2008 15th Working Conference on Reverse Engineering

Reverse Engineering Finite State Machines from Rich Internet Applications

Domenico Amalfitano°, Anna Rita Fasolino*, Porfirio Tramontana*

mimmo.amalfitano@alice.it, anna.fasolino@unina.it, ptramont@unina.it

° Consorzio Interuniversitario Nazionale per l'Informatica, Via Cinthia, 80126 Napoli, Italy
*Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II,
Via Claudio 21, 80125 Napoli, Italy

Abstract

In the last years, Rich Internet Applications (RIAs) have emerged as a new generation of web applications offering greater usability and interactivity than traditional ones. At the same time, RIAs introduce new issues and challenges in all the web application lifecycle activities. As an example, a key problem with RIAs consists of defining suitable software models for representing them and validating Reverse Engineering techniques for obtaining these models effectively.

This paper presents a reverse engineering approach for abstracting Finite State Machines representing the client-side behaviour offered by RIAs. The approach is based on dynamic analysis of the RIA and employs clustering techniques for solving the problem of state explosion of the state machine. A case study illustrated in the paper shows the results of a preliminary experiment where the proposed process has been executed with success for reverse engineering the behaviour of an existing RIA.

1. Introduction

Software technologies, processes and paradigms for developing Internet applications are evolving in constant and rapid way, offering new methods and solutions for producing more effective Web applications. A recent output of this trend is represented by Rich Internet Applications (RIA), a new generation of Web applications which offer greater dynamicity and interactivity to their users, overcoming usability limitations of traditional web applications.

The term Ajax, originally proposed by Garrett as an acronym for Asynchronous Javascript and XML [8], now indicates an approach for developing RIAs using a combination of Web technologies that allow a browser to communicate with the server without refreshing the current page. This aspect represents the most relevant point of novelty of RIAs, and it is essentially obtained thanks to an 'Ajax engine' written in Javascript that provides new processing ability to the client side of the web application. Moreover, this engine is able to communicate with the server on the user's behalf using the XMLHttpRequest (XHR) object that allows asynchronous retrieval of arbitrary data from the server, and has produced an important shift in the Internet's default request/response paradigm.

With respect to traditional web applications, RIAs not only differ for the enabling technologies, but also for several aspects regarding the web application lifecycle. According to [11, 12], the same development process of a RIA needs to be changed in order to take into account the new RIAs' capabilities, and traditional methods, models and techniques used for designing a web application need to be adapted or extended in order to cope with the new aspects. As an example, most approaches proposed in the literature to model a traditional web application are unsuitable to specify the behaviour of an application whose processing is no more exclusively performed by the server, but can be performed on either the client, or the server, or both sides of the application. Moreover, being a RIA an hybrid between a web application and a desktop application, new architectural styles for RIAs, such as the SPIAR model presented in [10], need to be proposed and validated.

In the context of software maintenance and testing processes, a key challenge consists of finding suitable models for representing the behaviour of a RIA, and of defining effective reverse engineering techniques for obtaining them from the code of an existing application. Suitable models for representing the RIAs' behaviour include the ones usually adopted for modelling GUIs evolution in event-driven software, such as the event-flow graphs proposed in [14, 3], or Finite State Machines (FSM). FSMs provide a convenient way to model software behaviour from a black-box perspective, and several techniques have been proposed in the literature to reverse engineering them from existing software applications [4, 13, 15] and from traditional web applications [1, 5]. However, none of these techniques has been yet applied for reverse engineering FSMs from RIAs.

In this paper, an approach for reverse engineering Finite State Machines representing the behaviour of Ajax-based RIAs will be presented. The approach requires the RIA's execution traces analysis, and uses a clustering technique (based on RIA's client interface equivalence

1095-1350/08 $25.00 © 2008 IEEE
DOI 10.1109/WCRE.2008.17

criteria) that allows the fundamental states of the RIA behaviour to be determined. The proposed technique is implemented by a software tool that provides an integrated environment for performing the reverse engineering process. The remainder of the paper is organized as it follows. Section 2 describes the event-driven behaviour of a RIA, and Section 3 presents the reverse engineering process designed for obtaining the Finite State Machines. In Section 4 the tool developed to support the reverse engineering process is illustrated, while Section 5 presents a case study of reverse engineering a RIA implemented in Ajax using the proposed process and tool. Section 6 finally provides concluding remarks and future work.

