Validation of a Dynamic Analysis Based Technique for Reverse Engineering of Rich Internet Applications

Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana

domenico.amalfitano@unina.it, anna.fasolino@unina.it, ptramont@unina.it

Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II,
Via Claudio 21, 80125 Napoli, Italy

Abstract

Dynamic analysis techniques provide a suitable approach for exploring and comprehending the behaviour of Rich Internet Applications (RIAs), a new generation of Web applications with enhanced interactivity, responsiveness and dynamicity. However, for addressing their intrinsic scalability problems, dynamic analysis techniques require ad-hoc solutions such as effective classification techniques for identifying equivalent behaviours exhibited by the application in several user sessions.

This paper relies on a dynamic analysis based reverse engineering technique proposed by the authors to reconstruct a model of the RIA behaviour based on Finite State Machines. This technique requires the analysis of the RIA user interface evolution shown in user sessions, and exploits user interface equivalence criteria for abstracting relevant states and state transitions to be included in the model. In this paper, the results of a validation experiment we carried out for assessing the effectiveness and the cost of this technique are presented. The experiment involved four distinct RIAs implemented with AJAX technique and its results showed strengths and weakness of the proposed technique, and demonstrated its effectiveness.

1. Introduction

Rich Internet Applications (RIAs) are a new generation of Web applications which exploit specific Web technologies for overcoming the usability limitations of traditional Web applications and offering greater interactivity, responsiveness and dynamicity to their users.

The new features of RIAs are basically due to the fundamental shift from the multi-page communication model between client and server, that is typical of traditional Web applications, to the single-page model that characterizes RIAs. In the multi-page model the user experience is limited by the need of continuous requests of Web pages from the client to the server, which delay the user fruition of the application. Vice versa, in an RIA the single-page user interface does not limit itself to request pages, but it is able to perform computations autonomously, send and retrieve data in the background asynchronously from the user's requests, redraw sections of a screen, and so forth, independently on the server or back end it is connected to. Consequently, a Rich Internet Application can be considered similar to a *desktop application*, but with the advantage of being accessible via Internet. See Google Maps, GMail, or Flickr as well known examples of RIAs.

This client side processing ability of RIAs is essentially due to a *client engine*, i.e., a client side component which operates between the browser and the Web server, being responsible both of rendering the user interface and of communicating with the server. As an example, in the case of RIAs implemented with *AJAX* technique [Gar05], the client engine is a *Javascript* engine that performs the elaborations associated with Javascript event handlers which can be fired by user events or other external events. Event handlers can be associated with client page widgets, and access and modify the client page using the *Document Object Model* (DOM) interface [DOM], a standard API for HTML and XML documents that defines the logical structure of documents and the way a document can be accessed and manipulated. In addition, Javascript elaborations can also produce *asynchronous* requests of portions of data to the server, or trigger server side elaborations.

While it yields a more interactive user experience, the RIA user interface model raises a number of new issues such as:

- Searchability: the dynamic generation of the client interface content disables the possibility of indexing an RIA by traditional approaches of search engines, which assume that each state in a Web application corresponds to a page and a distinct URL.
- Accessibility: accessing the full content of a Web page by assistive technologies, such as

screen readers, may worsen the user experience of an RIA, since they make an intensive use of dynamic DOM modifications by scripting.

- Analyzability: analyzing the functional characteristics of an RIA user interface is not as simple as in the case of traditional Web applications, because of event-driven dynamic generation of code, which make the application behavior less predictable.
- Testability: the possibility of having several units of work present in a single client page, which may carry out distinct elaborations on the DOM or server requests concurrently, is able to create a complex scenario where several types of new defects arise, such as incorrect manipulations of the DOM, unintended interleaving of server messages, swapped callbacks, etc. [Mar08], which require new and specific testing techniques.

A common and relevant problem when managing existing RIAs consists of comprehending their behaviour and developing a suitable model for representing it. Indeed, the event-driven behaviour of any RIA is less linear and predictable than the one of a traditional Web application one, and static analysis techniques cannot be sufficient for understanding it. Vice versa dynamic analysis, and in particular user session based dynamic analysis, is a technique that helps in discovering the behaviour of an existing application, using data about user session executions. However it suffers scalability problems because it requires that a wide set of user sessions be traced in order to exercise all possible application behaviours, and it requires effective analysis techniques for classifying equivalent execution traces, i.e., traces associated with the same behaviour.

Several approaches have been proposed in the literature for solving this classification problem, including concept analysis techniques [Sam04], sequence alignment algorithms [Mai08], clone detection techniques [Dil02]. However, the most of these techniques have been experimented with object-oriented software or traditional Web applications, and none of them has been already used in the context of Rich Internet Application. Hence, further investigation is needed for obtaining effective reverse engineering techniques for modelling an RIA behaviour, and based on dynamic analysis.

