML) diagrams. Though manual, sections of these strategies have been automated using existing code analyzers to expedite the entire process. One of the drawback of these strategies is the requirement of complex code parsers and analyzers, enabling relevant information extraction from large codebases. To increase efficiency and accuracy, they require extensive documentation and diagram support, in formats understandable by the supporting tools. High failure rates can be observed, if prerequisites are either unavailable or not following the appropriate format convention.

Dynamic analysis techniques overcomes these drawbacks. They allow us to argue about the properties of an application by inserting probes in it and then executing it. Our approach leverages these techniques to extract information necessary to set up the client-side support for the target mobile app.

Advantages of this approach include, easy migration to other languages and frameworks using lightweight codepoint weavers, no requirement of documentation or diagrams, multiple platforms support and full automation without even having prior knowledge of the application. To the best of our knowledge, our solution is the first attempt to fully automate the process of transforming web applications to mobile apps.

We have developed a prototype of our approach called KIRKE, to support transformation of web applications that are based on popular server-side languages. As a proof of concept, we transformed a Java Shopping Cart web application [8] using KIRKE and conducted a study to understand the correctness, coverage, usability, and performance of our implementation. The salient features of our prototype tool KIRKE are as follows.

1. The solution takes in source code and test cases only and it does not require any manual interventions.
2. The processing time is a function of number of classes containing HTTP methods, hence it is scalable.
3. The approach is generic enough to work for all popular server-side scripting languages, such as JSP and ASP.NET.
4. It generates native mobile apps, with support to integrate model specific sensors like accelerometer, GPS and temperature for all popular mobile platforms.
5. The code generated by our tool is human readable and documented, thereby making it easy to maintain and extend.

The rest of the thesis is structured as follows. In chapter 2, we discuss high level concept of the proposed approach along with the problem statement. Subsequently, in chapter 3 we present the proposed approach to transform a web application to a mobile app. Chapter 4 focuses upon the generalizability aspect of our approach. In chapter 5 we evaluate our approach using a case study. In chapter 6, we discuss background information and related research. Finally, in chapter 7 we conclude our research and propose future research directions.

2

Overview

2.1 Overview

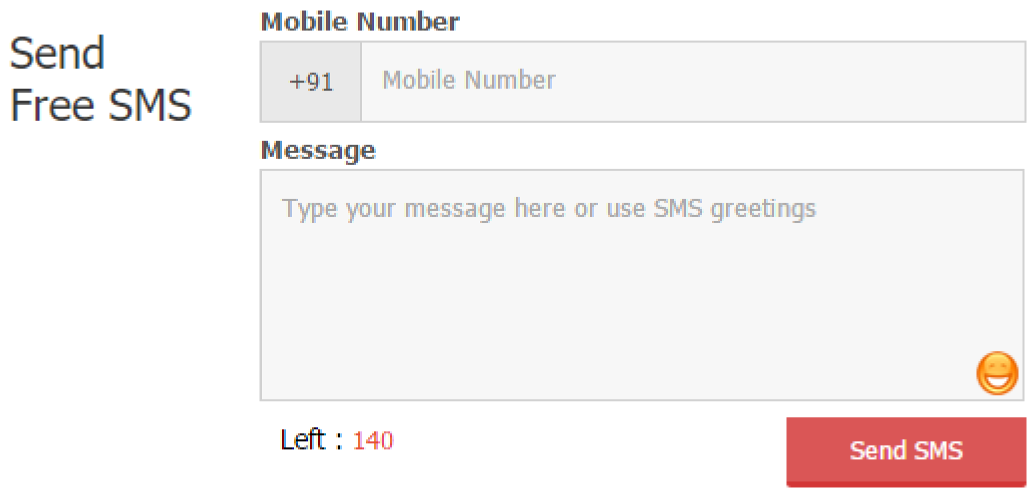
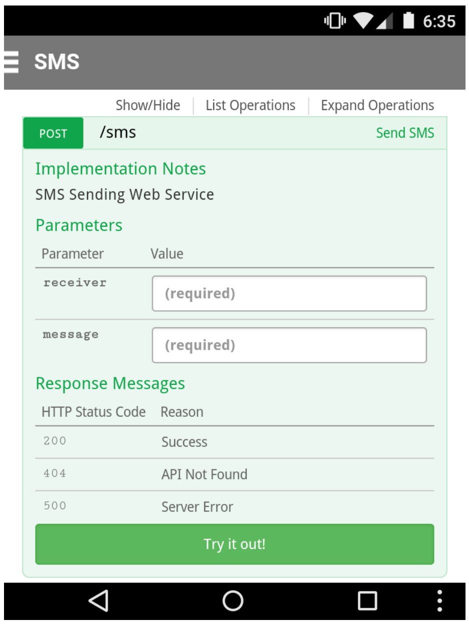


Figure 2.1: Send SMS page screenshot from way2sms.com

In this section, we give an overview of our approach using a simple example from a real-life application developed to build website `www.way2sms.com`. Figure 2.1 depicts a webpage which is used to send a sms to a desired mobile number. Our approach transforms the website's source code to allow sending sms from a native mobile application.

In order to achieve the required transformation, we perform a sequence of steps starting with a dynamic analysis of a web application in which the application code is automatically injected with synthetic method calls at fixed and pre-identified instrumentation points. We then rebuild the application along with the injected code and execute it with existing JUnit or Selenium test cases to enable maximum code coverage. Execution of injected code helps in the extraction of runtime parameters used by the application. This information is used to generate wrappers around the legacy code using templates and code generators which provide the API. The wrappers allow access to the internal legacy code via HTTP. Deployed web services are automatically annotated and described using a documentation framework [9], that provides a live documentation and testing environment for the API. This documentation is then used to generate client mobile applications that can run on multiple mobile platforms including Android, iOS, and Windows Mobile.

We have developed a tool Kirke that performs this transformation automatically. Figure 2.2 shows the output cross platform mobile application generated by Kirke for the "Send SMS" page.



Implementation Notes

SMS Sending Web Service

Parameters

Parameter	Value
receiver	(required)
message	(required)

Response Messages

HTTP Status Code	Reason
200	Success
404	API Not Found
500	Server Error

Figure 2.2: Generated mobile app to communicate with transformed synthesized Send SMS service from way2sms.com.

2.2 Problem Statement

The problem under consideration is to transform a web application, written using any server-side scripting language, to a mobile app for all popular platforms with minimal time and effort. In order to reduce time and effort, the transformation should use the existing code with minimal changes and require minimal manual interventions. The resulting mobile apps should provide correctness and coverage with respect to the legacy web application. Underlying are the major challenges, which we need to address, while designing a solution to the problem.

- Gathering extensive prior knowledge about web application's architecture, data flows, and UML diagrams will need extensive time and manual intervention. Therefore, our solution should not assume availability of any of these artifacts.
- Static analysis techniques require sophisticated, complex language parsers, and analyzers that are hard to construct and are not openly available for all modern languages. Moreover these techniques are often less scalable and less precise. Therefore any solution using static analysis techniques are specific to languages and will not support multiple scripting languages.
- Different mobile platforms require an application to be built and modified using different languages, which requires platform specific developers and designers. Therefore, our solution should generate platform independent mobile applications which are easy to modify using existing web application developers.

