Adaptive REST Applications via Model Inference and Probabilistic Model Checking

Carlo Ghezzi
Politecnico di Milano
DeepSE Group at DEI
Piazza L. da Vinci 32, 20133 Milano, Italy
Email: carlo.ghezzi@polimi.it

Mauro Pezzè
and Giordano Tamburrelli
Università della Svizzera Italiana
Faculty of Informatics, Via Buffi 13, Lugano, Switzerland
Email: {mauro.pezze | giordano.tamburrelli}@usi.ch

***Abstract*—In this paper we present a novel approach for adaptive REST Web applications that focuses on adaptation against changes in the navigational behaviour of users. The proposed solution exploits the Web server's log file to infer a Markov model that captures the navigational behaviour of system users over time probabilistically. The model is inferred incrementally as soon as new requests are issued to the server, and is analysed periodically to verify quantitative properties by means of probabilistic model checking. The results of the run-time verification trigger ad-hoc adaptation policies, which adjust the application to the user behaviours captured by the inferred model. The paper discusses the advantages of adopting probabilistic model checking for Web applications in terms of incrementality, retroactivity and efficiency, and illustrates these characteristics as well as the applicability of the approach with a practical example.**

I. Introduction

Building effective Web applications presents several technical challenges that relate to scalability, privacy and security. Advances in research and technology have produced many methodologies and techniques to overcome these difficulties: modern Web applications can scale up to millions of users, perform secure economic transactions, and integrate with each other via public APIs. Despite the big improvements in methods and techniques, some challenges are still open, and the effective development of Web applications still requires further investigation, as witnessed by the many annoying failures that still occur in modern Web applications [1].

A crucial factor that affects directly the success of a Web application is the knowledge and understanding that engineers have about the final behaviour of the system users. Underestimating the importance of this factor may lead to technical as well as non-technical failures. For example, from a technical perspective, an inaccurate knowledge of users' navigation preferences may lead to unsatisfactory user experiences. This issue may have–in turn–severe non-technical consequences such as a negative impact on revenues of an e-commerce application.

The user's behaviours cannot be fully anticipated or precisely predicted before the system deployment. Indeed, only publicly released applications allow us to collect statistics and metrics accurate enough to optimise the running implementation. Moreover, users may have multiple and heterogeneous navigational behaviours that depend on several factors, like user location, time of day, preferred browser, etc. To capture all these behaviours, engineers must analyse in detail the runtime usage of the systems to carefully mine navigational patterns that describe the users' navigational behaviours. This process is expensive, time cosuming and error prone. Moreover, even if engineers succeed in capturing all the relevant user behaviours, these behaviours may unpredictably change over time thus invalidating the analysis efforts.

We claim that the next generation Web applications should be able to autonomously detect the navigational preferences of their users, and consequently self-adapt and self-optimise over time. In other words we believe in *user-aware* and *adaptive* Web applications. To this end Web applications should reason about the user requests and react as soon as certain user behaviours are detected. A runtime process of self-analysis and adaptation requires *incremental* mechanisms that can properly react to users requests as soon as they are issued to the system. In addition, because of the large number of Web applications already in operation, the mechanisms should be *retroactively* applicable to existing systems.

This paper describes the preliminary results of an approach conceived to address these issues focusing on REST[1] architectures. The proposed approach continuously monitors the log file generated by a REST Web server, incrementally analyses the data appended to the log file by the server, and infers a Markov model, which probabilistically captures the navigational behaviours of users. The approach automatically analyses the model to formally verify quantitative properties specified by the application owner by relying on probabilistic model checking (PMC) [3]. The results obtained by this run-time verification trigger ad-hoc adaptation policies, which adjust the implementation of the application to match the user behaviour as captured by the model. For example, in an e-commerce application, the analysis may compute the probability that a certain product on sale is actually seen by a user. If such probability falls below a certain threshold the application may self-adapt to modify the order in which the products are visualised in order to optimise sales versus stock availability.

