ProCrawl: Mining Test Models from Multi-user Web Applications

Matthias Schur
SAP
Darmstadt, Germany
matthias.schur@sap.com

Andreas Roth
SAP
Karlsruhe, Germany
andreas.roth@sap.com

Andreas Zeller
Saarland University
Saarbrücken, Germany
zeller@cs.uni-saarland.de

ABSTRACT

Today's web applications demand very high release cycles—and consequently, frequent tests. Automating these tests typically requires a *behavior model:* A description of the states the application can be in, the transitions between these states, and the expected results. Furthermore one needs *scripts* to make the abstract actions (transitions) in the model executable. However, specifying such behavior models and writing the necessary scripts manually is a hard task. We present ProCrawl (Process Crawler), a tool that automatically mines *(extended) finite-state machines* from (multi-user) web applications and generates executable test scripts. ProCrawl explores the behavior of the application by systematically generating program runs and observing changes on the application's user interface. The resulting models can be directly used for effective model-based testing, in particular regression testing.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*testing tools, tracing*

General Terms

Algorithms, Design, Experimentation

Keywords

Specification mining; dynamic analysis; model-based testing

1. INTRODUCTION

Today's web applications are characterized by a high frequency of updates. To prevent such updates from breaking functionality, one has to test—thus frequent updates call for frequent tests. Automatically generating such tests requires a model describing the possible and the expected application behavior. However, typically web applications come without explicit models, which implies mostly manual and thus less efficient test creation—and also slows down understanding and maintenance of the application.

The field of *specification mining* aims to facilitate these activities by mining abstractions from programs and their executions—typically, models of the program's behavior. If these models are precise enough, they can even be used as post-facto specifications of the program and test engineers can apply them in a continuous integration environment to check for regressions after code changes. Specification mining has been used to successfully derive axiomatic specifications such as function and data invariants from programs [6], or finite-state machines describing states and transitions for individual classes [4, 5]. For such small-scale domains, it is fairly easy to validate specifications, because both program code and program state are accessible and amenable to symbolic reasoning and exhaustive testing.

Extracting models on system level is much more difficult. Program code and program state, for instance, may not be available for analysis, as the application may be distributed across several layers and sites. In general, the only assumption that can be made is that there is some user interface (UI) such as a web interface that allows for human interaction.

ISSTA'14, July 21–25, 2014, San Jose, CA, USA
ACM 978-1-4503-2645-2/14/07
http://dx.doi.org/10.1145/2610384.2628051

2. A MODEL MINER FOR WEB APPS

ProCrawl is a fully automatic tool for mining behavior models of web applications[1] and generating executable test scripts for system testing and validation. We discussed and evaluated our approach in [10], here we focus on the tool. Section 2.1 introduces the behavior models and describes the testing process facilitated by ProCrawl. The model quality strongly depends on the applied abstraction mechanisms and the selection of actions to be executed, which are described in Section 2.2 and Section 2.3, as well as the ability to resolve nondeterminism, which is discussed in Section 2.4.

2.1 Testing Web Applications with ProCrawl

To mine processes involving interacting users, ProCrawl instantiates multiple *actors* operating the system under test (SUT) via a web browser. ProCrawl is a black-box approach, i.e. it mines application behavior without requiring access to the source code of the SUT.

The resulting behavior model is an *extended finite-state machine* (EFSM) [3] in which a node denotes an abstract individual state of the web application, numbered in the order it was detected by ProCrawl, and a transition denotes a sequence of UI interactions that changes the state and is

[1] ProCrawl handles rich Web 2.0 applications, including dynamic technologies such as AJAX.

