Dynamic Model Extraction and Statistical Analysis of Web Applications: Follow-up after 6 years

Paolo Tonella(1), Filippo Ricca(2)
(1) FBK-IRST, Trento, Italy
(2) Unità CINI at DISI, Genova, Italy
tonella@fbk.eu, filippo.ricca@disi.unige.it

Abstract

In 2002 we proposed a method to reverse engineer a Web application model. The proposed method deals with dynamic Web applications, consisting of server components, typically interacting with a persistent layer, which build the Web pages displayed on the browser dynamically. Dynamic analysis and page merging heuristics were used for model extraction.

The proposed model was successfully adopted in the Web analysis and testing research community. However, the features of future Web applications (involving rich client components and asynchronous communication with the server) challenge its future applicability. In this paper, we analyze the key properties of the 2002 model and identify those modeling decisions that remain valid and can be used to guide the extraction of models for future Web applications.

1 Introduction

Models are widely adopted from the scientific community to resolve problems in many different areas from telecommunications, to bio-informatics, to civil and software engineering.

Importance of models in software engineering is before everybody’s eye. In software engineering models are used for several purposes as a means of abstraction, for enabling concise communication among the parts, as means to improve interoperability [1] and also as programming language (see MDA [11]). Many software “constructs” and processes can be represented as models and thus many software engineering problems can be reformulated (and simplified) as a problem about the model.

This is particularly true for Web applications where models are used for design purposes [2, 16, 3] but also as a support to analysis, testing and re-structuring of Web applications.

Roughly speaking, two somewhat opposite (or complementary) approaches exist in Web engineering (the same is clearly true in Software engineering): (a) Forward Engineering, which proposes models and methods aimed at supporting the design of Web applications; (b) the Reverse Engineering approach, which assumes that a Web application already exists and supports its analysis, testing and re-structuring using as base the abstract models recovered from the implementation.

The approach we have adopted in our 2002 work [17] is based on the reverse engineering paradigm. Under the assumption that a Web application already exists, we investigated techniques to support its analysis and testing. Our starting point was therefore the actual implementation of the Web application (Web pages, server programs, etc.). The techniques we investigated worked on the abstract models recovered from the implementation.

In this paper, first, we summarize the Web application model proposed in 2002 [17] and show how the model is extracted by means of a Crawler starting from the home page of the target Web application. Second, we confront it with alternative proposals [4, 6, 7, 9, 15] showing similarities and differences. Most of them are extensions or improvements of our model, others used it as a base to propose novel uses of it [6]. Third, we discuss the weaknesses and limitations of our model (and in general of all similar models). The model proposed in 2002 [17] had weaknesses in itself (e.g. merging different dynamically generated pages) but new technologies used for Web application development (e.g., AJAX) make them worse. Finally, we characterize the features that future Web models are expected to accommodate and we identify those features of our 2002 model that remain still valid for the construction of expressive models of future Web applications. Weaknesses and limitations prevent the usage of our 2002 model “as is”, to analyze and test future Web applications, but the proposed model can still be considered a useful starting point when trying to model Web applications of the future Internet.

This paper is organized as follows. Section 2 reviews
the 2002 model in a critical way, compares it with others Web application models and presents weaknesses and limitations. Section 3 characterizes the features that future Web models are expected to accommodate, while Section 4 bridges the gap between the 2002 model and future Web application models. Finally, Section 5 concludes the paper and outlines future work.

2 A critical review of the 2002 model

In this section, we summarize briefly the Web application model proposed in 2002 [17] and show how alternative modeling efforts went in a substantially similar direction. Then, we discuss the weaknesses and limitations of those models. As will be shown in the next sections, such limitations did not prevent usage of those models to effectively analyze and test Web applications, but they are of increasing concern when trying to model Web applications of the future Internet.

2.1 The 2002 model

The meta-model underlying the 2002 Web application model is shown in Figure 1. First-class entities in this model are:

- **Web pages**: the page being displayed on the Web browser is considered the most natural choice for the interaction state.

