МЕТОД ПОВЫШЕНИЯ КАЧЕСТВА ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ АВТОМАТНОГО ПОДХОДА

А. Ю. Законов

аспирант кафедры компьютерных технологии;
andrew.zakonov@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В течение последних лет существенно выросла как сложность веб-приложений, так и требования к их надежности. Зачастую по причине сжатых сроков разработки и часто меняющихся требований заказчика проверка качества веб-приложений осуществляется вручную без формальных критериев. Отсутствие моделей и формальной спецификации делает процесс проверки качества трудоемким и не позволяет гарантировать отсутствие ошибок. В статье предложен метод автоматического выделения явных состояний и построения автоматной модели для веб-приложений при помощи динамического анализа. Наличие автоматной модели позволит формализовать требования спецификации и использовать Model Checking для автоматизации процесса тестирования. Также разработан алгоритм поиска схожих состояний, благодаря которому даже для сложных веб-приложений, возможно автоматически строить модели с небольшим количеством состояний, которые будут понятны разработчику.

Введение

Сложность веб-приложений стремительно растет так же, как и требования к их надежности и безопасности. Вопрос проверки качества веб-приложений является актуальным, так как множество программ реализуются в виде веб-приложений и многие из них связаны с финансовыми транзакциями и конфиденциальной информацией: интернет-магазины (Amazon.com), интернет-банкинг, почтовые клиенты (GMail), социальные сети (Facebook), системы документооборота (Alfresco) и многие другие.

Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, обмен информацией происходит по сети [1]. Тестирование веб-приложений так же, как и тестирование пользовательских интерфейсов, тяжело автоматизировать, так как это требует эмуляции действий пользователя, таких как ввод текста, заполнение форм, переходы по ссылкам. Для сложных веб-приложений различных последовательностей действий пользователя может быть бесконечно много, поэтому проверить все варианты взаимодействия пользователя с системой невоз-

можно. На практике зачастую проверяется только небольшое подмножество возможных вариантов действий пользователя, что оставляет возможность появления разнообразных критических ошибок.

В статье предлагается метод автоматизации тестирования веб-приложений, основанный на автоматном подходе. Конечные автоматы позволяют удобно описывать взаимодействие системы с пользователем [2, 3]. Состояние веб-приложения — веб-страница, которую видит пользователь. Входные воздействия, которые могут менять состояние приложения, — это действия пользователя и ответы сервера на AJAX-запросы. Состояние приложения может меняться за счет перехода на новую веб-страницу или путем динамического изменения структуры (DOM-дерева) текущей страницы.

Исходный код приложения является наиболее точным описанием логики работы, но это представление неудобно для тестирования, так как не позволяет понять высокоуровневую логику приложения. Для этой цели в статье предложен метод автоматического построения автоматных моделей существующих веб-приложений. Автоматная модель обладает следующими преимуществами:

- описывает логику работы приложения в визуально понятном и читаемом формате;
- позволяет удобно записать требования спецификации, используя формальную логику, например, язык LTL, и автоматически проверить их, используя Model Checking;
- пригодна автоматической генерации тестов с заданными критериями (покрытие всех переходов/состояний).

Цель исследования: предложить метод автоматического построения автоматной модели веб-приложения, метод применения Model Checking и автоматизации тестирования для повышения качества веб-приложений.

В статье решаются следующие задачи:

1. Разработать метод автоматического выделения явных состояний и переходов между ними для веб-приложений.
2. Оценить применимость методов Model Checking для автоматных моделей веб-приложений и предложить соответствующие инструменты для верификации построенных моделей.
3. Предложить метод автоматической генерации тестов по построенной модели.

II раздел статьи содержит описание существующих методов проверки качества веб-приложений, их преимущества и недостатки. III раздел описывает предложенный подход к построению автоматной модели веб-приложений. В IV разделе подход проиллюстрирован на примерах существующих популярных веб-приложений.

Обзор существующих решений

В работе [4] авторы исследуют 24 различных метода тестирования и верификации веб-сайтов. В большинстве подходов предлагается начинать разработку с построения модели или строить модель вручную [5, 6, 7]. В работе [8] авторы предлагают вручную создать модель приложения и применять Model Checking для проверки свойств этой модели. Эти подходы не применимы к существующим сложным веб-приложениям, так как построение модели вручную трудоемко и не может гарантировать соответствие модели коду приложения. Более того, при изменениях кода приложения придется вручную обновлять модель.