In [Ama08], the authors proposed of using Finite State Machines (FSMs) for representing the behaviour of AJAX applications, and presented a reverse engineering technique and a tool for obtaining them from existing applications using dynamic analysis.

In this paper, we present the results of an experiment that aimed at assessing the effectiveness of this technique in reconstructing a model of the RIA behaviour that can be used for maintenance, evolution, or re-documentation purposes. The experiment involved four distinct Web applications implemented in AJAX, and was carried out for showing effectiveness and costs of the proposed technique and tool, and exploring its main strengths and weaknesses.

The paper is organized as it follows: Section 2 describes the characteristics of the FSM model we adopt for representing the behaviour of an existing Rich Internet Application. In Section 3 the FSM reverse engineering technique based on dynamic analysis of the RIA and the tool supporting its execution will be presented. Section 4 reports the experiments that have been carried out, while in Section 5 conclusive remarks and future work will be described.

2. A behavioural model for an RIA

Since an RIA can be considered to be a hybrid between a Web application and a desktop application, its behaviour may be represented by models that are usually adopted for event-driven software or GUIs, such as Event-Flow graphs [Bel06] or State Machines [Bin99], [Mem03].

Finite State Machines, which have also been used with success for modelling traditional Web applications [And05], provide an abstract view of a system in terms of states and transitions among them.

A FSM representing an RIA behaviour will be a triple (S, T, E) where S is a set of states reached by the RIA during its processing, T is the set of transitions between states, and E is the set of events that cause state transitions.

For developing a FSM, since the number of possible RIA states may be unbounded, we must use a *state abstraction criterion* for deciding which states of the RIA evolution will be represented in the model. Moreover, each RIA state must be characterized by a sub-set of its features and then, a *state representation criterion* will be needed for establishing which the relevant characteristics of each state are.

We propose of representing in the FSM all the elaboration states where the RIA receives any input solicitation by its user (state abstraction criterion), and of describing each state of the RIA by the User Interface shown to the user at that interaction time (representation criterion). Moreover, the proposed state characterization criterion requires that each *client interface* is characterized only by the sub-set of its *widgets* that are 'clickable' or, more in general, that

have a *registered event listener* and a corresponding *event handler*. Finally, *transitions* will be associated with user interactions (e.g., user events) that triggered the RIA migration towards the new state. The proposed FSM model of an RIA can be characterized by the information shown by the UML class diagram of Figure 1.

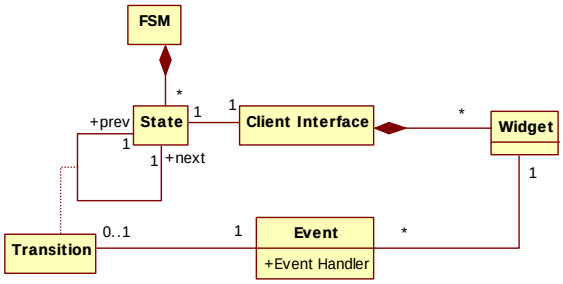

Figure 1: Conceptual model of the FSM

The example of FSM reported in Figure 2 represents the behaviour of an existing application that offers facilities for managing a list of data items. The model includes three states with labels 'view item list' (the starting state), 'edit item', and 'add new item', respectively, and seven transitions which move the RIA through the states.

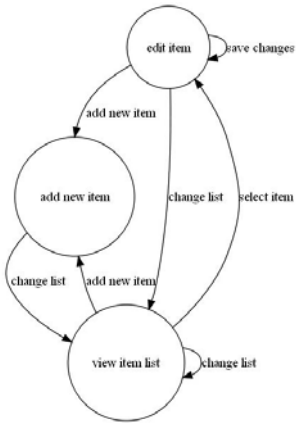

Figure 2: An example of FSM representing an RIA behaviour

3. The reverse engineering technique

The proposed technique of dynamic analysis for obtaining a FSM-based model of the RIA behaviour includes four sequential steps:

1) Automatic RIA instrumentation
2) Execution trace collection;
3) Trace analysis and classification;
4) FSM model abstraction and validation..

3.1 Automatic RIA instrumentation

Dynamic analysis requires that the subject application is preliminarily instrumented in order to record relevant information about its run-time. Information needed for reconstructing the proposed FSM model of an RIA refer both to *states* of the client interfaces captured before a new user event is fired on them, and to related raised *user events.*

Since in Rich Internet Applications the state of each user interface is provided by a specific configuration of the DOM model of the Web page, client interface states can be captured by accessing the DOM. To this aim, we have used an automatic and non-invasive technique that does not instrument the code of the application directly, but rather the browser that renders it. Further details about this technique are reported in [Ama08].

.