2. Modelling the Behaviour of a Rich Internet Application

The most evident difference between a Rich Internet Application and a traditional web application regards their presentation levels. More precisely, at the presentation level a traditional web application can be considered as a form-based software system [2] that uses a multi-page interface model where the user submits some input on the current web page, the server executes some data processing and responds by presenting a new page. Vice-versa, the interface model of a RIA can be considered as a single-page model where changes can be made to page components without the need of refreshing the page entirely. Changes are due to elaborations performed by *event handlers* that can be triggered by several types of events. In Ajax-based RIAs, the client interface rendered by the browser is dynamically up-to-dated by Javascript event handlers (belonging to the Ajax Engine) that access and manipulate the client page using the Document Object Model (DOM) interface. The DOM is a standard application programming interface (API) for HTML and XML documents that defines the logical structure of documents and the way a document can be accessed and manipulated [6]. Using the DOM API, a web page can be represented by a hierarchical data structure, made up of components (e.g., *element nodes*) that can be dynamically changed, added, or eliminated, and that the browser renders on the client interface.

At run-time, the client-side visible behaviour of a RIA is reflected by the event-driven evolution of the document object model of the RIA's web page. This evolution can be described using the information modelled by the UML class diagram reported in Figure 1. In the model, each Client Interface is associated with the corresponding *DOM* configuration. A DOM is composed of *DOM elements,* a *DOM element* can be associated with 0 or more *Event Listeners* registered with the element, and this pair is associated with the corresponding *Event Handler*. There are three types of *event listeners*, namely *user event*, *time event* (due to the occurrence of timeout conditions), and *http response event* listeners (due to receptions of responses to some http request, such as a request for a web page, or an asynchronous Ajax (XHR) request). An *Event Handler* may be either declared in the script code explicitly, or may be implicitly pre-defined (such as for the default event handlers of click events on hyperlink objects, or on form submit buttons). During the RIA execution, the occurrence of each Event at a given *Timestamp* will be registered by a *Raised Event*, and the processing triggered by the Raised Event may result in a *Transition* that links the starting Client Interface to an ending one reached at the *End Transition Timestamp*. Moreover, the handling of a raised event may instantiate 0 or more *HTTP Requests* (which may either be *Web Page Requests* to *Server Pages*, or *XHR Requests* directed to any *Server Side Resource)* or 0 or more *Time Event listeners*.

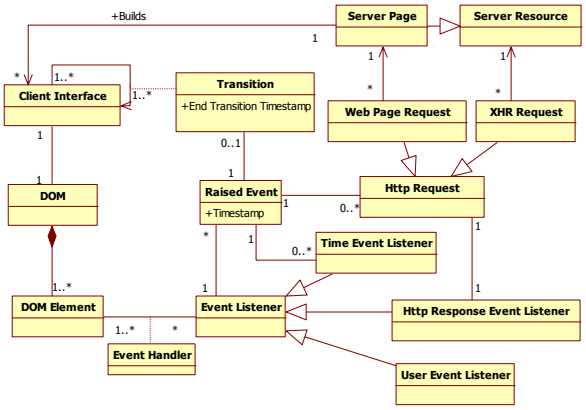

Figure 1: The conceptual model of a RIA client-side behaviour

This class diagram not only represents the client-side evolution of pure single-page Rich Internet Applications, but also of traditional "Web 1.0" applications. Indeed, the model depicts also typical aspects of a traditional web application, such as hyperlinks and forms (that are specific types of DOM elements) and related events (such as click on hyperlinks or submit buttons). Moreover, this model represents both synchronous Web Page Requests (that are typically made by a "Web 1.0" application) and asynchronous XHR requests (typical of RIAs) that are explicitly modelled as two particular cases of Http requests. In the following Section, the use of this information model for reverse engineering a Finite State Machine representing the behaviour of a RIA will be shown.

3. Reverse Engineering Process

The Reverse Engineering process proposed in this Section aims at reconstructing a Finite State Machine modelling the behaviour of an existing Rich Internet

Application using a combination of dynamic analysis and clustering techniques. The process is based on two sequential activities of *Extraction* and *Abstraction*, respectively.

3.1 Extraction

The Extraction activity is actually a user sessions tracing activity where the RIA dynamic analysis is executed in a controlled environment in order to trace sequences of events (making up the execution of specific use cases of the application) and to register corresponding results available on the client side of the application.

This tracing activity can be modelled by the statechart diagram shown in Figure 2 that includes two main iterative states, the *Event Waiting* and the *Event Handling Completion Waiting* one. When the tracing activity starts (*Start Tracing* state), a new Web page is loaded and rendered by the browser, and the *Event Waiting* state is entered where the page remains frozen until any event rises. Entering this state, the DOM Extraction activity- consisting of extracting and storing information about the structure of the currently rendered DOM - is carried out.