3

Research Framework and Solution Approach

3.1 Proposed Approach

Table 3.1 depicts all possible cases of web applications depending upon availability and usage of MVC design pattern and test cases. Proposed core approach assumes presence of both the parameters. Unavailability of test cases is compensated by manual generation of test cases using subject matter experts. In case MVC is not the underlying design pattern for web application, it is preprocessed to simulate MVC like structure. Core solution is discussed in details in the following paragraphs. Rest of the cases will be discussed later in the section.

Figure 3.1 depicts the schematic view of our approach. Our approach requires source code and test cases of web application as input and generate corresponding mobile apps. All of the steps are fully automated. Proposed approach also assumes MVC design pattern as the underlying architecture used by the web application.

Cases	MVC	Test Cases	Solution
Case 1	Yes	Yes	Core Approach
Case 2	No	Yes	Service Location Mapping + Case 1
Case 3	Yes	No	Generate Test Cases Using SMEs + Case 1
Case 4	No	No	Generate Test Cases Using SMEs + Case 2

Table 3.1: List of scenarios for developing a web application based upon implementation of MVC design pattern and availability of test cases.

For the context of this paper, we define a service as a functionality provided by the web application. A service is triggered directly through a human interaction i.e submitting a form, clicking a button or invoking a URL among others. For example adding a comment and liking a picture are some of the services offered by Facebook. A web application consists of multiple pages, each mapped with one or more services. It is a N-to-1 mapping scheme, where N services are getting mapped to 1 page. Our target is to make all the services provided by the web application accessible via a mobile app.

We will take "Send SMS" service example shown in Figure 2.1 as a running example to illustrate our approach. This service is used by millions of users for sending free SMS to any mobile number in India with a limit of 140 characters per message. Since way2sms is a closed source web application, we synthesized its background code as shown in Listing 3.1. The code accepts a receiver's mobile number and message as parameters bound inside a HttpServletRequest object. These parameters are extracted from the request object and passed to sendMessage() method. The code prints the appropriate status to screen, based upon success or failure of the message transmission.

In this section, we will describe the steps involved in the transformation of the server-based code into a mobile app. We use JSP to illustrate code snippets. The example is generic enough to completely illustrate the transformation process, yet simple enough to not get entangled in the complexity of the application logic and the syntax of JSP. Since a JSP page internally gets converted to a Servlet by the web container, we will demonstrate all of the steps using the servlet code.

In order to perform this transformation the approach leverages Model View Controller (MVC) [10] to efficiently transform web applications to mobile apps. MVC is a widely employed design pattern for developing web applications using server-side scripting languages. The MVC pattern segregates presentation, business logic, and data. This modularization facilitates the process of candidate service identification and instrumentation.

All server-side scripting languages support this pattern either out of the box (ASP.NET MVC and JSP-Servlet) or by using custom-built frameworks (PHP, Ruby, and Python).

3.1.1 Code Instrumentation

This step instruments source code of the web application with additional code, which extracts class names, interfaces, formal arguments, data types of the arguments, actual values passed

```java
public class SMS extends HttpServlet {
 @Override
 protected void doPost(HttpServletRequest request,
    HttpServletResponse response){
  Double receiver = request.getParameter("receiver");
  String message = request.getParameter("message");

  boolean sms_status=SMS.sendMessage(receiver,message);

  PrintWriter out = response.getWriter();
  out.println("<h1>" + sms_status + "</h1>");

}
```

Listing 3.1: Synthesized JSP code for Send SMS service offered by way2sms.com.

to the arguments, and invoked methods at run-time. The instrumented code executes inline with the legacy code and extracts run-time information.

In order to identify instrumentation points in the legacy web code, we exploit the separation of concerns property of the MVC design pattern adopted by JSP. Figure 3.2 shows how JSP exploits MVC for developing web applications. Following are the components involved in realizing the MVC used by JSP.

- **Model** JavaBeans and Plain Old Java Objects (POJOs) represent the Model. They are responsible for providing methods for accessing and updating data.
- **View** JSPs represent the View. They are responsible for accepting data from the JavaBeans and presenting it using graphical user interfaces in HTML.
- **Controller** The servlet represents the Controller. Every user interaction is converted to a HTTP request and handled by the servlet. It is the servlet's responsibility to delegate the request to appropriate Model and forward the corresponding response to be rendered by the JSP.

Each of the user's interactions, goes via HTTP methods in Controllers, hence controller methods act as a starting point for all the services, which makes them a candidate instrumentation point. There are seven such HTTP methods available in JSP controllers. They

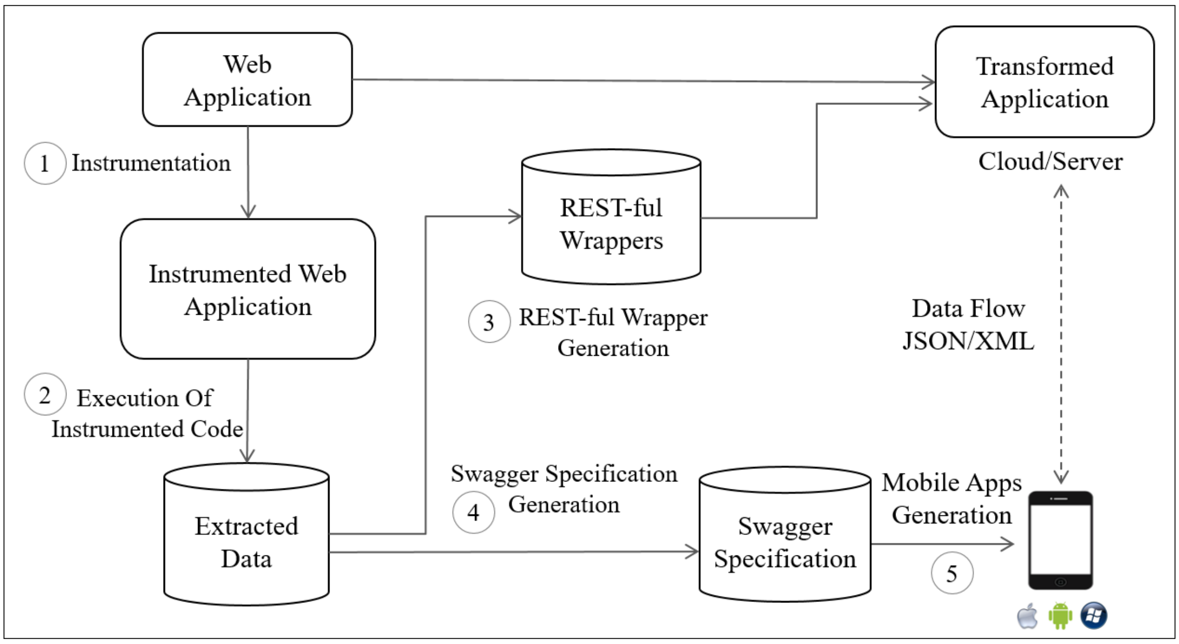


Figure 3.1: Proposed solution architecture depicting the steps required to transform a web application to mobile app.

are doGet, doPost, doDelete, doPut, doTrace, doHead, and doOptions. We instrument each of these methods to extract run time information when these methods are invoked.