- **Navigation links**: the main mechanism triggering state transition is considered to be hyperlink-based navigation.

- **Frames**: the main mechanism used to partition a page into independently evolving parts is considered to be the use of frames. If pages define the interaction state, frames support the decomposition of a state into independent substates.

- **Forms**: form submission is considered the main mechanism for the client to request services to the server.

Web pages may be static or dynamic. Dynamic Web pages are constructed by server side components in response to a user request, such as a form submission. While in principle it is possible to collapse all dynamic pages constructed by the same server component into a single Web page (*implicit-state model* [17]), often the same server component is used for different purposes and builds different Web pages, depending on some input parameters or on the state of the session. Hence the need to distinguish dynamic pages produced by the same server side component. In the *explicit-state model* [17], this is achieved by means of a page merging operation. The set of all possible dynamic pages is partitioned into a finite set of equivalence classes of dynamic pages. We proposed to recognize such equivalence classes based on prior knowledge or on heuristics (another merging classification has been presented in [5]):

- **Prior knowledge**: session, hidden and input variables are known *a-priori* to partition the space of dynami-
Figure 2. Implicit state model. Nodes with grey background are server programs. Input variables (with optional default values) and hidden variables (with values) are separated by ‘;’.

Figure 3. Explicit state model. Nodes with grey background are server programs. Input variables (with optional default values) and hidden variables (with values) are separated by ‘;’.

To clarify this point, we show an example (taken from [14]) of Web application for which both implicit-state (Figure 2) and explicit-state models (Figure 3) are given. The application consists of an initial static page \( H \) from which the user can navigate to a server program \( S \) through a link associated with a parameter, \( \text{state} \), which is assigned the constant value 1. \( S \) builds a dynamic HTML page, the content of which depends on the value of variable \( \text{state} \) which is received by \( S \). In particular, with \( \text{state} = 1 \), \( S \) builds a page containing one form which collects the values of variables \( x \) and \( y \) and transmits a value of \( \text{state} \) equals to 2 as a hidden variable. This is represented in the implicit-state model (Figure 2) as a \( \text{submit} \) link guarded by the condition \((\text{state}=1)\). Such a link is generated inside page \( D \) only when \( S \) receives a value of \( \text{state} \) equals to 1. Then, the server program \( S \) is invoked for the second time, now with \( \text{state} = 2 \). The behavior in this situation is different from the previous one, and the output page contains two new forms, respectively devoted to collecting the values of \( a \) and of \( b, c \), while it does not contain the previous form. This is the reason for the two \( \text{submit} \) links guarded by the condition \((\text{state}=2)\). Finally, the server program \( S \) is executed again, either by the first active form (gathering \( a \) as input) or from the second one (gathering \( b \) and \( c \)). The result of this execution is still different and the dynamic page \( D \) that is built now does not contain any form (\( \text{state} \) is equal to 3 and therefore all conditions are false). Its content varies also in the two cases where either \( a \) or \( b, c \) are filled in by the user. The explicit-state model of this example of Web application is provided on Figure 3. The server program \( S \) and the dynamic page \( D \) have been split into 4 pages, associated to the 4 different behaviors that may occur during an interaction, corresponding respectively to \( \text{state} = 1, \text{state} = 2, \text{state} = 3 \) and \( a \) gathered, and \( \text{state} = 3 \) and \( b, c \) gathered.