В статье [9] описан метод построения конечного автомата, который будет описывать заданное свойство веб-приложения. Полученные модели могут состоять из тысяч состояний и не могут быть использованы как понятное описание веб-приложения. В предложенном в данной работе подходе предлагается строить модель полностью автоматически для существующего приложения и затем, вручную проанализировав модель, описывать свойства веб-приложения, для проверки которых можно будет использовать Model Checking.

Попытка автоматизировать верификацию и построение моделей предпринята в работе [10], но со значительными ограничениями: поддерживаются только приложения, разработанные с использованием фреймворка Struts и технологии Java Server Page templates, и не поддерживаются AJAX-приложения. Предложенный в данной работе подход применим к значительно более широкому кругу веб-приложений и не накладывает ограничения на используемые технологии. В том числе принципиально важна поддержка технологии AJAX, которая повсеместно используется для создания динамических и интерактивных приложений.

Данная работа описывает подход, который развивает и обладает преимуществами по сравнению с существующими подходами, в том числе поддерживает одновременно как AJAX-приложения, так и переходы между страницами по гиперссылкам. Предложенный метод построения автоматной модели позволяет автоматически строить модель всего веб-приложения без ограничений. Полученная модель позволит описывать требования спецификации ко всему приложению в целом и автоматизировать их тестирование и проверку методом Model Checking.

Автоматическое построение модели веб-приложения

Под состоянием веб-приложения будем понимать исходный код веб-страницы, на которой находится пользователь. События — действия пользователя, которые включают в себя нажатия на кнопки и прочие управляющие элементы, ввод текста и заполнение форм.

Алгоритм обнаружения возможных состояний веб-приложения

Для автоматического построения модели веб-приложения необходимо обнаружить максимальное количество возможных состояний приложения. Данная задача неразрешима в общем случае, так как различных комбинаций действий пользователя может быть бесконечно много. В данной работе предложен алгоритм обнаружения новых состояний, основанный на случайной генерации действий пользователя:

1. Исходный код страницы анализируется и составляется список всех возможных действий пользователя actionlist, на которые реагирует данная страница приложения.
2. Выбирается случайное действие из списка возможных действий actionlist и выполняется программой, которая эмулирует действия пользователя.
3. Все действия запоминаются в виде троек *<state1,action,state2>*, где state1 — состояние приложения до выполнения действия, state2 — состояние приложения после выполнения действия, action — обозначение выполненного действия.
4. Перейти к шагу 1, если за последние *N* итераций было обнаружено хотя бы одно новое событие, не встречавшееся ранее (*N* — подбирается под конкретную задачу). Данное условие останова будет срабатывать в том случае, если все состояния приложения были обнаружены или если приложение попало в некоторое тупиковое состояние, которое не позволяет перейти к новым состояниям, а только возвращаться в предыдущие.

Предложенный алгоритм не может гарантировать обнаружение всех возможных состояний и переходов между ними, но при долгом времени работы может быть получена достаточно точная модель веб-приложения. Недостаток полученной таким образом модели заключается в том, что она может состоять из сотен и тысяч различных состояний и разработчик не сможет с ней работать. Для преодоления этого недостатка и построения читаемой модели предложен метод упрощения полученной модели.

Алгоритм упрощения модели методом объединения похожих состояний

Решить проблему излишне большого количества состояний можно путем объединения похожих состояний. Под состоянием мы подразумеваем исходный HTML-код страницы. Будем работать с HTML-кодом как с DOM-деревом и введем следующее рекурсивное определение похожести: вершины DOM-дерева *A* и *B* похожи, то есть *similar(A,B)==True* если и только если они одного типа, имеют одинаковый набор атрибутов и одинаковое количество подвершин, где каждая подвершина *A* похожа с соответствующей подвершиной *B*.

При сравнении вершин текстовые значения вершин игнорируются, сравниваются только структуры DOM-деревьев. Например, вершины * text * и * othertext * будут признаны похожими. Также при сравнении двух DOM-деревьев выполняется два дополнительных шага: фильтрация вершин по их типу и сворачивание деревьев. Те вершины, которые не влияют непосредственно на поведение веб-страницы и обработку действий пользователя, отфильтровываются: link, script, meta и другие. Сворачивание дерева позволяет считать похожими страницы, которые отличаются только количеством однотипных элементов, но не отличаются своим поведением. Примером таких веб-страниц может быть список писем в почтовом ящике или список результатов поиска. Список DOM-вершин x_1, ..., x_n будет свернут, если $\forall i, j \in [1; n]$ $\text{similar}(x_i, x_j) == \text{True}$ && $x_i.\text{parent} == x_j.\text{parent}$. В этом случае список вершин будет заменен на список из одной вершины x_1. Сворачивание выполняется следующим алгоритмом:

1. Выполнять обход DOM-дерева, начиная с листьев.
2. Для данной вершины составить список подвершин list_c.
3. Сравнить все возможные пары $x_i, x_j \in \text{list}_c$ и, если $\text{similar}(x_i, x_j) == \text{True}$, удалить подвершину x_j.