In case of "Send SMS" service, there is a doPost() method, to which we add a DataExtractor.extract() method, as shown in Listing 3.2.

3.1.2 Execution of Instrumented Code

After successful instrumentation of the code, it is compiled and executed. The objective, while execution, is to ensure that all the common cases are executed. Selenium is a suite of tools to automate web browsers across many platforms. If Selenium and JUnit test cases are available, they can be leveraged to ensure coverage. In the absence of these test cases, the instrumented code statements are tagged with unique identifiers. Each instrumented code is assigned a unique instrumentation identifier. Every time the extract() method is called, it extracts the run-time information and stores it. Instrumentation identifiers help in identifying services which were instrumented but not invoked during run-time. The code is then executed again for the initially uninvoked methods and this eventually helps in maximizing the coverage. The data retrieved by the instrumented code, is saved for the next step. For the case of

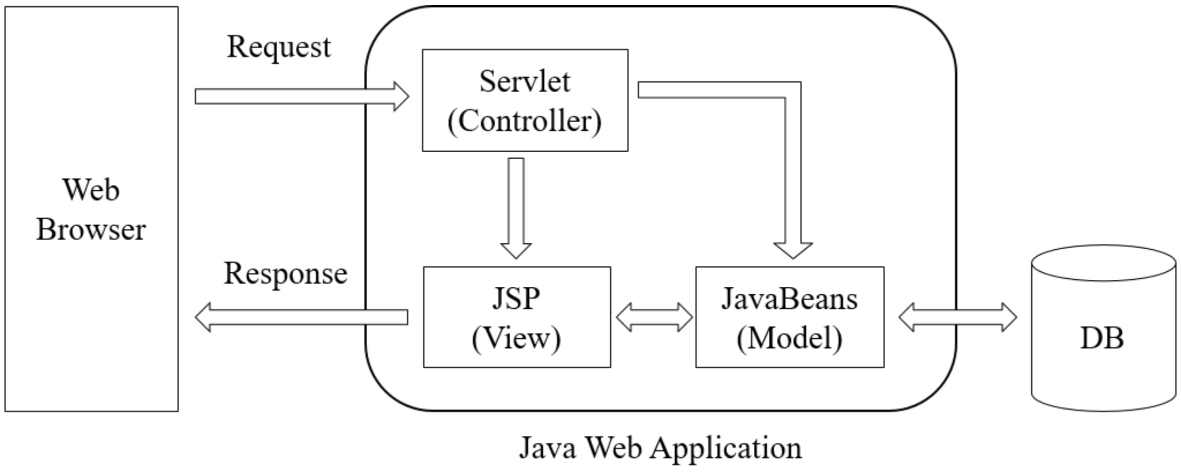


Figure 3.2: JSP's implementation of the MVC design pattern

"Send SMS", the instrumented code extracts the following information:

- Class name: SMS
- HTTP method: doPost()
- Parameters: {receiver, message}
- Values passed for the parameters at runtime: {989968****,"Rs.500 withdrawn with respect to transaction id: xyz"}
- Data type of parameters: {double, string}

The stored information is used to create REST-ful wrappers for the controller methods and also to create the corresponding mobile apps that invoke these newly generated APIs.

3.1.3 REST-ful Wrapper Generation

Each of the HTTP controller method is then wrapped within REST-ful functions that contain code segments to

- Extract URL parameters from the incoming HTTP request
- Route an incoming request to appropriate legacy code
- Invoke legacy code and marshal parameters to Java objects to be passed to legacy code
- Retrieve and unmarshal response generated by legacy code

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response){
  /* Instrumented Code */
  DataExtractor.extract(request,response,other_parameters);

  /* Legacy Code: Unchanged */
  Double receiver = request.getParameter("receiver");
  String message = request.getParameter("message");
         ...
}
```

Listing 3.2: Instrumenting synthesized JSP code for Send SMS service offered by way2sms.com with DataExtractor.extract() method, in order to extract run time information

- Implement session management and security

Traditionally, web application code developed using server-side scripting languages, is invoked by passing java objects and even returns java objects. Example: JSP controller methods are invoked by passing parameters of type HttpServletRequest and generate java objects as response. They do not use JSON or XML for passing data, with which mobile apps are created. The wrapper code containing REST-ful functions, translates back and forth between java objects and JSON or XML in order to communicate with legacy code and trap return objects. The data collected in the previous step has all the necessary information, such as the names of classes and parameters to generate REST-ful wrapper functions.

We have prepared a template for JSP, so that collected data is substituted in the template to generate the wrapper functions. Listing 3.3 depicts a sample template for the wrapper based on Java API for REST-ful web services (JAX-RS). Similar templates can be generated for other server-side scripting languages.

Listing 3.4 shows substitution of data extracted for “Send SMS” service in the template. The result is a REST-ful function that can be called via HTTP. This REST-ful function acts a wrapper code to the original JSP controller method. Whenever the wrapper function is called, it marshals the passed values of the parameters to the original code, traps the response object (by refactoring legacy application’s return statements), unmarshals response object, and forwards it to the client.

```
/* ${*} signifies a placeholder */

public class ${class_name} {
 @POST
 @Path("${service_path}")
 @ApiOperation(value = "${service_description}")
 @ApiResponses(value = {${service_response}})

 public Response doPost(${service_parameters}){
  ${wrapper_code}
}
```

Listing 3.3: JAX-RS template developed using Freemarker templates, required to develop Java REST-ful APIs

3.1.4 Swagger Specification Generation

The web application, with REST-ful wrapper functions, is deployed on a server. The wrapper functions help to access legacy services over HTTP by referring to specific URI's, e.g. `https://shoppingstore.com/api/sms.json?receiver=989968&message=TransactionCompleted`, refers to a synthesized REST-ful API, which sends a SMS message to a mobile number specified as a receiver. The API requires receiver's mobile number and the message as query parameters. It returns a JSON object as a response. Mobile apps will utilize the REST-ful API URLs in order to talk to legacy services. To automatically generate the mobile apps, we need a mechanism to translate the REST-ful wrappers to mobile application code.

We use a tool called Swagger-Codegen [11] that automatically converts REST-ful wrapper to corresponding Swagger specification. Swagger specification is a formal structure widely used to document REST-ful APIs. The formalized JSON documentation contains all the information required to interface with the API such as access URL, parameter description and types, and response codes. Listing 3.5 shows a sample swagger specification template.

REST-ful wrappers contains annotations required to document a specific API. Data specified in these annotations is provisioned semi-automatically and is used to generate better documentation. Before proceeding with mobile apps generation, all wrappers are converted to corresponding swagger specifications. Swagger-Codegen extracts specification from live code, using these annotations and wrapper code.