2.2 Model extraction

The analysis we proposed for model extraction [17] can be considered dynamic in that it needs the execution of the Web application; server pages and other server components are not statically analyzed. To obtain the model, the HTML pages of the target Web application are downloaded by a crawler\(^1\) [8] able to handle forms and dynamic pages (i.e. pages generated at run-time by a server side program). Input values which cover all relevant navigations are retrieved from an external repository of meaningful input values (previously inserted by the user or captured in log files), and downloaded pages are merged, in order to produce an abstraction over the set of HTML pages downloaded. Once the next Web page is visited by the spider, it is

\(^1\)A Web crawler (also known as a Web spider or robot) is a program that automatically traverses the hyperlink structure of a Web application and retrieves some information for the user.
compared against the previously visited pages. It is added to the model as a new equivalence class of pages only if it does not belong to any of the previously constructed equivalence classes of pages, according to any of the adopted criteria for page merging. Otherwise it is just added to the equivalence class it belongs to. The resulting model is computable in presence of high dynamism and requires the ability to parse just HTML. On the other side, the model obtained may be partial, if the inputs used to produce it do not cover all relevant behaviors of the Web application.

The model constructed in this way is then used for analysis and testing. Based on statistical information available from the user session logs, transition probabilities are computed. Among the useful analyses that can be carried out on the resulting model is the average path length to each page. When used for testing, coverage criteria are derived from the model. For example, coverage of the most likely paths can be used as a test case prioritization criterion.

### 2.3 Other Web application models

Several models for representing a Web application can be found in the literature. Most of them (WebML [2], OOHDM [16], Conallen [3]) have been proposed for developing Web applications. These models can not be used for representing an existing Web application because the information they represent does not reach the degree of granularity required for analysis, testing and understanding purposes. Design models aim at describing the Web application from a logical point of view at a high level of abstraction while reverse engineered models represent the components of a Web application at the implementation level.

The forward engineering Web application model closer to ours is that proposed by Conallen [3]. Web pages are considered first-class elements, and are represented as objects, using UML. Similarly, all other architecturally relevant entities as, for example, links, frames, and forms, are explicitly indicated in the model. The main difference between Conallen’s and our UML model of Web applications is in the emphasis given to design vs. analysis. In fact, the model by Conallen aims at describing the site from a logical point of view, as required when it is being designed. On the other side, we focused our model on the implementation of the site, which is the starting point for analysis and testing, and on the navigational features of the site.

Several Web application models at implementation level [4, 6, 7, 9, 15] have been proposed in recent years as a variation, an improvement or a parallel, independent development with respect to our model [17]. Most of them share a lot of features with our model.

Di Lucca et al. [9] adopted a similar meta-model, but have chosen white box reverse engineering for the server pages. Their technique is based on static and dynamic analysis. Static analysis of the HTML pages/server programs composing the Web application is realized by means of parsers and static analyzers, while dynamic analysis is executed to integrate and validate the model extracted statically. This technique presents two drawbacks: (i) it is source code dependent, i.e. all languages used in the implementation of the Web application need a specific parser, and (ii) it has heavy limitations when applied to very dynamic Web applications. Usually, with this kind of applications the model extracted by means of static analysis misses a lot of information, available only at run-time, with the consequence that all the relationships between components have to be recovered (often manually) by means of dynamic analysis.

In Di Luca et al.‘s representation [9], differences between static client pages and dynamic client pages, passive Web objects (e.g., images) and active Web objects (e.g., scripts) are remarked, and interface objects (i.e., objects that interface the Web application with a DBMS or an external system) are added. On the contrary, the model presented in [17] aims at explicitly representing user navigations. Consequently, internal entities such as the Web objects and interface objects are not considered.

Elbaum et al. [6, 7] propose a Web application testing approach that utilizes data captured in user sessions (stored in a modified log file) to create automatically test cases. The authors describe two similar implementations of this approach, and a “hybrid” implementation based on our 2002 model that combines the idea of using data captured in user sessions with structural testing [13]. Instead of inserting manual inputs into automatically derived test cases [13], the inputs are recovered from the modified log file. The coverage is still guaranteed and the process is completely automatic. This approach can be considered an improvement of [13].