The paper contributes to Web engineering in three main ways. It lays the foundations of a model-based approach to capture users' behaviours specifically conceived for REST architectures. It proposes a practical application of probabilistic model checking in the domain of Web applications. It proposes

[1] *REpresentational State Transfer* [2].

978-3-901882-50-0

an approach that applies incrementally and retroactively to legacy systems, differently from other related approaches, like Google Analytics [4].

The remainder of the paper is organised as follows. Section II introduces the proposed solution and summarises the main steps. Section III introduces a working example that is used in Section IV to present the details of the approach. Section V describes the prototype implementation of the proposed solution. Section VI discusses related work, and Section VII summarises the results and illustrates the ongoing research.

II. Incremental runtime analysis of REST applications

REST is an increasingly popular architectural style in which requests and responses are built around the *stateless* transfer of resources via URLs, and the state of the conversation among client and server is univocally identified by the URL of requests. We focus on this style because of its intrinsic advantages in terms of scalability, generality, and ease of encapsulation, which is particularly important to deal with in legacy systems. A complete discussion on REST is beyond the scope of this paper and can be found in [2], [5].

We propose an approach that captures the navigational behaviour of users at runtime, by inferring a Markov model from the log file generated by the application, and triggers ad-hoc adaptation strategies that adjust and optimise the running implementation. The approach is articulated in the following five steps:

Identifying the states: We automatically cluster the URLs of the application by means of regular expressions, and map the identified clusters to a finite set of labels.

Specifying the properties: The application owner defines a set of properties to be monitored and verified at runtime. The properties capture characteristics of the users relevant for the application domain. The properties, as described later on, are specified in an appropriate formalism suitable for PMC: probabilistic computational tree logic (PCTL) [6], [3]. The Appendix provides its formal definition.

Inferring the model: We automatically deploy an *inference engine* within the running implementation. The inference engine monitors the application log file, and infers a discrete time Markov chain (DTMC) [7], [3]. DTMCs are finite state automata augmented with probabilities: each state is characterised by a discrete probability distribution that regulates its outgoing transitions. In our case the states of the inferred DTMC correspond to the labels associated to clusters of URLs, and the probabilistic transitions correspond to the transitions between states as inferred from the log file. We provide a formal definition of DTMCs in the Appendix.

Analysing the model: The PMC comes into play periodically to quantitatively evaluate the specified properties against the inferred DTMC. The obtained results are numerical or boolean values that represent the matching of the desired properties with the inferred behavior, depending on the nature of the properties.

Adapting the application: The evaluation results may trigger one or more adaptation strategies that are implemented by the application owners by means of code fragments called *adaptation scripts*. Such snippets may re-configure the system, optimise or tune application parameters and, if needed, alert a human operator.

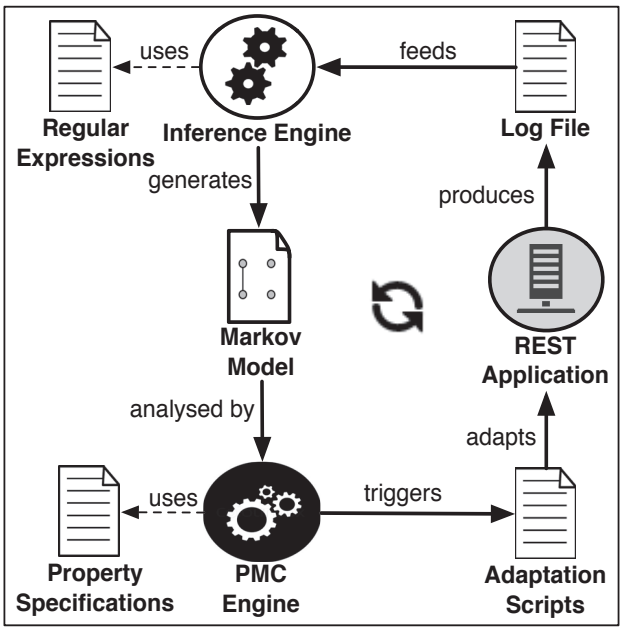

Fig. 1. Overview of the approach.