```
public class SMS_Service {
 @POST
 @Path("/sms.json")
 @ApiOperation(value = "SMS Sending Web Service")
 @ApiResponses(value = {@ApiResponse(code = 404, message = "
    API Not Found"),@ApiResponse(code = 500, message = "
    Server Error"),@ApiResponse(code = 200, message = "
    Success")})

 public Response doPost(@ApiParam(value = "Receiver Mobile
    Number", required = true) @QueryParam("receiver") Double
     receiver, @ApiParam(value = "Message To Be Sent",
    required = true) @QueryParam("message") String message )
    {

/* Wrapper code to invoke doPost() method, under SMS Class
   with receiver and message as parameters */

    }
}
```

Listing 3.4: REST-ful wrapper based on JAX-RS specification required to communicate with synthesized JSP code for Send SMS service offered by way2sms.com

Listing 3.6 shows the extracted swagger specification for the Send SMS module. Here, @ApiOperation specifies the functionality that the API is providing; @ApiResponses provides information about HTTP response codes that the API will generate while execution; and @ApiParam describes parameters required to invoke the API.

3.1.5 Mobile Apps Generation

In this step, we use the Swagger specification of the REST-ful APIs to generate native mobile apps.

We use Apache Cordova [12] to generate mobile apps from the Swagger specifications. Apache Cordova allows a native mobile app to be built using HTML, CSS, Javascript, and AJAX. It even allows an access to mobile's native APIs, such as Camera, Accelerometer and

```
/* ${*} signifies a placeholder */

"${path}" : {
   "${http-method}" : {
       "tags" :   ,
       "summary" :      ,
       "description" :  ,
       "operationId" :  ,
       "parameters" : [ {
          "name" :          ,
          "in" :            ,
          "description" :  ,
          "required" :     ,
          "type" :
          }],
       "responses" : {
         "${response-code}" : {
              "description" :   }
         }
       }
}
```

Listing 3.5: Swagger specification template used to document a REST-ful API

File Manager, to enrich the functionality of the app.

Swagger's live testing framework called Swagger-UI [11] parses swagger specification and generates form based user interface and necessary background code to communicate with the service depicted by the specification. We leverage this generated user interface and make it mobile friendly using Twitter Bootstrap, that provides css classes to make responsive interface elements. We pass this interface and code through Apache Cordova which converts it to a cross platform mobile application. Figure 2.2 shows the automatically generated mobile application for Send SMS module.

Kirke has been developed as a proof of concept for our proposed solution. KIRKE is designed to transform JSP and ASP.NET web applications to mobile apps that can run on Android, iOS and Windows and provide same set of functionalities as the legacy web application. KIRKE has been developed keeping maintainability as a major concern. It consists of interfaces that allow following modifications in the final transformed application:

- Addition of a new service
- Modifying existing service documentation and URIs
- Merging two or more existing services to create a new service with higher granularity
- Merging existing service with a 3rd party web service

Updating web application logic code has no effect on generated wrappers and mobile apps, unless the mappings themselves are modified. In that case KIRKE needs to be executed again to generate new wrappers and mobile apps based on new mappings. For example, if “Send SMS” is modified to accept “confirm mobile number” as a parameter along with the previously defined parameters.

3.2 Other Cases

Proposed approach assumes that the web application uses MVC design pattern under the hood and existing test cases are available at the time of transformation. Though this assumption will mostly be true keeping in mind the popularity of MVC architecture, there may exist cases where the assumption does not hold true. Case 1 has already been discussed in the proposed approach and can be handled by KIRKE.

Cases 3 and 4 can be handled by KIRKE, but requires manual generation of test cases using Subject Matter Experts(SMEs), that possess prior expertise in dealing with the web application. Subject matter experts are required to execute the application manually for identification of common flows and services available in web application. This may compromise on coverage. Higher the number of subject matter experts, higher the coverage and confidence in test case generation.

Case 2 can also be handled by KIRKE, but requires application to be preprocessed before feeding it to KIRKE. Preprocessing involves simulating artificial MVC like structure by identifying service entry points in source code. We refer to this processing as Service Location Mapping.

As discussed earlier in the section, MVC design pattern helps in identifying service starting points in code which simplifies the code instrumentation process. In the absence of MVC, Kirke is unaware of the code instrumentation points, and is unable to proceed without the required information.

A naive strategy would be to instrument all the methods in the code and develop RESTful APIs for each one of them. This will include APIs originating from service starting point methods as well as internal methods. Segregating between both is hard.

Dynamic analysis techniques can again be leveraged to map services to starting point methods. First step in this mapping is to generate method execution traces for the entire application. This can be achieved by using tools like AspectJ or Soot for java. These tools are used to instrument all the methods with tracing codes, that generate a trace file upon execution. These tools can instrument the code in both byte code and source code representations.

After successful instrumentation of code, code is executed using existing or generated test cases. Each test case represents user driven interaction in the application. Post test case execution, test case is mapped with the first method call in the method trace, as this method acts as the starting point for corresponding service. Similarly all other services are mapped to corresponding starting methods by exhausting available test cases. Eventually these methods will be used by Kirke for transformation processing.

Section 5 describes our experience with Kirke in transforming a Java Shopping Cart application built using JSP.

```
"/sms" : {
 "post" : {
   "tags" : [ "SMS_SEND" ],
   "summary" : "Send SMS",
   "description" : "SMS Sending Web Service",
   "operationId" : "doPost",
   "parameters" : [ {
    "name" : "receiver",
    "in" : "query",
    "description" : "Receiver Mobile Number",
    "required" : true,
    "type" : "double"
    },{
    "name" : "message",
    "in" : "query",
    "description" : "Message To Be Sent",
    "required" : true,
    "type" : "string" }],
   "responses" : {
    "404" : {
     "description" : "API Not Found" },
          "200" : {
     "description" : "Success"},
    "500" : {
        "description" : "Server Error"
      }
     }
   }
}
```

Listing 3.6: Swagger specification generated by Swagger-Codegen representing REST-ful API wrapper for Send SMS service offered by way2sms.com

4

Discussion

In this chapter, we discuss generalizing our approach to server-side scripting languages other than JSP and also the usability aspects of generated mobile apps.

4.1 Generalizability

JSP, ASP.NET, and PHP together constitute 75% of web applications, written using server-side scripting languages [13]. Hence, the initial goal of our proposed framework was to make it amenable to these languages. Our choice of using dynamic analysis was largely driven by its advantages over static analysis including higher precision, simplicity, scalability, and smaller development cost. This choice makes the techniques more generalizable to other server-side scripting languages, such as ASP.NET and PHP. The only change required is the change of templates and instrumentation code, which is a one time effort for any language.

4.1.1 ASP.NET

ASP.NET is a popular web development framework by Microsoft and supports three different patterns out of the box, web pages, web forms, and MVC. Following are the components in realizing MVC design pattern for ASP.NET MVC.

- **Model** Business logic in Model is described using C# files and the source is kept inside Models directory.
- **View** It is described using web forms and HTML. This source lies inside Views directory. View accepts data from Controller as an object and renders a user interface to display it.

- **Controller** It is provided using C# and lies inside Controllers directory. It handles user requests, invokes appropriate logic in the model and passes response objects to views using ViewBag. There can be multiple Controllers. All Controller names end with "Controller" suffix, which makes it easy to locate those files.

Each method in Controller maps to a unique URI in the web application with a corresponding View. There is one-to-one mapping between method in Controller and URI. We use these methods to extract data in Kirke. As each request from user goes via these methods, it provides complete coverage to Kirke. Each of these methods is an instrumentation point to instrument code for the data extraction. Rest of the steps in re-engineering are the same, except for the template in listing 3.3. The template is in for C# code instead of Java. Listing 4.3 shows the wrapper template for ASP.NET. Listings (4.1, 4.2, 4.4) demonstrates Send SMS service transformation for ASP.NET.

Though proposed approach is applied in similar ways to both JSP and ASP.NET applications, we have noticed some key differences between both languages that we needed to handle. In JSP, a controller method can take only seven names as described earlier. On the other hand, ASP.NET controller methods can assume any custom name. A controller in ASP.NET can accept multiple parameters as opposed to single HttpServletRequest object in JSP. Instrumentation codes need to be customized depending upon controller method parameters. ASP.NET even accepts Model objects as parameters. Hence, instrumentation code was modified to extract information from model classes about data members passed inside a model. ASP.NET uses Web API in place of JAX-RS for wrapper generation. This is achieved by change in the template. Also REST-ful function code is modified to support C# instead of Java. The rest of the steps (swagger specification and mobile app generation) are similar for all other languages.

4.1.2 PHP

Unlike JSP and ASP.NET, use of MVC design pattern is not prevalent among PHP web applications. If a web application is not following MVC pattern, it means that we do not have have defined entry points into the web application code, when user visits the application using a browser. Kirke has to inject code in all the methods, thereby increasing the time to re-engineer the mobile app. The rest of the steps are similar. As in other languages described earlier, the completeness of the transformation process in case of PHP is dependent on the