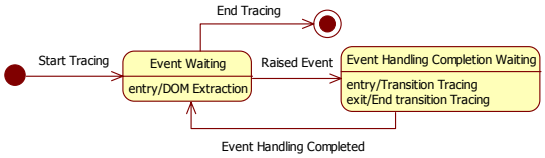

Figure 2: The Tracing Activity of the Extraction Step

The occurrence of a raised event causes a transition from the *Event Waiting* state to the *Event Handling Completion Waiting* state.

Entering the *Event Handling Completion Waiting* state, a *Transition Tracing* activity is carried out consisting of extracting and storing information about the raised event, such as its type and timestamp of raising, and the eventual DOM element node which it has been raised on. When the event handling is completed, the time of the event handling termination, the eventual HTTP requests or time event listeners that have been instantiated are captured and stored (by the End Transition Tracing activity) and the process returns to the *Event Waiting* state. While in the Event Waiting state, the *Tracing activity* can be stopped for exiting from the process Extraction step.

3.2 Abstraction

In the *Abstraction* activity, an FSM modelling the RIA behaviour is abstracted, starting from data collected by dynamic analysis. This activity is accomplished in three steps, namely *Transition Graph generation*, *Clustering*, and *Concept Assignment*.

In the first step, the RIA *Transition Graph* (TG) is built: this graph models the flow of RIA client interfaces that were generated during dynamic analysis. TG nodes represent the RIA traced client interfaces, while TG edges represent events that triggered the generation of new interfaces.

In the *Clustering* step, the Transition Graph is analysed and a Clustering technique is used for grouping together its equivalent nodes and edges. The clustering technique is based on the evaluation of several *interface equivalence criteria* that define the required properties of two client interfaces (or two transitions between interfaces) in order to consider them equivalent. As an example, a first possible equivalence criterion (c1) considers equivalent two RIA's interfaces if the corresponding DOM structures include the same set of 'active element nodes' (that is elements with registered event listeners) and offer the same interaction behaviour to their users (by means of a same set of event handlers). A further equivalence criterion (c2) may consider two client interfaces to be equivalent if they include the same set of 'active' DOM elements and have the same set of Http requests, or time event listeners that have been instantiated when the RIA showed that specific client interface.

Given an interface equivalence criterion, two TG nodes will be considered equivalent and clustered together if the corresponding RIA client interfaces satisfy the equivalence criterion. Moreover, two TG edges between equivalent nodes will be considered equivalent only if they are associated with the same type of event, that is raised on the same DOM element, and is managed by the same event handler.

The resulting collapsed TG is submitted to the *Concept assignment step*, which will finally generate the FSM. During this step, each node of the TG will be initially assumed as a distinct state of the State Machine, and the software engineer knowledge and experience will be needed for validating or discarding this hypothesis. The transitions between states will be deduced accordingly.

4. The Reverse Engineering Tool

The execution of the proposed reverse engineering process is supported by a tool (named RE-RIA tool) that provides an integrated environment for performing the process activities. The tool has been implemented with Java-based technologies, and its architecture (shown in Figure 3) is composed of GUI, Extractor and Abstractor packages, and a relational database that stores the extracted information and produced abstractions.

The GUI package includes a *Browser* and a *Reverse Engineering Process Manager* component. The *Browser* is actually the instantiation of a Mozilla Firefox Browser inside a Java GUI, allowing RIAs to be navigated through the GUI, while their structure and behaviour can be at the same time accessed by other components of the tool. The *Reverse Engineering Process Manager* component

provides the user with several functionalities for the reverse engineering process configuration and management.

The Extractor package comprises two components, namely the *DOM Extractor* and the *Trace Extractor* ones. The *DOM Extractor* is a Java component interacting with the Browser in order to extract and store information about instantiated DOM element nodes that are rendered by the browser. The *Trace Extractor* component interacts with the Browser in order to trace and collect information about event raising and related processing, besides the termination of the execution of their event handlers. The information collection has been implemented by inserting non-invasive observers and probes. As an example, for tracing user events the *bubbling* and *capturing* standard mechanisms defined by the W3C for DOM event dispatching [9] have been exploited.