The first two steps occur at design time, while the others occur at runtime and shape a *feedback loop* between model and implementation as illustrated in Figure 1.

III. The mycardealer.com working example

We illustrate the details of our approach by means of an example that is simple enough to explain the proposed solution and at the same time easily generalizable to real world REST systems. The example is the web presence of a car dealer (*mycardealer.com*), which regularly advertises sales of new and used cars. Users may browse the announcements and contact a sales agent through the website to have additional details and schedule an appointment to see the car they like.

The website is composed of several pages and resources identified by the URLs summarised in Table I (the meaning of the third column will be clear later on). The table reports only a relevant subset of the application URLs, and summarises some of them by means of prefixes. Some of the URLs are parametric, for example /sales/new/?page=⟨n⟩, or have a parametric structure, for instance /sales/new/car/⟨id⟩/. Due to the stateless nature of REST applications, the URLs completely identify the resources of the application and the state of the interaction between the users and the system.

IV. The Approach at Work

In this section we describe the details of the five steps of the proposed solution referring to the working example.

URL	Description	Label
/home/	Homepage of mycardealer.com	*Homepage*
/sales/new/	The first page that lists the announcements of new cars on sale.	*NewCarsPage*
/sales/new/?page=⟨n⟩	The n^{th} page with announcements of new cars on sale, ⟨n⟩ is an integer index.	*NewCarsPage*
/sales/new/car/⟨id⟩/	Detailed view of the (new) car on sale identified by the string identifier ⟨id⟩.	*NewCar*
/sales/used/	The first page that lists the announcements of used cars on sale.	*UsedCarsPage*
/sales/used/?page=⟨n⟩	The n^{th} page with announcements of used cars on sale, ⟨n⟩ is an integer index.	*UsedCarsPage*
/sales/used/car/⟨id⟩/	Detailed view of the (used) car on sale identified by the string identifier ⟨id⟩.	*UsedCar*
/news/list/	The page containing a list of articles related to the car domain (new models, trends, etc.).	*NewsPage*
/news/⟨id⟩/	The page containing a specific article listed in the news page identified by the string ⟨id⟩.	*NewsArticle*
/contacts/?car=⟨id⟩	The URL that submits the form used to contact a sales agent for a specific car identified by the string ⟨id⟩.	*Contacts*
/about/	Details concerning the car dealer: address, partnerships, etc.	*About*
/media/, /jss/, /CSS/	The URLs with these prefixes refer to secondary resources (Javascript, CSS, images, etc.).	-

TABLE I. THE MAIN URLS OF MYCARDEALER.COM

A. Identifying the states

In the first step, we identify a set of regular expressions[2] that map the URLs of the application under analysis to a set of labels. Labels are strings that associate a semantics to sets of URLs. For example, the following regular expression:

$$\hat{}/sales/new/\backslash?page = (\backslash d+)/\$$$

may be associated to the label "*NewCarsPage"* indicating that requests to URLs matching such expression correspond to users interested in announcements of new cars on sale, while the regular expression:

$$\hat{}/news/(\backslash d+)/\$$$

may be associated to the label "*NewsArticle"* indicating that requests to URLs matching such expression correspond to users viewing an article of the news page. The state identification process is performed only for the relevant URLs of the application under analysis. Indeed, URLs associated to secondary resources, not crucial for capturing the behaviour of users, are not mapped to regular expressions. For example, URLs that represent CSS or Javascript resources are not relevant to capture users' navigation actions, since they are automatically requested by browsers, and thus do not require any mapping to labels. Such URLs are simply ignored. The third column in Table I reports the labels associated to URLs via regular expressions for our example. To improve the readability of the explanation we will explain the details about the property specification step later in the paper, and now focus on the transition inference process.