```
namespace Way2sms.Controllers
{
    public class SMSController : Controller
    {
        // POST: /User/SendSMS
        [HttpPost]
        [AllowAnonymous]
        public async Task<ActionResult> SendSMS(SMS model)
        {
            boolean sms_status=SMS.sendMessage(model);

            ViewBag.status = sms_status;

            return View();
        }
    }
}
```

Listing 4.1: Synthesized ASP.NET code for Send SMS service offered by way2sms.com.

coverage provided by the system test cases. In future, we plan to work on using dynamic analysis techniques to find out commonly used entry points (candidate services) in the code. The plan will be instrument only these entry points. The resulting re-engineered mobile app will provide only those functionalities provided by the transformed app. So we tradeoff coverage with work involved in re-engineering.

4.2 Usability of the Mobile App

The re-engineered mobile app displays user interface generated using Swagger UI and is composed of interactive forms to communicate with the REST-ful URLs. Users of the app fill the forms as per the displayed documentation extracted from the web application. Upon submitting the forms, the mobile app sends a HTTP request to the REST-ful URL and renders the response in the user interface of the mobile app. The generated code for the mobile apps is such that the interface adapts itself to orientation changes of portrait and landscape modes. Look and feel of the interface remains identical across various platforms

```
public class SMSController : Controller
{
    // POST: /User/SendSMS
    [HttpPost]
    [AllowAnonymous]
    public async Task<ActionResult> SendSMS(SMS model)
    {
        /* Instrumented Code */
        Hashtable request = new Hashtable();
        hashtable[0] = model;
        DataExtractor.extract(request, this.GetType().Name,
           ViewContext.RouteData.Values["action"]);

        /* Legacy Code */
        ...
    }
}
```

Listing 4.2: Instrumenting synthesized ASP.NET code for Send SMS service offered by way2sms.com with DataExtractor.extract() method, in order to extract run time information

due to usage of Twitter Bootstrap. The usage also provides CSS classes to build additional user interface elements. We used Google Nexus 4 (Android 4.4), OnePlus Two (Android 5.1.1), Apple iPhone 5C (iOS 8), and Nokia Lumia 510 (Windows 8.1) for our study. In future versions of KIRKE, we intend to improve the UI such as by porting images from the web applications to mobile apps. This way the mobile app will have the same look as that web application.

```
/* ${*} signifies a placeholder */

namespace ${namespace}
{
    public class ${class_name} : ApiController
    {
        /// <summary>
        ///     ${service_description}
        /// </summary>
        /// <response code="${service_response_code}">${
           service_response_desc}</response>
        public async Task<HttpResponseMessage> Get(${
           service_parameters})
        {
            ${wrapper_code}
        }
    }
}
```

Listing 4.3: ASP.NET Web API template developed using Freemarker templates, required to develop C# REST-ful APIs

```
/* ${*} signifies a placeholder */

namespace Way2sms.API
{
    public class SMS_Service : ApiController
    {
        // POST api/<controller>
        /// <summary>
        /// SMS Sending Web Service
        /// </summary>
        /// <param name="receiver">Receiver Mobile Number</
           param>
        /// <param name="message">Message To Be Sent</param>
        /// <response code="200">Success</response>
        /// <response code="404"">API Not Found</response>
        /// <response code="500"">Server Error</response>