Два состояния веб-приложения будут похожими, если корневые вершины соответствующих DOM-деревьев, после шагов фильтрации и сворачивания будут похожими.

Пример использования

Для апробации предложенного подхода разработан набор инструментов на языке Python 2.7. Для работы с веб-страницами и эмуляции действий пользователя используется фреймворк Selenium [11]. Для укладки графа и представления автоматной модели в виде PNG-изображения используется фреймворк GraphViz. Разработанный в рамках данного исследования инструмент для заданного веб-приложения автоматически создает модель в виде конечного автомата, который описывает обнаруженные состояния приложения и возможные переходы между ними. Метки на состояниях отражают заголовки страниц, а метки на переходах действия, которые приводят к смене состояний, записанные в формате: «нажмите на объект L» или «введите текст A в поле B». Указания на объекты внутри страницы записаны на языке XPath.

На рис. 1 приведены примеры результата работы инструмента для веб-приложений TadaList.com и m.VK.com. Для TadaList за 9 минут работы инструментом было выполнено 250 случайных действий на странице приложений, обнаружено 216 различных состояний и переходов между ними. После применения алгоритма поиска и слияния похожих состояний была получена модель, состоящая из 15 состояний. Для m.VK.com за 6 минут было выполнено 200 случайных действий, что позволило обнаружить 170 различных состояний и переходов между ними. Алгоритм поиска и слияния похожих состояний позволил упростить модель до 19 состояний.