3.2 Execution trace collection

The aim of this activity consists of recording a set of execution traces of the RIA from user sessions. An execution trace can be modelled as the following sequence of pairs generated during the user session:

$\ldots\langle I_i, event_i\rangle, \langle I_{i+1}, event_{i+1}\rangle,..$

where, in each pair $\langle I_k, event_k\rangle$, I_k represents a user interface state and $event_k$ is the user event occurred on that interface during the execution.

In order to obtain an adequate model including all the application behaviours, execution traces being collected should exercise the RIA in all possible ways and conditions. Two main approaches are usable for collecting an adequate set of execution traces. The former approach is based on the knowledge of the expected behaviours of the RIA, and uses such knowledge to pilot user sessions in order to exercise all behaviours. Vice versa, in the latter approach (that can be considered an explorative approach) this knowledge is not available, and the only feasible strategy consists of collecting execution traces from unconstrained user sessions (such as real user sessions) until it can be assumed that traces cover all possible behaviours. A termination criterion is in this case needed to stop user session collection.

3.3 Trace analysis and classification

A relevant problem of dynamic analysis techniques consists of detecting and filtering out redundant information contained in the set of collected execution traces. More precisely, a given set of user session execution traces will usually include more executions of the same behaviours and, correspondently, more instances of the same user interfaces.

Each group of logically equivalent interfaces defines an equivalence class, and if equivalent user interfaces are detected in the set of execution traces, it is possible to substitute them with the corresponding equivalence class, thus simplifying the information contained in the execution traces.

Our technique solves this classification problem on the basis of *interface structural equivalence criteria* that define the required properties of two client interfaces in order to consider them equivalent: in particular, two client interfaces are considered equivalent if they include exactly the same set of 'active widgets' (that is, elements with registered event listeners), offering the same interaction behaviour to their users (by means of a same set of event handlers), and having the same values of some additional properties (such as its absolute indexed *path*, or unindexed *path*[1] in the page DOM, its visibility property, and so on).

In particular, since today's RIA may have different characteristics, we have proposed three different *interface structural equivalence criteria*, C1, C2, and C3, which evaluates the equivalence of two interfaces on the basis of a different set of active widgets properties. The definition of these criteria is reported in the following Table 1.

3.4 FSM model abstraction and validation

When the trace collection activity ends, the FSM abstraction step can be entered: the execution traces will be analysed and a machine modelling the behaviour of the analysed application will be defined. The resulting FSM=(S, T, E) will include a set S of states corresponding to all interface equivalence classes discovered by a considered equivalence criterion, while the set T of transitions will be defined on the basis of recorded transitions between consecutively visited client interfaces. The set E will include all events that were registered on client interfaces. Of course, the set of registered events may be divided into equivalence classes too, by suitable classification criteria. However, at the moment we have not used any event classification technique, and we decided to report all registered events in the final FSM model.

Table 1: Client Interface Equivalence Criteria

Criterion	Description
C1	Two client interfaces I1 and I2 are equivalent if the same active widgets of I1 are also included in I2 and vice versa, and they have the same indexed path, the same type of corresponding listeners, and the corresponding event handlers have the same name.
C2	Two client interfaces I1 and I2 are equivalent if the same active widgets of I1 that are visible and enabled are also included in I2 and vice versa, and they have the same indexed path, the same type of corresponding listeners, and the corresponding event handlers have the same name
C3	Two client interfaces I1 and I2 are equivalent if the same active widgets of I1 that are visible and enabled are also included in I2 and vice versa, and they have the same unindexed path.

The model validation activity is required for assessing the correctness/adequacy of the reconstructed FSM, and for assigning each validated state with a meaningful description. Generally, the correctness of such a model depends on the objectives of the task the model was produced for (such as comprehension, testing, maintenance of the application, etc.), and its evaluation will be based on the judgment of an expert on the specific field.

3.5 Tool support

The proposed Reverse Engineering technique can be executed with the support of the RE-RIA (Reverse Engineering RIA) tool that provides an integrated user-friendly environment where execution traces collection, traces analysis and classification, and FSM abstraction and validation activities can be performed. The tool was developed with Java-based technologies, and its GUI offers a Web Browser (an instantiation of a Mozilla Firefox Browser inside a Java GUI) where user sessions relevant data can be captured and recorded. Moreover, the tool implements the trace analysis and classification techniques, and it automatically abstracts the corresponding FSM model from a given set of execution traces. In addition, the tool provides some facilities for supporting the validation activity of a FSM made by an expert of RIAs, and performs several measurements about the executed reverse engineering activities that were

[1] An example of indexed path of a widget is:
/HTML/BODY/TABLE[2]/TR[3]/TD[4]
while the corresponding unindexed path will be:
/HTML/BODY/TABLE/TR/TD

needed to carry out the experimental study that we present in the following Section.