        public async Task<HttpResponseMessage> POST(Double
           receiver, String message)
        {
            /* Calling Code To Invoke SendSMS() method,
               under SMSController Class with parameters
               being Receiver and Message converted to SMS
               Model */
        }
    }
}
```

Listing 4.4: REST-ful wrapper based on ASP.NET Web API specification required to communicate with synthesized ASP.NET code for Send SMS service offered by way2sms.com

5

Evaluation

5.1 A Case Study

When we re-engineer a web application, coverage, correctness, ease of use, and efficiency in execution needs to be ensured. Coverage means no functionality from the original application is missed while transforming. Correctness means all the implemented functionalities work as they were in the original application. Ease of use is to check whether users can interact with the transformed app with same efforts or not. Efficiency of execution measures how lightweight is the generated mobile app.

While it is easier to check correctness, checking coverage involves exhaustive enumeration of all test cases. To maximize code coverage unique instrumentation IDs are injected along with instrumented code that help in recognizing services, which were instrumented but not invoked at run time. In practice, for large web applications, such enumeration of test cases may not be possible. To overcome this problem, one can select the important test cases and check correctness for these.

5.1.1 Selection of the Web Application

According to mobile analytics firm Flurry, from 2013-2014 shopping app usage recorded highest growth than any other category of apps [14] which depicts huge demand of eCommerce websites turned mobile apps. Hence, we selected an open source bookstore shopping cart application for evaluation [8]. The application provides functionalities of catalog management, customer management, online retail management, and customer order processing system. It is built using JSP. It follows MVC design pattern and uses MySQL database. It consists of over 14K lines of code. We transformed it to a mobile app in 84 seconds, excluding execution and documentation time that varies from user to user, using a laptop running our tool. Laptop was running on an Intel dual core 64-bit processor with 4GB of RAM. As this is a representative web application of e-retailer and has sufficiently large number of lines of code, we chose is for the case study.

5.1.2 Methodology

We selected ten important functionalities of the bookstore application. Web application is composed of seven different user interaction screens spanning these ten functionalities from a customer's point of view, where customer being a mobile app user in an e-commerce scenario. Hence, these functionalities represent all the steps required to complete a transaction on an e-commerce platform. We instrumented 80 methods during our evaluation. In absence of such MVC controller methods as discussed for PHP, we will need to instrument all 121 methods in the concerned application without the knowledge of candidate services, thereby increasing the time for code instrumentation and execution. The amount of work is less than rewriting 14K+ lines of code into code that follows service oriented architecture and then coding the mobile apps. To check correctness and ease of use, we conducted a survey and asked participants to use the ten functionalities on web application and mobile app. To check correctness, we compared the outputs of both systems. To check ease of use, we asked the participants to give a subjective rating of their difference in experience. Figure 5.3 shows transformation of *View All Products* service from the bookstore application to a mobile app, enumerating all the steps.

We conducted survey on thirty participants. These participants were chosen in two groups. Group 1 consisted of fifteen graduate students, majoring in Computer Science and have expertise in mobile technology. Group 2 consisted fifteen candidates, who are not majoring in Computer Science but have basic familiarity in using smartphones. Each participant first completed the ten functionalities on web application and then performed the same on the re-engineered mobile app. In particular, we asked the following questions to the participants.

1. Whether you were able to successfully execute functionalities via both web and mobile versions?
2. If not, what problem/s they experienced?

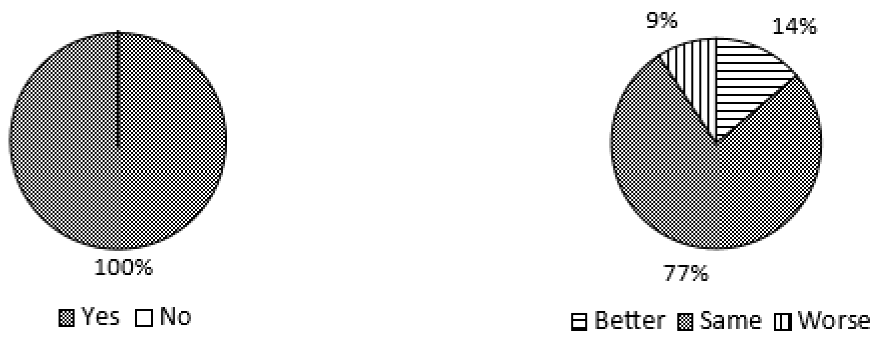


Figure 5.1: Survey analysis for Group 1 (a) Percentage of functionalities successfully executed on the mobile app (b) User experience with the mobile app in comparison with that of web application

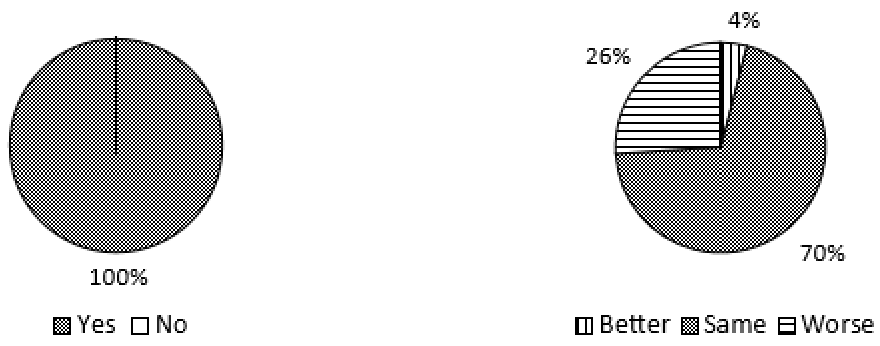


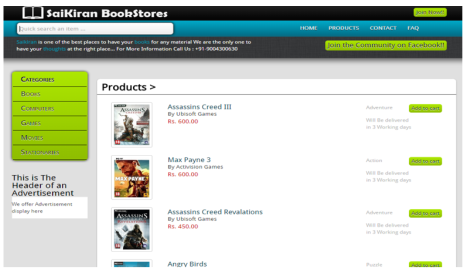
Figure 5.2: Survey analysis for Group 2 (a) Percentage of functionalities successfully executed on the mobile app (b) User experience with the mobile app in comparison with that of web application

3. Whether responses on both applications matched or not? For example, ViewCatalog should display exactly same products and information on both the systems.
4. How was their experience using the mobile application, better, same, or worse?

5.1.3 Results

According to the survey results, both the groups were successfully able to execute the functionalities, using the web application and mobile app. This ensures correctness for the chosen ten functionalities.

At least 70% candidates in both the groups marked their responses to be “same” , in terms of how they felt about using the mobile app. This measures ease of use, given already


SaiKiran BookStores
Join Now!!
Quick search an item ...
HOME
PRODUCTS
CONTACT
FAQ
Saikiran is one of the best places to have your books for any material We are the only one to have your thoughts at the right place... For More Information Call Us : +91-9004300630
Join the Community on Facebook!!
Categories
Books
Computers
Games
Movies
Stationaries
This is The Header of an Advertisement
We offer Advertisement display here
Products >
Assassins Creed III
By Ubisoft Games
Rs. 600.00
Adventure
Add to cart
Will Be delivered in 3 Working days
Max Payne 3
By Activision Games
Rs. 600.00
Action
Assassins Creed Revalations
Rs. 450.00
Angry Birds
Puzzle

(a)

```
<body>
   <%--Code Inserted By Migration Tool--%>
   <%
  DataExtractor.extract(request,response,this.getClass().getSimpleName());
    %>

    <%  if(session.getAttribute("user") == null ){
      // Then new user, show join now %

     <jsp:include page="includesPage/_joinNow.jsp"></jsp:include>
     <% } else { %>

     <jsp:include page="includesPage/_logout.jsp"></jsp:include>
     <% } %>

     <jsp:include page="includesPage/_search_navigationbar.jsp">
     </jsp:include>

     <jsp:include page="includesPage/_facebookJoin.jsp"></jsp:include>
```

(c)

```
{
  "swagger" : "2.0",
  "info" : {
              "version" : "1.0.0"  },
  "host" : "domain.compute.amazonaws.com",
  "basePath" : "/bookstore",
  "schemes" : [ "http" ],
  "paths": {
  "/viewproduct_005f/viewproduct_005.json" : {
     "get" : {
           "tags" : [ "viewproducts_005f" ],
           "summary" : "View All Products",
           "description" : "",
           "operationId" : "method",
           "responses" : {
               "404" : {
                  "description" : "API Not Found" },
               "200" : {
                  "description" : "success" },
               "500" : {
                  "description" : "Server Error" }
             }
           }
```

(e)

```
<body>
    <% if (session.getAttribute("user") == null ){// Then new user, show join now %>

    <jsp:include page="includesPage/_joinNow.jsp"></jsp:include>
    <% } else { %>
    <jsp:include page="includesPage/_logout.jsp"></jsp:include> <% } %>

    <jsp:include page="includesPage/_search_navigationbar.jsp">
    </jsp:include>

    <jsp:include page="includesPage/_facebookJoin.jsp"></jsp:include>

    <div class="container_16">
    <div id = "contents">
```

(b)

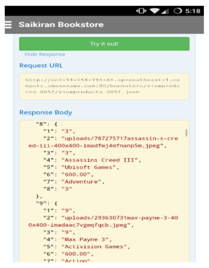
```
@Path("/viewproducts_005f")
@Api(value = "/viewproducts_005f", description = "View All Products")
public class ViewProducts_005f_jsp {

   @GET
   @Path("/viewproducts_005f.json")
   @ApiOperation(value = "View All Products")
   @ApiResponses(value = {@ApiResponse(code = 404, message = "API Not
                                                              Found"),
                          @ApiResponse(code = 500, message = "Server
                                                              Error"),
                          @ApiResponse(code = 200, message = "OK")})

   public Response method(){

      /* RESTful function code */
       return Response.ok(jsonResponse).build();
   }
}
```

(d)