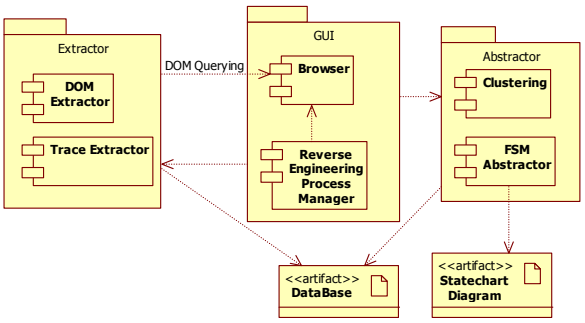

Figure 3: The RE-RIA tool architecture

The Abstractor package includes two components, the *Clustering* and the *FSM Abstractor* component, respectively. The *Clustering* component is a Java component that evaluates the equivalence criteria for each couple of nodes of the *Transition Graph* and finally groups together Transition Graph's equivalent nodes and edges. The resulting collapsed Transition Graph is stored in the database. The *FSM abstractor* component supports the Concept Assignment step during which Transition Graph nodes and edges are analysed in order to abstract states and transitions of the Finite State Machine. In particular, this component offers a GUI where a user can make hypotheses about the FSM, validate them, and store validated hypotheses in the tool database.

5. Case Study

This Section presents a case study that shows how the reverse engineering approach proposed in the paper has been successfully used for abstracting a Finite State Machine modelling the behaviour of a medium-size open source RIA. The subject of the experiment was an Ajax-based RIA named *FilmDB* [7] that provides registered users with several functionalities for data management of a personal movie archive. The server side of this application is implemented by 99 PHP server pages (624 kBytes) that generate client pages containing several scripts (coded in Javascript) implementing a complex user interface. Moreover, *FilmDB* interacts with other server side resources by exploiting Ajax (XHR) requests.

In the experiment, several use cases offered by the application were submitted to the reverse engineering process. In the following, we report data about the reverse engineering of two specific use cases, the former allowing a user to enter his/her personal movie area, and the latter for exiting this area. Five user sessions were used for exercising all the alternative scenarios of these use cases.

The process was performed by one of the process authors (who had just a user experience with the RIA) with the support of the RE-RIA tool. In the *Extraction step*, the Trace Extractor component captured and stored various data about traced sessions. Table 1 reports summary session data captured by the tool. The Transition Graph associated with these execution traces was quite complex, and included 60 nodes and 59 edges.

Table 1- Summary data about traced sessions

#Client interfaces	60
# Extracted DOM elements	6015
# Traced transitions	59
# Total traced User events (of which)	42
# Click on a DOM element	11
# Mouseover on a DOM element	10
# Mouseout on a DOM element	7
# Keydown	14
# XHR response reception events	8
#Client interface reception events	4
# Timeout events	5

In the *Abstraction step*, the clustering technique was executed for simplifying the Transition Graph. Both the equivalence criteria (c1) and (c2) (illustrated in Section 3.2) were used by the software engineer in order to compare produced results. Both the resulting collapsed versions of the TG were submitted to the *Concept Assignment* activity. In particular, the TG produced by the (c1) criterion comprised 8 nodes and 22 transitions (with respect to the 60 nodes and 59 edges of the initial graph). The *Concept Assignment* activity revealed that most TG nodes could be associated with single states of the FSA, while some nodes could not be associated with meaningful states, but had to be split into further logical states. This activity lasted about one hour.

The TG produced by the (c2) criterion comprised 12 nodes and 23 transitions. The *Concept Assignment* (that lasted about 45 minutes) revealed that all nodes of this TG could be associated with meaningful states. Indeed, the graph nodes associated with RIA's client interfaces having the same DOM element nodes, but differing just for the set of instantiated XHR objects, Http Requests,

and timeout listeners were correctly considered non equivalent. Therefore, this graph reconstructed correctly the behaviour of the RIA, thanks to the correct consideration of Ajax synchronization communication mechanisms. Figure 4 shows the final FSM that was obtained. The results obtained by comparing these two different equivalence criteria confirmed us that interface equivalence criteria are able to influence the effectiveness of our technique, and thus they need to be accurately designed and experimentally validated. Figure 4 shows the final FSM we obtained.

