Generator Code Opaque Recovery of Form-Oriented Web Site Models

Dirk Draheim, Christof Lutteroth
Institute of Computer Science
Freie Universität Berlin
Takustr.9, 14195 Berlin, Germany
draheim@acm.org lutterot@inf.fu-berlin.de

Gerald Weber
Department of Computer Science
The University of Auckland
38 Princes Street, Auckland 1020, New Zealand
g.weber@cs.auckland.ac.nz

Abstract

This paper describes source-code independent reverse engineering of dynamic web sites. The tool Revangie constructs a form-oriented analysis model solely from the usage of a web application. The recovered models can be, for example, exploited for the purpose of requirements engineering and load test development. Revangie can explore a given web application fully automatically or can passively record its usages. The collected data, i.e., data about screens, server-side programs, and system responsiveness, are analyzed in order to build a user interface model. The paper presents several useful classification techniques that can be utilized to reconstruct adequate models.

1. Introduction

This article presents Revangie – a tool that is able to recover the model of a dynamic web interface of a web site without looking at the source code. We choose to recover models independently of source code because of the manifold of languages, platforms, and architectures a dynamic web site can be implemented in. Tools that perform source code dependent reverse engineering are usually restricted to a certain language, platform etc., whereas Revangie is independent of all those. It is much easier to analyze HTML code than the source code that generates it and also more convenient because the HTML code can be explored through the single point of access of an HTTP port, whereas the generating code can have a complex deployment structure. In the case that the source code is inaccessible, analysis must be source code independent anyway, as it is the case in typical product benchmarking efforts. It is an essential claim that the analysis of the generated HTML is sufficient to recover sophisticated models.

We explain the motivation for Revangie by describing its role in the Angie tool suite. Revangie is used to recover form-oriented models from a dynamic web site [1]. The textual description of this model in the language Angie [1] can be subsequently used for forward engineering of click dummies, as they are conveniently used for requirements engineering, or customizable systems, which can help in migrating to model-driven architecture. In addition to this, Revangie can collect data about user behavior that can be used for load testing. It is further work to provide the load test tool Angil that simulates real users on the basis of an annotated version of the Angie language.

In order to perform coherent analysis, we need a model that describes the user interface of web applications adequately. The form-oriented user interface model [1] uses typed, bipartite state machines in which one set of states denotes client pages and the other set denotes server actions that generate the pages. These graphs contain all the information of simple page diagrams, but in addition to this they model the relationship between server-side actions and pages. Note that the pages in the model do not represent individual screens as seen by the user, but classes of screens of which many instances may be generated by different server actions. An important property of the model is its type system: actions as well as pages have a signature which specifies a type for the data accepted by an action and a type for the data displayed on the screens of a page. These signatures help us in determining which forms and links invoke which actions. Furthermore they help us in identifying similar actions and similar screens. An action is uniquely identified by the program that is invoked, which is referenced in the URI, plus the set of labels with which parameters are passed to it through the CGI.

2. The Reverse Engineering Process

Revangie can work in different modes: i) the crawl mode, which works on the client side, and ii) the snoop mode, which works at any point in the communication line between client and server and the guide mode. The crawl mode works like an automated web browser: it uses an

Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)
1095-1350/04 $20.00 © 2004 **IEEE**

HTTP client to request pages, submit values and analyze the trace of submitted values and visited pages. It starts at a specified URL, retrieves the corresponding HTML screen and scans it for links, forms and some other tags. Then we can infer which action is targeted by each link and form. Each screen in turn must be classified as being an instance of a certain page. In this way, we can obtain information about which pages and actions are in the model, which action can be invoked from which page and which page is generated by which action. It would be naive to think that the analysis model of every non-trivial web-application could be reconstructed automatically. The problematic point is the data submitted on a page, which usually has to suffice certain constraints and therefore cannot be generated generically. The snoop mode collects data of actual sessions of a web application in order to analyze it and reconstruct a model afterwards. It can either monitor the HTML communication of one user by taking the role of a proxy server, or monitor the communication of all users by taking the role of a facade to the web server. This mode allows us to collect realistic session data of either one or a multitude of users. In the case of many users, it can therefore be utilized to determine certain user model parameters statistically. An analysis model with this additional data, like distributions for the turnaround time of a page, navigation probabilities or sample input, is ideal for performing fully automated realistic tests, especially load-testing. The guide mode tries to combine the advantages of the crawl and single-user snoop modes: i) automation and ii) the possibility to enter form data manually. This means that at certain points that are hard to handle automatically, the user is asked to choose the next form or link and to enter appropriate values.

3. Screen Classification

We need to classify each received screen to a page in order to construct an adequate model. There are different equivalence relations that can be used for screen classification. The equivalence relations together with the subset relationship form a lattice.

Trivial identity is the coarsest possible equivalence relation. All screens are equivalent. It is practically unimportant, but forms the top ($\top$) element of the lattice.

Screen identity is the finest possible equivalence relation for screen classification and consequently forms the bottom ($\bot$) element. Each screen that is received by a web client gets its own page, even if two screens have the same HTML code.

Textual identity groups screens with the same HTML code into the same page.

Source identity groups screens into the same page that were generated by the same action.

Targets identity groups screens with identical targets signature, i.e., the same set of signatures of the server actions targeted by a screen. It is an important basic classification because it produces form-oriented models with page-action transitions that are valid in each case. Targets identity can be coarsened by excluding forms and links that target external actions, like links to other web sites.

Form targets identity. Targets identity can be coarsened to form targets identity by excluding the signatures of links from the targets signature.

Title identity groups screens with identical HTML titles.

Pattern identity groups screens that match a user-defined pattern. This may be a textual pattern, a purely syntactical pattern or a mixture of both; regular or at most context-free patterns are usually sufficient.

Similarity or dissimilarity of screens according to some textual or structural distance metric can be used to cluster them into pages. A similarity measure suitable for clustering can also be created by composition of single similarity indicators, like any of the discussed ones, which can be combined, for example, by a weighted sum.

A *conjunction* $A \wedge B$ groups those screens that are equivalent by both A and B, yielding a refinement of both A and B.

A *disjunction* $A \vee B$ groups those screens that are equivalent by A or B, yielding a coarsening of both A and B.

Our classification apparatus is especially expressive because of the notions of conjunction and disjunction. Potential candidates for conjunction are, for example, targets identity and title, or targets identity and pattern identity. By combining and configuring classification techniques adequately it is possible to reconstruct an expressive model with adequate page granularity. Note that it depends on the intended usage of a model which model granularity is adequate, i.e., a given model granularity must be judged with respect to an external criteria. For example, targets identity can be considered to fit best many load testing scenarios, whereas textual identity is the classification of choice for static HTML pages.

References

[1] D. Draheim, C. Lutteroth, and G. Weber. Source Code Independent Reverse Engineering of Dynamic Web Sites. Technical Report B-04-10, Institute of Computer Science, Freie Universitt Berlin, June 2004.

[2] D. Draheim and G. Weber. *Form-Oriented Analysis - A New Methodology to Model Form-Based Applications*. Springer, September 2004.