```
http://ec2-54-254-193-84.ap-southeast-1.compute.amazonaws.com:80/bookstore/viewproducts_005f/viewproducts_005f.json
```

```
    "8": {
      "1": "3",
      "2": "uploads/787275717assassin-s-cre
ed-iii-400x400-imadfwj4mfnanp5m.jpeg",
      "3": "3",
      "4": "Assassins Creed III",
      "5": "Ubisoft Games",
      "6": "600.00",
      "7": "Adventure",
      "8": "3"
    },
    "9": {
      "1": "9",
      "2": "uploads/293630731max-payne-3-40
0x400-imadaac7vgwqfqcb.jpeg",
      "3": "9",
      "4": "Max Payne 3",
      "5": "Activision Games",
      "6": "600.00",
      "7": "Action",
```

(f)

Figure 5.3: Bookstore case study stepwise transformation for View All Products service (a) Web interface screenshot (b) Code of web application in JSP embedded along HTML (c) Corresponding instrumented code (d) REST-ful API wrapper using JAX-RS to communicate with legacy code (e) Swagger specification for the generated API (f) Corresponding cross platform mobile app.

existing familiarity with the web app. There is an increase of 17% for responses marked "worse" while using mobile app. This is due to the fact that the mobile app, coming out of the tool, provides basic UI sans the images and icons in the JSP application. The messages from the mobile app are in JSON or XML format and not yet hidden from the user. Although participants from Group 1 are familiar with such messages, those from Group 2 are not. In our future work, we will work on removing the JSON and XML messages and incorporating UI elements in the mobile app. This will improve the mobile app from UI point of view.

The resulting mobile app is platform independent. We successfully deployed and tested it on Android (5.1.1), iOS (version 8), and Windows (version 8.1) mobile platforms. Though native, mobile app projects similar user interface for all the devices. Android-based mobile application executable along with a video demonstration representing mobile app usage on all three mobile platforms is available for download at `https://github.com/rohitmehra/legacymodernization` and transformed web application with REST-ful API wrappers can be viewed at `http://ec2-54-254-193-84.ap-southeast-1.compute.amazonaws.com/bookstore/api-docs`.

Metrics	Web Application	Mobile App
Battery Consumption	333 mW	205 mW
CPU Utilization	26%	10%
Memory Usage	184 MB	98 MB

Table 5.1: Performance comparison between web application running inside a mobile browser and generated native mobile application.

5.2 Performance

We conducted performance evaluation of generated mobile application in comparison to the web application. While the web application was executed on Mozilla Firefox mobile browser (ver. 41.0.2), the mobile application executed natively on the same Android (ver. 5.1.1) device with Snapdragon 810 processor and 4GB RAM. We profiled both systems for one minute of usage while performing the same tasks. We used Power Tutor for measuring battery consumption and GameBench profiler for measuring CPU and memory consumption. Table 5.1 shows the collected data.

Overall, the mobile app was better in using the system resources as it is a native app with only essential libraries.

Both Apache Cordova applications and Web Browser use similar HTML rendering engines (web view) under the hood. But Web Browsers provide lot more features including extensions and plugins, navigation controls, background service, and user interface that bloats its power consumption. This is evident from the mobile app sizes where our mobile app is 2.5 MB in size as compared to 37 MB for the Mozilla Firefox android browser.

6

Related Work

6.1 Related Work

We use REST-ful APIs to covert web applications into services, which are in turn accessed by mobile apps. REST [15] and SOAP [16] are HTTP-based protocols to realize Service Oriented Architecture (SOA) [17]. SOA is a distributed system, where services are deployed on a server and can be accessed via an authorized channel, using intermediary data-exchange formats like JSON and XML over the Internet. The seminal work of Thomas Erl [18] describes concepts, technology, and design patterns, which form the basis of SOA [19], which enables components of a modular application to communicate with each other over HTTP communication protocol. Application services are at the heart of SOA, where each service is responsible for a particular task. Service orientation leads to platform, vendor, and technology independence.

Liu *et al.* [20] talks about the motivation and key issues in performing non-web application to web services migration. Harry Sneed [21] proposed a way to create a wrapper around the non-web application to expose the internal business processes using the web services. Major steps in this process included salvaging the legacy code to extract out business processes, which can then be linked to a wrapper. The wrapper must have interfaces and arguments similar to the legacy business process in order to forward the control to the legacy system and retrieve processed response. This process is semi automated and is based on languages like C and COBOL that are not as complex as the modern web application architectures. It even requires manual intervention and data flow analysis (static) to identify blocks that need to be transformed.

Matos *et al.* [22] leveraged the architecture proposed by Sneed [21] but instead of plain code refactoring, they proposed annotating the source code using specific pattern matching rules, reverse engineering the code to its corresponding graph representation, and then applying graph transformation rules to obtain the re-engineered application. They not just provide existing functionality as web services but do so while complying with the service oriented principles [23]. Researchers at Microfocus defined manual steps required to transform a legacy Cobol application to a Java web application, with the help of cross compilers and visual cobol IDE plugin [24]. Candidate service identification, knowledge gain and wrapper generation needs to be performed manually with the support of provided IDE.

Szymanski *et al.* provide a way of extracting resource model from existing source code and then refining the model to match the final output requirements [25]. Once the resource model is obtained, it is used to generate REST-ful interfaces for accessing these resources [26]. They also present a way to establish data transfer objects that will model the representation of state using intermediary representation formats.

Stroulia *et al.* models behavior of legacy user interface as a state-transition diagram and derives application specifications based on frequently occurring interaction patterns. But it focuses on accessing services through WWW interfaces rather than REST-ful APIs as in our approach [27]. It also talks about front-end form generation for accessing information through WWW. Marchetto *et al.* looks at the process from a manual perspective and informs about the steps and tools required to perform such migration manually and emphasized on testing for correctness [28]. Zhen *et al.* provides an evidence as to why web browsers are slow on mobile devices [29]. This urges a need to go native rather than developing mobile website.

7

Conclusion and Future Work

7.1 Conclusion and Future Work

We presented Kirke that uses a novel approach for re-engineering web application using server-side scripting language to cross mobile platforms using dynamic extraction of services. Two major contributions of this research includes employing dynamic analysis instead of static analysis to increase generalizability for different frameworks and languages and automated conversion of REST-ful APIs to cross platform native mobile apps. We evaluated correctness and coverage of Kirke by employing it to re-engineer a publicly available large JSP web application into mobile apps for Android, iOS, and Windows Mobile in less than two minutes.