### 2.4 Issues and limitations

Recent trends and technologies used for Web application development raise issues on the applicability of the 2002 model as is. Moreover, the model had weaknesses in itself. They were overcome by means of heuristics that have been proven viable and effective, under some assumptions on the characteristics of the dynamic Web sites being modeled. Current and future Web applications involve technologies that often violate such assumptions.

The first-class entities in the 2002 model are Web pages, navigation links, frames and forms. However, this set is not always adequate for modern Web applications. In particular, frames are no longer the mainstream mechanism for Web page decomposition. Server side includes, DOM (Document Object Model) manipulation and dynamic page generation using SPAN/DIV and stylesheets tend to replace frames, thus solving a number of issues related to the
usage of frames (e.g., recursive inclusion of the top-level page, with all its frames, inside a nested frame). Navigation is achieved via Javascript, in addition to standard anchors. Communication with the server is also not necessarily mediated by form submission.

The aim of the 2002 Web application model was that of “describing a Web application in terms of composing pages and allowed navigation links” [17]. Nowadays this is not enough, since the client’s state is not necessarily modified by means of some navigation from page to page. Code running on the client may operate directly on the page structure without involving any page loading from the server. So, the model of a Web application as a hypermedia – where the state is the current hypernode and navigation triggers state transition – does not account for the level of dynamism available on the client side. Even though no navigation through hyperlinks occur, the user interaction may result in a sequence of transitions from state to state on the client, as well as in a sequence of actions performed by the server.

Frames are the native way of supporting independent change of different page portions. They are available directly as an HTML construct. However, they have a number of limitations and constraints\(^1\) that prevent Web developers from designing arbitrary decompositions and decomposition mechanisms — for example: frames can be included recursively; search engines do not deal with frames well; and, frames create problems with printing. Hence, they were soon abandoned in favour of server-side solutions. Instead of leaving the browser control on the independently evolving parts of a page, it is the server that implements such a control, without any limitation or constraint. Hence, usage of frames for page decomposition in a general purpose Web model looks over-restrictive, with respect to the increased number of possible ways to realize it.

Forms and form submission remains an important mechanism for client-server communication. However, it is no longer the exclusive one. Javascript code may send and receive messages to/from the server asynchronously, without involving any form submission and page reloading, making the exchange of information richer, but also more difficult to track and analyze.

Overall, the 2002 model was successful in accommodating the server-side dynamism involved in emerging Web applications of those years, but it looks now inadequate to accommodate the increasing level of dynamism of modern Web applications, supporting a richer client side interaction.

### 3 Modeling the future Web

In this section, we characterize the features that future Web models are expected to accommodate. The novel technologies and interaction paradigms that characterize the future Web demand for novel models and model extraction algorithms. One notable, mainstream example of such technologies is Ajax, but we want to abstract from the technological details of Ajax and we prefer to deal with its novel features, regarded at a more conceptual level.

#### 3.1 Modeling the client side state

The user experience offered by hyperlink based navigation is quite limited. In a pure hyperlink based model, any modification of the page structure and content is mediated by a navigation action, which involves a communication with the Web server. Rich multimedia content and responsive user interfaces demand for mechanisms that go beyond hyperlink navigation. More advanced features, such as adaptive interfaces and user centered interactions require alternative mechanisms. Overall, more computational load is moved to the client, which becomes richer than just a browser rendering an HTML page.

Several solutions are available to develop a rich client experience (Ajax, Applets, Flash, etc.). Some deviate radically from the original Web application model (e.g., Applets, Flash), using an interpreter installed as a browser plugin for a language that has nothing to do with traditional HTML Web pages. Other approaches (e.g., Ajax) take advantage of the reflection capabilities of Javascript to expose and manipulate the Web page structure directly. This is achieved by means of the Document Object Model (DOM), a representation of a Web page accessible and modifiable through Javascript. In these approaches, the browser remains the main engine running the client code, but the reflection capabilities offered by the DOM make it much more powerful. The client state can be changed in response to user events without any communication with the server, since Javascript code can be activated by user events and can modify the DOM. The client state to be modeled becomes richer too.