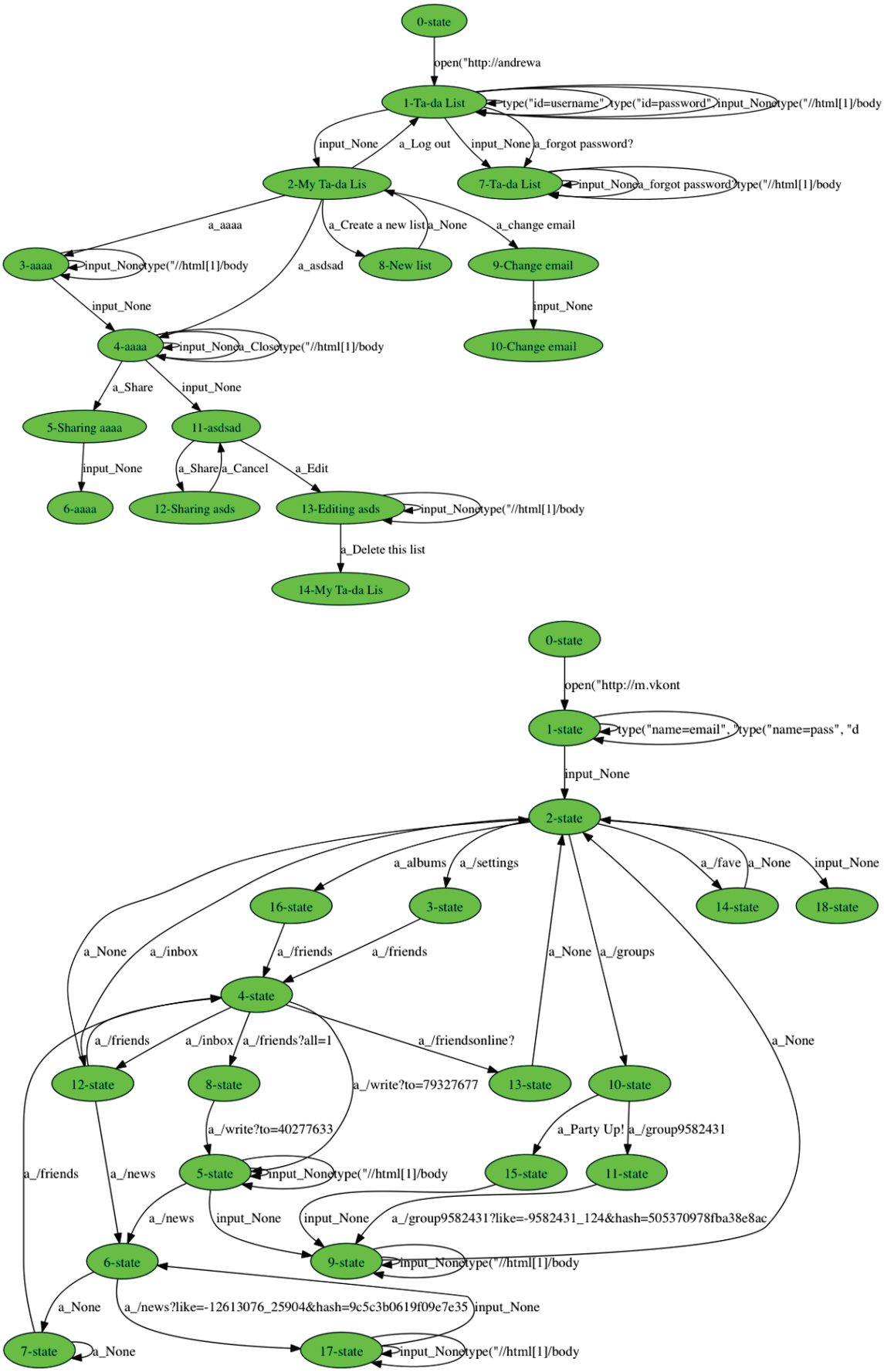

Рис. 1. Автоматически построенные модели веб-приложений

Заключение

Проверка качества веб-приложений является сложным и трудоемким процессом, так как приложения предполагают активное взаимодействие с пользователем. Вариантов различных последовательностей действий пользователя может быть неограниченно много для сложных приложений, поэтому нет возможности проверить все приложение вцелом. В работе сделана попытка автоматизировать процесс проверки качества веб-приложений.

Предложен метод автоматического построения автоматных моделей существующих веб-приложений с учетом специфики заданной области. Даже для сложных веб-приложений возможно построение наглядных и понятных моделей, благодаря разработанному алгоритму поиска и слияния похожих состояний. Модели могут быть использованы для формализации требований спецификации и автоматической проверки выполнения этих требований при помощи верификатора. При обнаружении верификатором контрпримера, этот контрпример может быть автоматически переведен в выполнимый тест, запуск которого приведет к ошибке в самом веб-приложении. Также наличие модели может существенно автоматизировать процесс создания набора тестов, который позволит проверять заданные части веб-приложения.

Литература

1. Wikipedia, Web-Application article. http://en.wikipedia.org/wiki/Web_application
2. *Шалыто А. А., Туккель Н. И.* Программирование с явным выделением состояний // Мир ПК. 2001. № 8. С. 116–121; № 9. С. 132–138.
3. Программирование мобильных устройств. http://is.ifmo.ru/science/MD–Mobile.pdf
4. *Alalfi, M. H., Cordy, J. R.,Dean, T. R.* Modelling methods for web application verification and testing: state of the art. Softw. Test., Verif. Reliab. (2009) 265–296.
5. *Hassan A. E., Holt R. C.* Architecture recovery of web applications. Proceedings of the 24th International Conference on Software Engineering ICSE, ACM Press: New York, NY, USA, 2002; 349–359.
6. *Antoniol G., Di Penta M., Zazzara M.* UnderstandingWeb Applications through Dynamic Analysis. Proceedings of the 12th International Workshop on Program Comprehension IWPC, 2004; 120–131.
7. *Di Lucca G. A., Di Penta M.* Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications. Proceedings of the Seventh IEEE International Symposium on Web Site Evolution WSE, IEEE Computer Society: Washington, DC, USA, 2005; 87–94.
8. *Sylvain Halĺe, Taylor Ettema, Chris Bunch, Tevfik Bultan*. Eliminating navigation errors in web applications via Model Checking and runtime enforcement of navigation state machines. ASE 2010: 235–244.
9. *Haydar M.* Formal Framework for Automated Analysis and Verification of Web-Based Applications. In ASE(2004) 410–413.

10. *Atsuto Kubo, Hironori Washizaki, Yoshiaki Fukazawa.* Automatic Extraction and Verification of Page Transitions in a Web Application // Apsec. 14th Asia-Pacific Software Engineering Conference. 2007. Pp. 350–357.
11. Antawan Holmes, Marc Kellogg. Automating Functional Tests Using Selenium // Proceedings of the conference on AGILE 2006. July 23–28, 2006. Pp. 270–275.