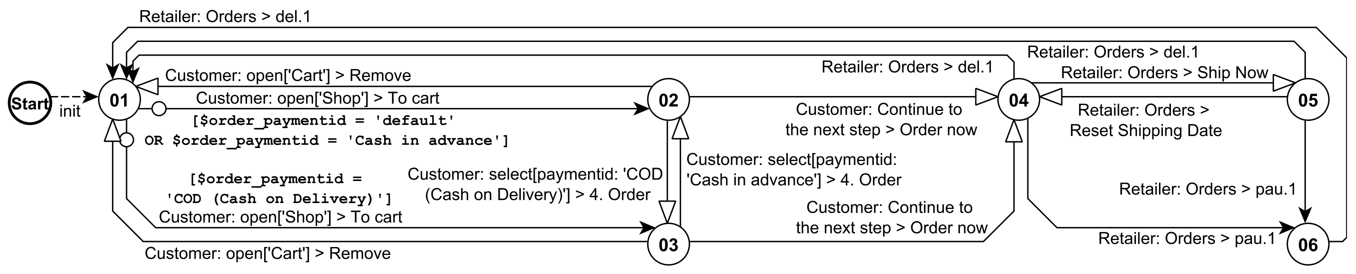
Retailer: Orders > del.1
Retailer: Orders > del.1
Retailer: Orders > Ship Now
Customer: open['Cart'] > Remove
Retailer: Orders > del.1
Start
init
01
Customer: open['Shop'] > To cart
02
Customer: Continue to the next step > Order now
04
Retailer: Orders > Reset Shipping Date
05
[$order_paymentid = 'default' OR $order_paymentid = 'Cash in advance']
Customer: select[paymentid: 'Cash in advance'] > 4. Order
[$order_paymentid = 'COD (Cash on Delivery)']
Customer: select[paymentid: 'COD (Cash on Delivery)'] > 4. Order
Retailer: Orders > pau.1
Customer: open['Shop'] > To cart
Customer: Continue to the next step > Order now
Retailer: Orders > pau.1
03
06
Customer: open['Cart'] > Remove

Figure 1: The ProCrawl model of the ordering process in OXID eShop, involving a customer and a retailer. Transitions are labeled with the action that causes the state change (del.1=delete; pau.1=cancel), as well as a transition guard that defines under which conditions the transition is enabled. For instance, transition (1a) is enabled if paymentid = default $\vee$ paymentid = 'Cash in advance' holds; if paymentid = 'COD (Cash on Delivery)', transition (1b) is enabled instead.

performed by one of the actors. As an example, Figure 1 shows an EFSM extracted by ProCrawl[2] that models the ordering process in OXID eShop[3], a popular open source e-commerce platform: Customers can add (Transition 1a, 1b) and remove (9, 10) products from their shopping cart, select a payment method (2, 6) and submit the order (3, 7). Incoming orders can be shipped (4), canceled (12, 14) or deleted (5, 8, 13) by the retailer via the shop back-end UI.

Figure 2 illustrates the testing process facilitated by ProCrawl, which is detailed as follows.

1. ***Tool Configuration.*** To mine the model depicted in Figure 1, ProCrawl requires a configuration including

   - the participating *actors*: in our example a customer with the URL to the shop front-end and a retailer with the URL to the admin UI, including login credentials.
   - a *start action* which is executed by one of the actors (Customer: open['Shop'] > To cart in our example).
   - an *exploration scope* definition for each actor, i.e. the views of the web application to be checked for changes after executing an action. In our example the URL to the shopping cart and the URL to the order history of the customer, as well as a sequence of actions to reach the order list with the retailer.
     If no scope definition is provided, ProCrawl simply checks the page that is rendered in the browser before and after executing an action, as it is done by web crawlers such as Crawljax [8].
   - an optional *test fixture*[4] to set the SUT to the initial state, which may be necessary to complete behavioral exploration after a nonreversible action (i.e. entering a dead-end in the EFSM).

   The default configuration can be adjusted for each actor by specifying the type and number of browsers, maximum click depth and HTML elements to be considered or ignored, as well as the input elements for which different values are generated. State abstraction can be changed by providing DOM filters and selectors, and test oracles can check the SUT for errors during behavior exploration.

2. ***Automatic Behavior Exploration.*** The exploration process works fully automatically; ProCrawl can mine test models for real-world web applications without requiring any user intervention. The process can be observed live: for each actor one or multiple web browsers are started, on which the UI interactions are performed. Every time ProCrawl detects a new (nonreflexive) transition, the performed actions are exported as a Selenium script, which can be executed with the Selenium IDE[5] or Java/command line tools such as SeleneseRunner[6]; the behavior model is exported in GraphML[7] format, which can be displayed in a graph editor (cf. Figure 1). Additionally ProCrawl exports states in text format, such that they can be easily compared using a diff tool. The exploration process can be traced via a detailed runtime log and continues until the set of actions to be explored is empty or a configurable stop condition holds.