#### 3.2 Modeling asynchronous interactions

The synchronous request-response model of early Web applications is also not fully adequate with respect to the needs of the future Web. A richer, more responsive client can express its potential only if its operations are decoupled, whenever possible, from the server computation. It is unacceptable to block the client whenever some information has to be transmitted to the server.

The current technological trend aims at replacing the old synchronous request-response protocol with one where asynchronous requests can be also sent to the server. The server replies with an asynchronous response that triggers the execution of an event handler on the client. Highly

\(^1\)http://www.mediacollege.com/internet/html/frames/pros-cons.html
interactive and content rich Web applications require a responsive client, that can be effectively implemented only by means of asynchronous communication with the server.

By permitting multiple threads running asynchronously in parallel, the complexity of the Web application being developed increases remarkably. All problems related with concurrency become also problems of Web developers. Data races and deadlocks are just two examples of problems associated with concurrently executing application threads. Designing these applications will be a challenge for nowadays Web developers, since no adequate notation has been developed yet in forward engineering Web design models. Model extraction for analysis and testing purposes will be a major challenge for researchers as well.

4 Analysis and testing of future Web applications

In this section, we bridge the gap between the 2002 Web application model and the future Web model. We revise the basic decisions made in the 2002 modeling approach, trying to identify those that remain still valid for the construction of expressive models of future Web applications.

4.1 Dynamic analysis for model extraction

Future Web applications will involve dynamic DOM manipulation and asynchronous client-server interaction. Static model extraction becomes immediately impractical, given such two features, hence dynamic analysis is expected to remain the mainstream approach. In particular, manipulation of the DOM occurs at run time and is based on reflection primitives provided by Javascript. Static analysis of the usage of such primitives is expected to provide few clues on the page modification being performed, since it will necessarily involve several run-time parameters. In the general case, the desired output is undecidable. Hence, resorting to dynamic analysis, as proposed for the 2002 model, will remain a key decision for model extraction.

Dynamic model extraction for advanced Web application based on asynchronous communication and DOM manipulation is a hard task [10]. It involves similar abstraction problems as those encountered in 2002 with page merging. Dynamic analysis usually produces a huge amount of information, in the form of execution traces, that needs to be properly abstracted in order to become meaningful. Hence the need for appropriate abstraction functions.

In model extraction for Web applications manipulating the DOM, the natural choice for the client’s state is the DOM state. This corresponds to the explosion of the substates in a Web page, which is a single state in the 2002 model. In the extreme case of a Web application consisting of just one page manipulating the DOM through reflection, it provides the entire application state.

Abstraction of the DOM state involves generic abstraction functions (e.g., a \texttt{UL} DOM element may be \texttt{null}, may contain zero items, or may contain 1 or more items). However, domain specific abstractions may be needed for DOM elements that are not intrinsically constrained by their type (e.g., an \texttt{INPUT} element) [10].

4.2 Verification of structural properties

Verification of structural properties of future Web applications will be of fundamental importance to deliver the desired level of quality. Modeling the asynchronous behaviour of these applications and identifying places where data races or deadlocks may occur will provide invaluable feedback to developers. This is similar to the analyses defined on top of the 2002 model [17].

An example of structural properties that are relevant for future Web applications are the 	extit{synchronism warnings} detected in [10]: Let us indicate as \( r_1 \) and \( r_2 \) two asynchronous requests sent to the server and let \( c_1 \) and \( c_2 \) be the corresponding callbacks, activated on the client when receiving the responses. The following execution sequences may occur:

- **Nominal (no reordering):** \( (r_1; c_1; r_2; c_2) \)
- **AsyWarn1 (swapped callbacks):** \( (r_1; r_2; c_2; c_1) \)
- **AsyWarn2 (dependent request):** \( (r_1; r_2; c_1; c_2) \)

\textit{Nominal} is the usually expected sequence, where each request is immediately followed by the associated callback. \textit{AsyWarn1} occurs when \( c_2 \) starts before \( c_1 \). It may produce an incorrect final state of the client if both \( c_2 \) and \( c_1 \) manipulate the same DOM portion. The reasons for \( c_2 \) starting before \( c_1 \) may be network delays, scheduling of threads on the server (second thread terminates before first one starts), scheduling of callback activation on client (second callback scheduled before first one), etc.