B. Inferring the model

The models inference process is grounded on some basic assumptions that we discuss here. We assume a log file structured as a list of rows that track the interactions between the users and the Web server of the application under analysis. Each row represents a request of a Web resource issued by a client to the Web server. Hereafter we use the terms row and request interchangeably. We also assume that each row contains the following data: the IP address of the user that issued the request to the Web server, a timestamp that represents the time of the request, and the requested URL. These assumptions correspond to the typical pieces of information logged by common Web servers.

The inference engine generates a DTMC where the set of states S contains one state for each label as defined in the state identification step, together with the two additional states *Init* and *End*. The *Init* state corresponds to the initial state of the model (s_0), the *End* state is a sink state and represents the users who leave the system. In our example the Inference Engine produces a DTMC with thirteen states corresponding to the to eleven labels reported in the third column of Table I plus the *Init* and *End* states.

[2]In this paper we refer to the syntax of Java Regular Expressions [8].

```
1.1.1.1 – [20/Dec/2011:15:35:02] – /home/
1.1.1.1 – [20/Dec/2011:15:36:16] – /sales/new/
1.1.1.1 – [20/Dec/2011:15:37:44] – /sales/new/?page=2
1.1.1.1 – [20/Dec/2011:15:38:12] – /new/car/AR811/
1.1.1.1 – [20/Dec/2011:15:40:03] – /sales/new/?page=2
1.1.1.1 – [20/Dec/2011:15:41:12] – /new/car/BA964/
1.1.1.1 – [20/Dec/2011:15:41:20] – /contacts/?car=BA964
```

Listing 1. An excerpt of a log file

The inference engine processes each row as soon as it is appended to the log file. The engine interprets each row of the log file as a *transition* among states in S. A transition is characterised by a *source* and a *destination* state in S. The destination state is computed for a row r by extracting the URL contained in the request and by matching it to the regular expressions provided in the state identification step. The inference engine considers the row r as a transition to the state identified by label L if and only if the first expression matched by the URL corresponds to label L. If the URL does not match any expression the row is discarded.

The inference engine computes the source state for a row r by associating the row to a specific user. The engine extracts the IP address contained in r and assumes that distinct addresses correspond to different users. If a row r has been generated by an IP address processed for the first time the engine assumes r as the first interaction of a certain user with the system. To model such initial interaction the *Init* state is considered as the source state of the transition associated to r. If r has been generated by a previously processed IP, the engine retrieves the destination state of the last row characterised by the same IP address and assigns such state as the source state of r.

In reality, it may occur that two different users that browse the system in different occasions may have the same IP address. To cope with this scenario the inference engine refers to the timestamps and assumes that requests issued by the same IP address but with significantly different timestamps have been issued by different users and treats them consequently. The minimum temporal distance among timestamps that distinguishes two different users is a parameter called *UserWindow*, which can be tuned.

We illustrate these concepts with the running example, and consider the log fragment in Listing 1 which contains the interaction of a user with the system. The listing reports only the information relevant for the analysis, we filtered data not relevant for the example, like the GET/POST request type or the user-agent.

The log represents the interaction of a user who first navigates from the homepage to the page that lists the announcements of new cars. She subsequently browses the second page of the announcements and, later on, navigates to the page of a specific announcement ($AR811$). Subsequently, she decides to navigate back to select a more appealing one ($BA964$). This second announcement is particularly interesting for the user and she decides to contact a sales agent by clicking the appropriate button that corresponds to the URL $/contacts/?car = BA964$.