3. ***Model Validation.*** ProCrawl mines a model by observing the behavior of a web application. This model serves as a "*gold standard*" oracle [2] for automatically generating executable system tests detecting regressions which eliminate behavior from one revision of the application to the other. However, in absence of configured test oracles, ProCrawl does not know if the observed behavior is correct (*oracle problem*)—except for strong indicators such as crashes. Therefore the model must be validated before using it for regression testing.

4. ***Regression Test Generation.*** The behavior models mined by ProCrawl can be directly used for automatically generating regression tests [3], which can be executed via the generated Selenium scripts. A failing test either means the behavior of the SUT changed disruptively, or the UI changed, making the Selenium scripts fail, in which case the scripts have to be regenerated or updated manually. In our evaluation on three real-world web applications the generated tests detected both, disruptive process changes as well as changes on the UI [10].

2.2 State and Event Abstraction

Web applications typically separate functionality in different *views*, which can be accessed by executing *navigational*

[2] For better readability we numbered the transition labels.
[3] http://www.oxid-esales.com/en
[4] Typically an SQL script resetting the database of the SUT.
[5] http://seleniumhq.org/projects/ide
[6] https://github.com/vmi/selenese-runner-java
[7] http://graphml.graphdrawing.org

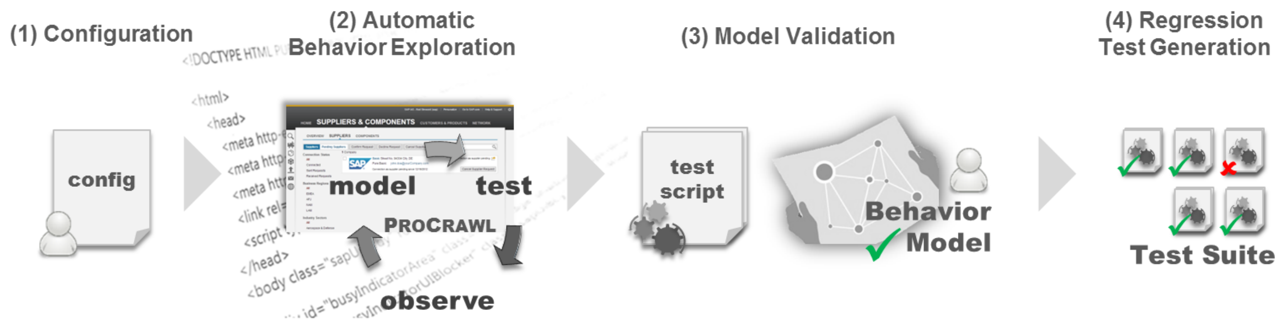

Figure 2: The testing process with ProCrawl. Given a configuration (1) ProCrawl automatically explores the application behavior (2) generating executable test scripts and a behavior model, which is validated by the user (3) and serves as a "gold standard" oracle for generating regression tests (4).

actions on the UI. To extract a model describing the business logic of the SUT that is exposed via the UI, ProCrawl abstracts over the visible UI elements to identify similar states and excludes navigational actions from the behavior model. Without proper abstraction every single change in the UI presentation would lead to a distinct state.

State Abstraction. The assumption of the state abstraction mechanism applied by ProCrawl is that the current state of the SUT is reflected in the UI, where each actor/view relation exposes a specific part of the application state. ProCrawl distinguishes different states of the SUT using an abstraction function over the state of the *Document Object Models* (DOMs) of these actor/view relations; concrete DOM states are partitioned by the presence of specific UI elements. This allows ProCrawl to determine the application state without requiring access to the SUT's source code. The set of views (*exploration scope*) is configured for each actor. ProCrawl automatically navigates to these views with each actor in parallel, extracts the DOMs and computes the state by applying filters on the set of DOMs. In its default configuration ProCrawl abstracts over visible hyperlinks, buttons and text extracted from the DOMs. However, the type of HTML elements to be considered for state abstraction can be changed by providing XPath expressions or CSS selectors.