4. The experiment

This Section illustrates an experiment that was carried out using real Rich Internet applications. In particular, the experiment was designed for answering the following research questions:

(RQ1) What effect do the interface equivalence criteria have on the effectiveness of the reverse engineering technique in reconstructing a behavioural model of an RIA?

(RQ2) What combination of technique factors provides the best cost-effectiveness ratio?

4.1 Modelling Effectiveness and Cost of the technique

For evaluating the effectiveness of the technique we analyse the quality of its output and, in particular, the *correctness* of the FSM model produced by it. For FSM correctness evaluation, we consider an expert-based evaluation approach which requires the involvement of an expert in the specific field and task for which the model was required. As an example, if the model was produced with the aim of supporting the comprehension of the application behaviour during maintenance, the model can be considered to be correct if it describes its behaviour *correctly,* according to the opinion of an expert maintainer of RIAs.

The FSM correctness evaluation problem can be modelled as the problem of comparing the FSM model *M* produced by the technique by analysing a given set of execution traces *T* (including a set *I* of visited user interfaces of the RIA) against the FSM model *O* which would have been produced by the expert by analysing the same set of execution traces *T*.

Since both the expert and the technique actually distribute the set of visited interfaces I into a set of partitions (e.g., the states of the respective FSM models), the comparison of the models M and O corresponds to the evaluation of the distance between their partitions. This problem is illustrated by the example in Figure 3 that shows two partitions M, and O of a same set of interfaces I, where M={C1, C2, C3} represents the set of partitions of a reconstructed FSM model, and O={O1, O2, O3, O4} represents the expert's model partitions. The difference between these models may be expressed by the minimum number of interface move operations between partitions that are needed for transforming the set of partitions M into the set O. In the example, d(M,O)= 3, since three interfaces, named I_3, I_4 and I_8, need to be moved. This number of operations represents an *edit distance* [Alm99], and can be evaluated using well known partition distance computation algorithms. Here, we adopt the following efficient algorithm proposed in [Kon05] for computing partition edit distance.

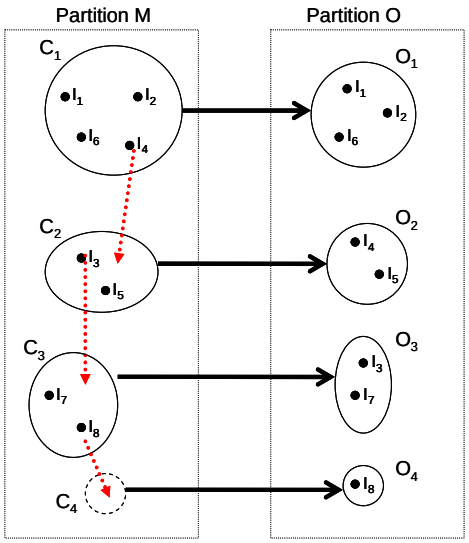

Figure 3: An example of partition comparison

Partition distance Evaluation. Given a set *I* of user interfaces I_i, and given two partitions of *I*, M1 and M2, respectively, we have:

$M_1 = \{C_1, \ldots, C_a\} = \{ \{I_1, \ldots, I_{c1}\}, \ldots, \{I_x, \ldots, I_{ca}\} \}$,
such that $\cup_{i=1..a} C_i = I$
$M_2 = \{O_1, \ldots, O_b\} = \{ \{I_1, \ldots, I_{o1}\}, \ldots, \{I_y, \ldots, I_{ob}\} \}$
such that $\cup_{j=1..b} O_j = I$

The algorithm computes the square difference matrix Δ, whose generic element, $\Delta_{ij} = |C_i - O_j|$ is the cardinality of the difference set between clusters C_i and O_j from partitions M_1 and M_2, respectively [2].

$$\Delta = \{\Delta_{ij}\} = \{|C_i - O_j|\} \qquad \forall\, i,j \in [1, n]$$

Hence, the algorithm of [Kon05] computes partition distance by iteratively selecting the pair of partitions C_x, O_y such that Δ_{xy} has the minimum value in Δ, and

[2] If a≠ b, the algorithm requires |a- b| empty clusters to be added to the partition having the smaller cardinality. n=max(a,b) will be the final dimension of Δ square matrix.

by reducing the Δ matrix by deleting the x-th row and y-th column, and by updating the distance d by the relative distance $\Delta_{xy.}$ The algorithm is illustrated in Figure 4:

```
int d=0   //initializes the distance d
FOR k=1 to n
  Find (x,y): Δxy=minij (Δij)∀ i,j ∈ [1, n]
  d=d+Δxy  // increments the distance d
  Delete row x from Δ
  Delete column y from Δ
END FOR
```

Figure 4: Partition distance computation algorithm

Once the partition distance *d (M, O)* between the reconstructed model M and the expert's model O has been computed, it is possible to measure the effectiveness of the reverse engineering technique by the following *Correct Interface Ratio (CIR)* metric:

CIR (M) =
= # of correctly partitioned interfaces of M / Cardinality (I) =
= 1- d(M, O) / |I|

where I is the set of interfaces included in the set of analysed execution traces and |I| is its cardinality.. Of course, the best effectiveness values correspond to CIR=100%, representing the case of two identical partitions.