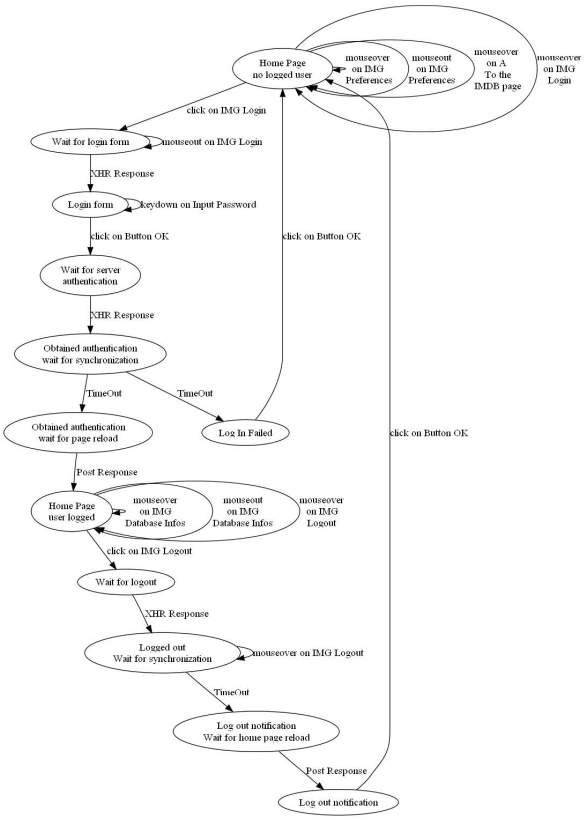

Figure 4 – The Resulting FSM from the case study

6. Conclusions

This paper presented the initial results of a research project that aims at defining and validating techniques for reverse engineering software representation models from Rich Internet Applications. The research preliminarily addressed the problem of reverse engineering a Finite State Machine modelling the client-side behaviour of a RIA and proposed a tool-supported approach for obtaining this model. In the paper we have presented this approach, that requires dynamic analysis of the application and uses a clustering technique based on client interfaces equivalent criteria in order to abstract the relevant states of the RIA's behaviour.

Some case studies we performed showed the approach feasibility, and highlighted future works to be addressed. In particular, we plan to extend the experimentation with further case studies in order to assess the scalability of the approach. Moreover, we will address the adequacy of the proposed model for supporting maintenance and testing activities involving RIAs.

References

[1] A. Andrews, J. Offutt, R. Alexander, "Testing Web applications by modeling with FSM", *Software and System Modeling*, 2005, 4 (3), 326-345

[2] D.L. Atkins, T. Ball, G. Bruns, K. Cox, "Mawl: a domain-specific language for form-based services". *IEEE Transactions on Software Engineering*, 1999, 25 (3), 334-346.

[3] F. Belli, C. J. Budnik, L. White, "Event-based modelling, analysis and testing of user interactions: approach and case study", *Software Testing Verification and Reliability*, J. Wiley & Sons, Ltd., 2006, 16, 3-32.

[4] G. Canfora, A.R. Fasolino, G. Frattolillo, P. Tramontana, "Migrating Interactive Legacy Systems To Web Services", *10th IEEE European Conference on Software Maintenance and Reengineering, CSMR 2006*, 23-32.

[5] G. Di Lorenzo, A. R. Fasolino, L. Melcarne, P. Tramontana, V. Vittorini, "Turning Web Applications into Web Services by Wrapping Techniques", *Proc. of 14th Working Conference on Reverse Engineering, WCRE 2007*, 199- 208

[6] Document Object Model (DOM), W3C, available at: http://www.w3.org/DOM/

[7] Ajax FilmDB, available at: http://sourceforge.net/projects/ajaxfilmdb/

[8] J.Garrett, "AJAX: A new approach to web applications". *Adaptive Path*, 2005.

[9] B. Hohrmann, P. Le Hègaret, T. Pixley, "Document Object Model Events", W3C, 21 December 2007, available at http://www.w3.org/TR/DOM-Level-3-Events/events.html,

[10] A. Mesbah, and A. van Deursen, "An Architectural Style for Ajax", *Proc. of the Sixth Working IEEE/IFIP Conference on Software Architecture(WICSA '07)*, 9-18

[11] J.C. Preciado, M. Linaje , F. Sanchez, S. Comai, "Necessity of methodologies to model Rich Internet Applications", *Proc. of the Seventh IEEE international Symposium on Web Site Evolution (WSE 2005)*,7-13.

[12] J.C. Preciado, M. Linaje, S. Comai, F. Sanchez, "Designing Rich Internet Applications with Web Engineering Methodologies", *Proc. of 9th IEEE Int. Symposium on Web Site Evolution, (WSE 2007)*, 23- 30.

[13] E. Stroulia, M. El-Ramly, P. Sorenson, "From Legacy to Web through Interaction Modeling", *Proc. of IEEE Int. Conference on Software Maintenance, ICSM 2002*, 320-329

[14] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for GUI-based software applications". *ACM Trans. Softw. Eng. Methodol.*,2007, 16(1), 4-39.

[15] N. Walkinshaw, K. Bogdanov, M. Holcombe, S. Salahuddin, "Reverse Engineering State Machines by Interactive Grammar Inference", *Proc. of 14th Working Conference on Reverse Engineering, WCRE 2007*, 209-218.