Action Abstraction. ProCrawl classifies actions that do not change the observed state of the SUT (self-loops in the EFSM) as navigational. These actions are specific to the SUT's UI and therefore volatile throughout the application lifecycle. However, since process scenarios typically span several views, ProCrawl may need to execute navigational actions in order to activate an HTML element performing a functional action. So the derivation of a single step in the process scenario may require the execution of multiple navigational actions plus one functional action. ProCrawl addresses this issue by introducing two layers of abstraction: a *model layer* which is independent of navigational actions, and executable *scripts* as a second layer, binding navigational to functional actions. In this way, if the UI of the SUT changes, it is sufficient to update the navigational actions in the generated scripts only; the model itself remains untouched.

2.3 Causality-Driven Behavior Exploration

ProCrawl maintains a set of *pending actions* $A_p(s_i)$ for each state s_i. During behavior exploration, it removes and executes an action from $A_p(s_c)$, where s_c is the current state of the SUT, determines the new application state s_n, updates $A_p(s_n)$ and the EFSM and continues until A_p is empty or a configurable stop condition holds. Adding click actions for a static (configured) set of HTML elements in s_i to $A_p(s_i)$ as it is done by web crawlers such as Crawljax [8], would result in executing and finally combining unrelated actions in a single model. Therefore ProCrawl computes $A_p(s_i)$ by generating actions for a configurable subset of HTML elements in $E_b(s_i) \setminus E_b(s_0)$, where $E_b(s_i)$ is the multiset of elements in state s_i, s_0 is the initial state, and $A_p(s_0) = \{a_0\}$ with a_0 being a configured start action. Thus $A_p(s_i)$ only contains actions with a causal relationship to a_0. Consequently the behavior models only contain actions that are part of a causal chain as it is common in processes; resulting in modularized models that are easier to maintain.

ProCrawl implements a weighted nearest neighbor algorithm to systematically explore the behavior graph, i.e. if $A_p(s_i) = \emptyset$, it computes a path to the pending state with the smallest distance from s_i, weighted in terms of UI interactions to be performed. If there is no known path from s_i to a pending state or a configurable reset condition holds, ProCrawl resets the SUT to s_0 by executing the test fixture provided in the configuration.

We evaluated the effectiveness of this approach on three real-world web applications, where ProCrawl inferred behavior models with high *precision* (≥ 0.82), *recall* (≥ 0.75) and *accuracy* (≥ 0.86) [10].

2.4 Transition Guard Learning

With the procedure described above, ProCrawl can already derive suitable models. Sometimes however the extracted state machine is nondeterministic. While this is natural as the system state is only partially reflected in the UI, nondeterministic transitions lead to problems during test generation. For instance selecting Cash on Delivery as payment method in Figure 1 (Transition 2) adds a surcharge which is reflected in State 03. After ordering (3), the selected payment method is not reflected in the UI anymore. Adding an item to the shopping cart in State 01 after removing a product from the cart (10) or deleting the order (5, 8, 13), remembers the payment method, leading to another To cart transition (1b). However, model-based testing usually fails if the state of the SUT is not well defined in the model after executing an action. Therefore ProCrawl strives to resolve nondeterminism by inferring mutually exclusive *guard conditions* on extended state variables computed from data input elements on the UI and the input data generated during behavior exploration.

These guard conditions must evaluate to true in order to enable the transition. For instance the To cart transition (1b) is only enabled in State 01 iff Cash on Delivery has been selected as payment method, which becomes an extended state variable (order_paymentid). Nondeterministic transitions are then annotated with the inferred guards, producing an extended finite-state machine [3]. ProCrawl infers these guards by creating a classification problem over the path history of the nondeterministic transitions, and iteratively generates additional runs to produce samples for learning a decision tree classifier [9] (*active learning*), which is then transformed into the guard conditions shown in Figure 1 (square brackets). The details will be discussed in an upcoming paper.