For modelling the cost of the proposed reverse engineering technique, we analysed the cost of its single activities, by distinguishing between manual, automatic, or semi-automatic activities, and by defining an approximate cost evaluation metric for each of them. Results of this analysis are reported in Table 2.

Intuitively, C_{coll} , C_{abstr} and C_a depend on the number of analysed trace interfaces, and grow with it. $C_{analysis}$ depends both on the number of analysed trace interfaces, and on the number of active widgets included in analysed interfaces, while C_{mov} grows with the number of interface move operations needed for correcting the reconstructed model, that is with the partition distance *d(M, O)*.

If we consider negligible all the costs of the automatic activities, the most relevant cost factors include C_{coll} and $C_{mov.}$. Therefore, C will depend both on the size of analysed trace interfaces, and on the number of active widgets included in analysed interfaces.

4.2 Experimental Procedure

Four distinct Web applications were selected and, for each application, the following steps were performed:

1) User Session Trace collection;
2) Gold Standard (GS) FSM model production;
3) FSM model abstraction;
4) Effectiveness evaluation.

Details about these steps are discussed in subsequent sections.

Table 2: Activity Cost analysis

Activity	Manual or Automatic	Activity Cost description
1. Trace Collection	Semi-automatic	It is the cost C_{coll} of collecting user session traces (with the tool support)
2. Trace Analysis and Classification	Automatic	It is the cost $C_{analysis}$ needed for classifying analysed interfaces into a set of equivalence classes, on the basis of the equivalence criterion C
3. FSM abstraction	Automatic	It is the cost C_{abstr} of defining the FSM on the basis of recovered interface equivalence classes
4. FSM Validation	Manual	It includes the cost (C_a) for analysing all trace interfaces and the cost (C_{mov}) of moving interfaces between partitions for correcting the reconstructed model

4.2.1 Experimental Materials. We involved in the experiments two software engineers who were expert in developing and maintaining Rich Internet Applications developed in AJAX, and in finite state machine matters, but not in the subject Web applications, and five under-graduate students from the Software Engineering courses held at the University of Naples, in Italy.

Subject applications were four real AJAX applications available online, with a rich user interface, which offered most of their use cases in a single-page interface [Mes07b]. The first application W1 (Tudu) is an open source application offering 'todo' list management facilities (such as adding, deleting, searching for todos, organizing lists of todos, and so

on). The second one, W2 (Pikipimp), is a free application allowing a user to upload his photos and add some graphical effects. The third application W3 (TheList) is a demo application providing functionalities to manage a list of task descriptions. The fourth one, W4 (Buttonator) is a simple utility for Web developers that offers functionalities for generating buttons with different shapes, size, and colours. Table 3 summarizes the main characteristics of these applications, including their URLs, and count of considered use cases and relative scenarios.

Table 3: Subject applications

Subject Applic.	URL	Use Cases	Scenarios
W1	http://app.ess.ch/tudu /welcome.action	8	17
W2	http://www.pikipimp.com	1	2
W3	http://www.agavegroup.com /agWork/theList/ theListWrapper.php	3	10
W4	http://www.buttonator.com	1	8

4.2.2 Step (1): Trace Collection. A set of two/ three students per application were trained about the application use cases (and their normal and alternative scenarios), and were asked for collecting a set of user session traces. We required each student to cover each use case of the application at least two times with their user sessions.

This task was accomplished with the support of the RE-RIA tool and returned a collection of execution traces ET per application. Characteristics of collected traces for each application are reported in Table 4.

Table 4: Collected Traces for subject applications

Subject Application	Collected User Session Traces	Collected Interfaces
W1	30	1885
W2	8	533
W3	11	731
W4	11	829

4.2.3 Step (2): Gold Standard production. The experts (who worked in group) produced a FSM reference model of the behaviour for each Web application, the so called 'Gold Standard' (GS) model, to be used for comparative analysis. Each GS model was obtained by analysing the collected execution traces ET (with the support of RE-RIA tool) and the component user interfaces, and producing a GS= (S, T) that had to satisfy the following requirements: (1) *each state of the model had to be associated with a relevant state of the application User Interface from which a user action could be executed,* (2) *a transition between states had to link pairs of consecutively visited user interfaces*, (3) *each transition had to be labelled by the user actions that triggered the transition.*

Each GS model provided a specific partitioning of execution trace interfaces. Characteristics of obtained GS models for each application are reported in Table 5.