\textit{AsyWarn2} occurs when the second event, which generates \( r_2 \), is triggered before activating the callback for the first request \( r_1 \). The final state may be correct or not, depending on \( r_2 \). If the request \( r_2 \) depends on the results produced by the execution of \( c_1 \) (for example, the request \( r_2 \) includes information from the DOM and this information is modified by \( c_1 \)), \( r_2 \) is sent to the server with incorrect information. As a consequence, the final state may be incorrect or some output values may be different from the expected ones.

Web developers to be unaware or not to care too much about these synchronism problems, probably because they work under the optimistic assumption that the time between GUI events is much higher than the time to send the asso-
ated requests to the server, compute the responses, send them back to the client and run the related callbacks. However, this assumption might be wrong for example in cases where the network is slow or the server is overloaded.

4.3 Coverage testing and automated test case generation

Future Web applications will involve more interaction, reflection and asynchronous behavior. As a consequence they will be more difficult to test. Model extraction will be a fundamental prerequisite for testing. Similarly to the use of the extracted model proposed in 2002 [17], dynamic model extraction of future Web applications will open to the possibility of novel test coverage criteria definition and test case generation methods.

Paths in the extracted model can be associated with test cases. However, not all paths are equally good at revealing faults. In particular, long interaction sequences are expected to have a bigger fault revealing potential. However, the number of long sequences derived from a model tends to increase exponentially with their length. Hence the need to define test coverage criteria and test generation methods that go beyond the simple coverage of the extracted model. Among the large number of long interaction sequences, only those that exercise the most critical code-DOM interactions should be selected for testing.

Preliminary results in this direction have been obtained, based on the notion of semantically interacting events [10]. Javascript events $e_1$ and $e_2$ interact semantically if there exists a state $S_0$ such that their execution in $S_0$ does not commute, i.e., the following conditions hold:

$$ S_0 \Rightarrow e_1; e_2 S_1 $$
$$ S_0 \Rightarrow e_2; e_1 S_2 $$
$$ S_1 \neq S_2 $$

where $S_0$ is any state in the Web application model. We have a semantic interaction whenever the effects on the DOM state of the two callbacks $e_1$ and $e_2$, associated with $e_1$ and $e_2$, are not independent, i.e., swapping the order of execution brings the application to a different state. Semantically interacting event sequences are considered more likely to reveal faults than sequences involving non-interacting events, since in the latter case the length of the sequence is not exploited: shorter sequences would reveal the same faults. Similar results have been obtained in GUI testing [12, 18].

5 Conclusions

Web engineering is a rapidly changing field of software engineering where technologies and approaches tend to become obsolete quite quickly. Mainstream Web applications in 2002 were based on dynamic HTML pages constructed at run-time by server components. We defined a Web application model and derived a set of analysis and testing techniques from it based on that development approach.

The future Web applications are delegating increasingly more computation to the client. To achieve higher responsiveness, GUI management and server communication are going to change. We expect that those changes will in part invalidate our modeling approach. However, some fundamental issues will remain the same: a high level of dynamism will be involved in run-time DOM manipulation and asynchronous message handling. Hence, we think that analysis will be still dynamic and based on merging/abstraction heuristics similar to those proposed in 2002.

Our future research agenda comprises a set of novel modeling methods to be used with future Web applications. We plan to use those methods for Web application analysis and testing. We think that our experience with the 2002 Web application model will be the starting point and will guide our key modeling and model extraction decisions.

References