3. IMPLEMENTATION

ProCrawl is implemented in Java, using Selenium[8] for web browser automation. For building the classifier used to infer transition guards, we use a variation of the C.4.5 algorithm [9] provided by Weka[9]. To increase the number of samples for building the classifier, we transform the EFSM into a B machine [1] and use the ProB [13] model checker to generate additional paths with untested input values.

4. RELATED WORK

There are various approaches that infer finite-state machines (FSMs) capturing program behavior as sequencing of events (cf. [11]), most of them working on object level (cf. [4]). Only few approaches [7, 12] mine EFSMs by combining FSM inference with data rule inference. The major difference to these techniques is that ProCrawl systematically generates program runs to learn application behavior ("experimental analysis" [14]), while most dynamic approaches rely on a given set of program runs (traces); this allows ProCrawl to verify its hypotheses and explicitly control the exploration scope, which would otherwise be implicitly induced by the traces. In this regard ProCrawl is similar to Crawljax [8], a state of the art web crawler targeting AJAX sites. State-flow graphs extracted by Crawljax depict the various navigational paths and UI/DOM states within an AJAX site; a node represents a single (abstract) runtime DOM state of an AJAX site and transitions represent single UI events. However, restricting state abstraction to a single DOM state of a single user, limits the number of actions that can be detected by the crawler and often leads to a nondeterministic FSM, prohibiting effective model-based testing. In the models extracted by ProCrawl, a node represents multiple abstract DOM states (views) of multiple users and transitions refer to generated scripts encapsulating UI specific navigational paths and functional actions. Furthermore nondeterminism is effectively eliminated by inferring transition guards over the input data.

5. CONCLUSION

ProCrawl is a fully automatic tool to mine behavior models including executable test scripts from multi-user web applications for regression testing and system validation. Its state abstraction is specific enough to capture essential process steps, including cycles; yet generic enough to be applicable to a diverse range of web applications.

[8] http://seleniumhq.org

[9] http://www.cs.waikato.ac.nz/ml/weka

6. ACKNOWLEDGMENTS

The work presented herein is partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. `01IC12S01`. We thank Anne Keller for useful comments on an earlier revision of this paper.

7. REFERENCES

[1] J.-R. Abrial. *The B-book: Assigning Programs to Meanings.* Cambridge University Press, New York, USA, 1996.

[2] R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Object Technology Series. Addison Wesley, 1999.

[3] K. T. Cheng and A. S. Krishnakumar. Automatic functional test generation using the extended finite state machine model. In *Proc. of the 30th International Design Automation Conference*, DAC '93, pages 86–91, New York, USA, 1993. ACM.

[4] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: extracting finite-state models from java source code. In *ICSE*, pages 439–448, New York, USA, 2000. ACM.

[5] V. Dallmeier, N. Knopp, C. Mallon, S. Hack, and A. Zeller. Generating test cases for specification mining. In *ISSTA*, pages 85–96, New York, USA, 2010. ACM.

[6] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Trans. Softw. Eng.*, 27(2):99–123, 2001.

[7] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE*, pages 501–510, New York, USA, 2008. ACM.

[8] A. Mesbah, A. Van Deursen, and S. Lenselink. Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *ACM Trans. Web*, 6(1):1–30, Mar. 2012.

[9] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Francisco, CA, USA, 1993.

[10] M. Schur, A. Roth, and A. Zeller. Mining behavior models from enterprise web applications. In *ESEC/SIGSOFT FSE*, pages 422–432, 2013.

[11] N. Walkinshaw, B. Lambeau, C. Damas, K. Bogdanov, and P. Dupont. Stamina: A competition to encourage the development and assessment of software model inference techniques. *Empirical Softw. Eng.*, 18(4):791–824, Aug. 2013.

[12] N. Walkinshaw, R. Taylor, and J. Derrick. Inferring extended finite state machine models from software executions. In *WCRE*, pages 301–310, 2013.

[13] S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker. Applying model checking to generate model-based integration tests from choreography models. In *TestCom/FATES*, pages 179–194, 2009.

[14] A. Zeller. Program analysis: A hierarchy. In *ICSE Workshop on Dynamic Analysis (WODA 2003)*, pages 6–9, 2003.