Table 5: Gold Standard models characteristics

Subject Application	GS states	GS transitions
W1	15	52
W2	4	16
W3	4	9
W4	19	54

4.2.4 Step (3): FSM abstraction. Using the proposed reverse engineering technique and the same set of execution traces ET for each application, three FSM models M1, M2, and M3 were obtained per application and for each analysed execution trace, each one on the basis of a different interface equivalence criterion (C1, C2, and C3, respectively). Each model provided a different partitioning of execution trace interfaces.

4.2.5 Step (4): Effectiveness and Cost Evaluation. Using the Partition distance computation algorithm described in section 4.1, the partition distance d (M, GS) for each model M, and the CIR metric values were computed.

4.3 Threats to validity

Some *internal, construct,* and *external* threats can affect the validity of the obtained results.

Internal validity threats concern factors that may affect a dependent variable and were not considered in the study. The subjective steps of the experiment may influence obtained results. In particular, it is possible that the Gold Standard FSM model be defined in different ways, depending on the involved software engineer experts. Analogously, the validation of the FSM models produced by the technique may vary depending on the subjective opinion of the expert. In order to limit these threats, we involved two experts of RIAs for producing the GS models and validating the

reconstructed FSMs, for obtaining a more reliable and objective result. Moreover, the representativeness of user session traces may be threatened by the maturity effect of the same persons involved in trace collection. We limited this effect by involving two/ three students per application, who had no previous experience about the applications.

Construct validity threats concern the relationship between theory and observation. We used the partition distance for effectiveness evaluation and as main cost factor of the validation step of the technique. It is possible that counting the number of interface move operation between partitions may not be the only way for assessing effectiveness and the validation effort. However, we used this metric since it provides an objective way for comparing two partition sets. Moreover, another threat consists of the fact that we evaluated the technique effectiveness and cost by taking into account only the states of the reconstructed FSM model, and not its transitions. However, this choice can be considered a valid approximation for preliminarily validating the proposed approach. Further experiments should address this aspect in a specific manner, in order to obtain confirmations of the obtained results.

External validity threats are conditions that limit the ability to generalize the results of our experiment to other contexts, such as other RIAs. Since our experimental data referred to only four RIAs, no statistical evidence could be deduced. However, the selected applications were from different domains and were representative of Rich Internet Applications. Additional studies involving more applications, having different interface styles are needed to confirm or confute the obtained results.

4.4 Results

4.4.1 Evaluating Effectiveness. Using the experimental procedure, the reverse engineering technique reconstructed a FSM for each considered application, equivalence criterion, and set of analysed execution traces. Thus, the CIR value for each FSM was evaluated. For brevity, we report in Figure 5 only the FSM CIR values for application W1, depending on the considered equivalence criterion and on the analysed execution traces. In this figure, execution traces are ordered on the horizontal axis on the basis of their size (that is, number of included interfaces).

The CIR values indicate that the FSM models we obtained by criterion C3 approximated the respective Gold Standard model very well (their CIR values were always not less than 75%), differently from the models reconstructed by C1 and C2. Analogous considerations were done for W2 e W3, whereas, for W4, a similar good result was obtained by criterion C2.

In order to explain the effectiveness difference between criteria, we analysed the characteristics of RIA interfaces included in the set of analysed traces. We deduced that the C3 criterion worked well (that is, it effectively classified equivalent interfaces) if the RIA interfaces mostly presented collections (such as tables or lists) of active widgets with the same tag, but with different and dynamically defined collection size. Vice versa, C2 worked well in case of interfaces without this type of collections. Finally, C1 was the less effective criterion in both types of interfaces, since it did not consider the visibility and enabling properties of active widgets.

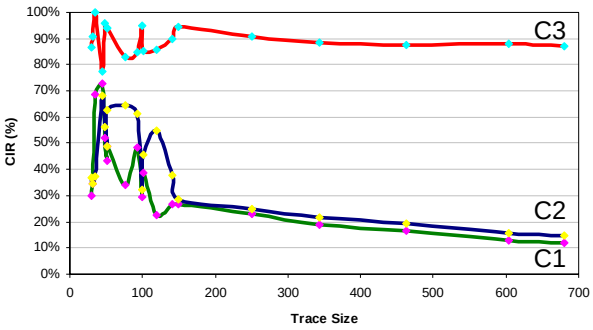

Figure 5: The values of CIR for W1 (Tudu)

Since in the considered experimental setting C3 was most effective than C2 (in 3/4 applications) and more effective than C1 (in 4/4 applications), we concluded that the most effective criteria were always C2 and C3, but the best criterion between C2 and C3 depended on the characteristics of the analysed application.

Thus, the answer to RQ1 question was that *the interface equivalence criterion actually influences the effectiveness of the technique.*

4.4.2 Evaluating Cost-Effectiveness. For answering the second research question RQ2, we studied the relationship between *cost* and *effectiveness* of the technique. To this aim, we recall that the main cost factors are C_{coll}, which grows with the number of analysed trace user interfaces, and C_{mov} which grows with the partition distance *d(M, GS)*.