Our approach works the best when the web application is written using MVC design pattern as it requires minimal amount of code instrumentation. Among the web applications that are written using server-side scripting languages, the majority ones are written using JSP, ASP.NET, or PHP. While JSP and ASP.NET follow MVC design pattern out of the box, PHP does not. In the latter case, Kirke instruments code in all the methods, thereby taking relatively more time. We plan to use dynamic analysis techniques to find out commonly used entry points in PHP code. Kirke will instrument code at these entry points to speed up the re-engineering process by sacrificing coverage.

Unmarshalling response produced by legacy JSP code provides only values and not the keys which makes responses hard to analyze. In future we intend to address this issue by designing mechanisms to automatically extract keys from the source code. We also plan to work on the usability aspect of our mobile app and automatically incorporate design features from the web application's design, to provide a customized feel and experience.

References

[1] C. V. Mobile, “Cisco visual networking index: Global mobile data traffic forecast update, 2013–2018,” *San Jose, CA*, 2014. 1

[2] CioInsight, “Mobility tops the list of digital priorities,” 2014. [Online]. Available: http://www.cioinsight.com/it-strategy/mobile-wireless/slideshows/mobility-tops-the-list-of-digital-priorities.html 1

[3] A. Lipsman, “Major mobile milestones in may: Apps now drive half of all time spent on digital,” 2014. 1

[4] InMobi, “Global mobile media consumption wave 3 report,” 2014. [Online]. Available: http://info.inmobi.com/rs/inmobi/images/Global%20Mobile%20Media%20Consumption%20Wave%203%20Report.pdf 1

[5] T. Magazine, “Half a billion facebook users only visit on mobile devices,” 2015. [Online]. Available: http://time.com/3686677/facebook-mobile 1

[6] I. . M. A. of India, “Mobile internet report 2014,” 2014. [Online]. Available: http://www.iamai.in/PRelease_detail.aspx?nid=3498&NMonth=11&NYear=2014 1

[7] F. Times, “Indian flipkart to go app-only,” 2015. [Online]. Available: http://www.ft.com/intl/cms/s/0/b8deaa7c-2549-11e5-9c4e-a775d2b173ca.html 1

[8] “Saikiran bookstore shopping cart.” [Online]. Available: http://deadnight7.github.io/saikiranBookstoreApp 2, 25

[9] “Swagger: Rest documentation framework.” [Online]. Available: http://swagger.io 5

[10] R. Morales-Chaparro, M. Linaje, J. Preciado, and F. Sánchez-Figueroa, "Mvc web design patterns and rich internet applications," *Proceedings of the Jornadas de Ingenierıa del Software y Bases de Datos*, pp. 39–46, 2007. 8

[11] "Swagger codegen." [Online]. Available: http://swagger.io/swagger-codegen 13, 15

[12] A. Cordova, "Apache cordova, learn more about the project," 2012. [Online]. Available: http://cordova.apache.org 14

[13] A. Mishra, "Critical comparison of php and asp .net for web development," *International Journal Of Scientific & Technology Research*, vol. 3, no. 7, 2014. 19

[14] "Shopping, productivity and messaging give mobile another stunning growth year." [Online]. Available: http://flurrymobile.tumblr.com/post/115194992530/shopping-productivity-and-messaging-give-mobile 25

[15] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000. 30

[16] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (soap) 1.1," 2000. 30

[17] G. Mulligan and D. Gravanin, "A comparison of soap and rest implementations of a service based interaction independence middleware framework," in *Simulation Conference (WSC), Proceedings of the 2009 Winter.* IEEE, 2009, pp. 1423–1432. 30

[18] T. Erl, *Service-oriented architecture: concepts, technology, and design.* Pearson Education India, 2005. 30

[19] R. Perrey and M. Lycett, "Service-oriented architecture," in *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on.* IEEE, 2003, pp. 116–119. 30

[20] Y. Liu, Q. Wang, M. Zhuang, and Y. Zhu, "Reengineering legacy systems with restful web service," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International.* IEEE, 2008, pp. 785–790. 30

[21] H. M. Sneed *et al.*, "Wrapping legacy software for reuse in a soa," in *Multikonferenz Wirtschaftsinformatik*, vol. 2, 2006, pp. 345–360. 30

[22] C. Matos and R. Heckel, “Migrating legacy systems to service-oriented architectures,” *Electronic Communications of the EASST*, vol. 16, 2009. 30

[23] T. Erl, *Soa: principles of service design.* Prentice Hall Upper Saddle River, 2008, vol. 1. 31

[24] “Take your step-by-step journey from cobol to mobile.” [Online]. Available: https://www.microfocus.com 31

[25] C. Szymanski and S. Schreier, “Case study: Extracting a resource model from an object-oriented legacy application,” in *Proceedings of the Third International Workshop on RESTful Design.* ACM, 2012, pp. 19–24. 31

[26] M. Gulden and S. Kugele, “A concept for generating simplified restful interfaces,” in *Proceedings of the 22nd international conference on World Wide Web companion.* International World Wide Web Conferences Steering Committee, 2013, pp. 1391–1398. 31

[27] E. Stroulia, M. El-Ramly, and P. Sorenson, “From legacy to web through interaction modeling,” in *Software Maintenance, 2002. Proceedings. International Conference on.* IEEE, 2002, pp. 320–329. 31

[28] A. Marchetto and F. Ricca, “From objects to services: toward a stepwise migration approach for java applications,” *International journal on software tools for technology transfer*, vol. 11, no. 6, pp. 427–440, 2009. 31

[29] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, “Why are web browsers slow on smartphones?” in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications.* ACM, 2011, pp. 91–96. 31

[30] A. A. Almonaies, J. R. Cordy, and T. R. Dean, “Legacy system evolution towards service-oriented architecture,” in *International Workshop on SOA Migration and Evolution*, 2010, pp. 53–62.

[31] R. M. Babu, M. B. Kumar, R. Manoharan, M. Somasundaram, and S. P. Karthikeyan, “Portability of mobile applications using phonegap: A case study,” 2012.

[32] K. Bennett, “Legacy systems: coping with stress,” *Software, IEEE*, vol. 12, no. 1, pp. 19–23, 1995.

[33] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, “Evaluating cross-platform development approaches for mobile applications,” in *Web information systems and technologies.* Springer, 2013, pp. 120–138.

[34] Mulloy, “Api facade pattern - a simple interface to a complex system,” 2012. [Online]. Available: https://pages.apigee.com/api-facade-pattern-ebook.html

[35] B. Mulloy, “Web api design: Crafting interfaces that developers love,” 2012.

[36] D. Seven, “What is a hybrid mobile app,” *Icenium. Blog publication at http://bit.ly/OMVQVN accessed*, vol. 23, no. 9, p. 2012, 2012.

[37] J. M. Wargo, *Apache Cordova 4 Programming.* Pearson Education, 2015.

[38] “Swagger ui.” [Online]. Available: http://petstore.swagger.io

[39] N. Sinha and R. Karim, “Responsive designs in a snap,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ACM, 2015, pp. 544–554.

Appendix A

Survey Questionnaire

The following functionalities need to be performed on both Web and Mobile versions of Bookstore application and feedback should be recorded as per the form.

A.1 Register A New User

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.2 Login User

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No

- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.3 View User Details

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.4 Edit User Details

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.5 Change User Password

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.6 View All Products

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.7 View Specific Products Details

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.

- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.8 Add Product To Shopping Cart

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.9 View Shopping Cart Contents

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse

A.10 Log Out User

- Were you able to execute functionality successfully on Mobile Device?
  - Yes
  - No
- If No, mention the problem/s you experienced.
- How was the experience on Mobile Device?
  - Better
  - Same
  - Worse