Intuitively, the partition distance grows with the trace size, too, and experimental data confirmed this trend. As an example, Figure 6 shows the distance *d(M, GS)* of the W1 application (Tudu) as the size of the trace and the equivalence criterion varied. Consequently, $C_{mov.}$ grows with the trace size.

Hence, since the CIR values did not significantly improve with the size of the trace (see Figure 5), to reduce the cost of the technique without affecting its effectiveness, it is necessary to find the best FSM model (among the ones obtained varying the equivalence criterion) that is obtainable from the shorter execution trace. The best FSM model will be the one having the best CIR value among the ones that include all GS states, with the smaller trace size.

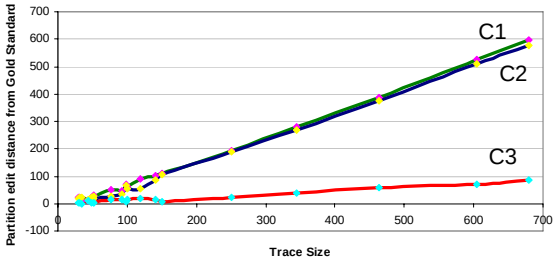

Figure 6: Partition distance d values for W1 (Tudu)

The number of states of FSM and GS models we obtained in the experiment involving the W1 application is reported in Figure 7, as the size of the trace and the equivalence criterion varied. A similar trend was observed for the other applications.

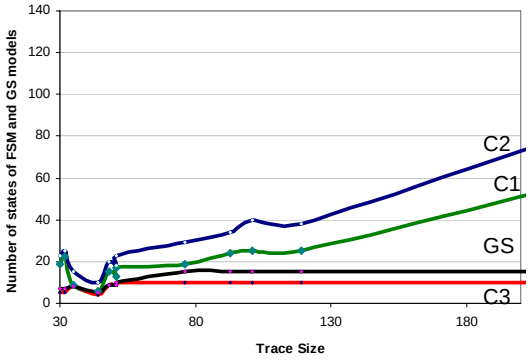

Figure 7: Number of states of GS and FSM models for W1 (Tudu)

Figure 7 shows that the number of states of FSM models produced by C3 definitely tends to a stable value, while it did not happen for models produced by C1 and C2. Hence, we could hypothesize a possible criterion for selecting the execution trace (from a given set of user session traces, with unknown GS state coverage) and the equivalence criterion that has the maximum probability of producing a suitable FSM model with the minimum cost. This *cost-effective selection criterion* indicates (1) of choosing the criterion where the number of states of the reconstructed FSM assumes a stable value, and (2) of choosing the model produced by this criterion from the smaller trace in correspondence of which the number of FSM states assumes the stable value: the related FSM model will be the most cost-effective one.

For validating this criterion, we analysed the CIR values of the models selected by it for each application and for each interface equivalence criterion. The analysis confirmed that this selection criterion actually determined the model with the best cost-effectiveness, and in particular for W1, W2 and W3 this model was produced by the C3 criterion, while for W4 two acceptable FSM models were reconstructed by C2 and C3 criteria. However, the model produced by C2 was the best one because it had the better CIR value (100%).

The following Table 6 reports, for each Web application, the size of the trace from which the most cost-effective FSM model was reconstructed and, for this model, the equivalence criterion that produced it, the number of states, CIR and *d* distance values.

Table 6: Data about FSM models with the best cost-effectiveness ratio

RIA	Trace length	Best Criterion	FSM states	CIR	d
W1	93	C3	10	85%	14
W2	23	C3	2	65%	8
W3	40	C3	4	100%	0
W4	60	C2	19	100%	0
W4	60	C3	19	62%	23

4.5 Discussion

The experiment showed that a key point of the proposed reverse engineering technique is represented by the interface equivalence criteria that allow dynamically produced execution traces of the application to be analysed and simplified in order to abstract a representative model of the RIA behaviour. These criteria are general and reusable for any type of client interfaces of RIAs, differently from the technique [Mar08] that requires that specific features allowing the correct classification of equivalent DOM states be tailored manually with application-specific mechanisms. Moreover, their effectiveness on discriminating different DOM states is not dependent on the choice of any similarity threshold, differently from the 'Levenshtein' distance-based technique proposed by [Mes08].

As the experiment showed, the most effective criterion will depend on the characteristics of analysed RIA client interfaces. However, in the reverse engineering

process, given a set of execution traces, it is possible to use all proposed criteria and generate all the corresponding FSM models in an automatic way, and to find the model to be submitted to a manual validation activity on the basis of the cost-effective selection criterion that emerged from our experiment. The selection criterion, indeed, is able to indicate the FSM model with best cost-effectiveness ratio.

5. Conclusions

In this paper we presented the results of a validation experiment involving four real Web applications that showed cost-effectiveness of a reverse engineering technique for obtaining a model of a Rich Internet Application behaviour by dynamic analysis.

The applicability of the proposed reverse engineering approach depends just on the capability of analysing the client side of the RIA, so it can be used for any type of RIAs (such as Javascript, Flash, or Silverlight based RIAs), provided that specific client interface analysers are available.

The model is reconstructed on the basis of the hypothesis that the analysis of the user interactions with the client interface can produce a good approximation of the RIA behaviour. This hypothesis is acceptable and coherent with an incremental comprehension approach that preliminarily abstracts the salient points of the dialogue between the user and the application, and, subsequently, deepens the comprehension by taking into account other types of interactions, too (like server-side asynchronous events, or time events). We plan to address these further aspects of the problem in future work. Moreover, as regards the scalability of the proposed technique, we plan to carry out further experiments in order to assess the relationships between the scale of the RIA and the costs of our technique.

References

[Alm99] A. Almudevar, C. Field, "Estimation of single-generation sibling relationships based on DNA markers", *J. Agricult. Biol. Environ. Statist.*, American Statistical Association, 4(2), 1999, pp.136–165.

[Ama08] D. Amalfitano, A.R. Fasolino, P. Tramontana, "Reverse Engineering Finite State Machines from Rich Internet Applications", *Proc. of the 15th Working Conference on Reverse Engineering*, 2008, IEEE Computer Society Press, pp. 69 - 73

[And05] A. Andrews, J. Offutt, R. Alexander, "Testing Web applications by modeling with FSM", *Software and System Modeling*, 2005, 4 (3), pp. 326-345

[Bel06] F. Belli, C. Budnik, L. White, "Event-based modelling, analysis and testing of user interactions: approach and case study", *Software Testing, Verification and Reliability*, 2006; 16: 3- 32.

[Bin99] R.V. Binder *Testing Object-Oriented Systems. Models, Patterns, and Tools*, Addison Wesley, 1999.

[Dil02] G.A. Di Lucca, M. Di Penta, A.R. Fasolino, "An approach to identify duplicated Web pages", *Proc. Of the 26th Computer Software and Applications Conference*, 2002, IEEE Computer Society Press, pp.481 - 486

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

[Gar05] J. Garrett, "AJAX: A new approach to Web applications", *Adaptive Path*, 2005.

[Hor07] B. Hohrmann, P. Le Hègaret, T. Pixley, "Document Object Model Events", W3C, 21 Dec. 2007, available at http://www.w3.org/TR/DOM-Level-3-events/events.html

[Kon05] D.A. Konovalov, B. Litow and N. Bajema, "Partition-distance via the assignment problem", *Bioinformatics* 21(10), 2005, pp. 2463-2468.

[Mai08] M. de A. Maia, V. Sobreira, K.R. Paixão, S.A. de Amo, I.R. Silva, "Using a Sequence Alignment Algorithm to Identify Specific and Common Code from Execution Traces", *Proc. of 4th Int. Workshop on Program Comprehension through Dynamic Analysis*, 2008, 6- 10.

[Mar08] A. Marchetto, P. Tonella, F. Ricca, "State-Based Testing of Ajax Web Applications", *Proc. Of 2008 Int. Conference on Software Testing, Verification and Validation*, IEEE Computer Society Press, pp. 121-130.

[Mem03] A. Memon, L. Banerjee, A. Nagarajan, "GUI ripping: reverse engineering of graphical user interfaces for testing", *Proceedings of the 10th Working Conference on Reverse Engineering*, 2003, IEEE Computer Society Press, pp.260 - 269

[Mes07] A. Mesbah, and A. van Deursen, "An Architectural Style for Ajax", *Proc. of the Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA '07)*, IEEE Computer Society Press, pp. 9-18

[Mes07b] A. Mesbah, A. van Deursen, "Migrating Multi-page Web Applications to Single-page AJAX Interfaces" *Proc. of 11th European Conference on Software Maintenance and Reengineering*, 2007, IEEE Computer Society Press, pp. 181 - 190

[Mes08] A. Mesbah, E. Bozdag, A. Van Deursen, "Crawling AJAX by Inferring User Interface State Changes", *Proc. Of Eight Int. Conference on Web Engineering*, 2008, IEEE Computer Society Press, pp. 122-134

[Pre07] 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)*, pp. 23-30.

[Sam04] S. Sampath, V. Mihaylov, A. Souter, L. Pollock, "A Scalable approach to user-session based testing of Web applications through Concept Analysis", *Proc. of 19th International Conference on Automated Software Engineering*, 2004, IEEE Computer Society Press, pp. 132- 141.