The inference engine interprets each row as a transition in the DTMC. We start from the first row in the listing that has been generated by IP 1.1.1.1 and we assume that such IP is processed for the first time or that the timeout UserWindow for this address has already expired. Given these assumptions, the row corresponds to an interaction of a new user with the system. Since the URL in the request matches the regular expression associated to the label $Homepage$, the engine interprets such row as a transition from the *Init* state to the *Homepage* state:

$$< Init \rightarrow Homepage >$$

We now consider the second row in the log. In this case, the engine notices that the IP corresponds to the IP already seen in the former row and thus identifies a known user, and sets the initial state to be equal to the destination of the transition associated to the last occurrence of such IP (*Homepage*). In this case the request matches the regular expression associated to the label $NewCarsPage$ and thus the engine infers the following transition:

$$< Homepage \rightarrow NewCarsPage >$$

The engine processes the whole log similarly and infers the following transitions from the Listing 1:

$$< Init \rightarrow Homepage >$$
$$< Homepage \rightarrow NewCarsPage >$$
$$< NewCarsPage \rightarrow NewCarsPage >$$
$$< NewCarsPage \rightarrow NewCar >$$
$$< NewCar \rightarrow NewCarsPage >$$
$$< NewCarsPage \rightarrow NewCar >$$
$$< NewCar \rightarrow Contacts >$$

From the sequence of the transitions generated so far, we may recognise the navigational behaviour of the user that we previously described in plain text.

The inference engine uses the sequence of transitions generated from the log to update two sets of counters: a set of counters $c_{i,j}$ for each pair of states $\langle s_i, s_j \rangle \in S \times S$, and a set of counters t_i for each state $s_i \in S$. The inference engine increments the counter $c_{i,j}$ for every transition from state s_i to s_j, the counter t_i for every transition with source state equal to s_i, independently from its destination state. The counter t_i represents the number of times users navigated from state i towards any other state, while counter $c_{i,j}$ represents the number of times users specifically navigated from state s_i to state s_j.

The inference engine uses the counters to estimate the transition probability $\mathbf{P}(s_i, s_j)$ from state s_i to state s_j by simply computing the following frequency:

$$\mathbf{P}(s_i, s_j) = \frac{c_{i,j}}{t_i}$$

The inference engine computes the probability for every possible pair of states s_i and s_j. The transition probability $\mathbf{P}(s_i, s_j)$ estimated as the ratio between the transitions observed from s_i to s_j and the total amount of transitions observed from state s_i correspond to the *maximum likelihood estimator* [9] for $\mathbf{P}(s_i, s_j)$. We exemplify this process referring to the working example. Let us assume to have 45 transitions from the state $Homepage$ to the state $NewCarsPage$ and 15 transitions from the state $Homepage$ to the state $UsedCarsPage$. The data of this simplified scenario tell us that users navigate from the homepage to the announcements of new cars three times more likely than to announcements of used cars. By applying our inference rule we have that:

$$\mathbf{P}(Homepage, NewCarsPage) = 0.75$$
$$\mathbf{P}(Homepage, UsedCarsPage) = 0.25$$

The inference engine models also users who leave the system. The expiration of the UserWindow timeout for a certain IP address implies that the user associated to that address left the system. As a consequence a future request from the same address is considered as issued by a different user and thus initially attached to the *Init* state. In this setting, as soon as a the timeout expires, the engine generates a new transition towards the *End* state, from the destination state of the last transition associated to such IP address.

Referring to the Listing 1, we consider the scenario in which the UserWindow expires for IP 1.1.1.1. In this case the Inference Engine generates the transition:

$$< Contacts \rightarrow End >$$

to model the terminated interaction between the user associated to the IP address 1.1.1.1 and the system. Such transition contributes to updating the counters as previously explained.

Given the set of states S defined by labels plus the additional states *Init* and *End* together with the transition probabilities inferred by the engine, we now have all the ingredients to generate a complete DTMC that represents the navigational behaviour of the users. Figure 2 illustrates a possible model for the mycardealer.com example. For the sake of readability, the figure does not report the transitions from each state to the *End* state. Such transitions can be easily deduced by comparing the sum of the probabilities of the transitions exiting a state to value one. If we consider for example the state *NewCar* that has two outgoing transitions associated with probabilities 0.4 and 0.15, respectively, we can deduce that the probability of the transition from state *NewCar* towards the *End* state has a probability 0.45.

It is important to notice how the DTMC probabilistically captures the navigational behaviour of user. From the model, we can easily deduce that the probability that a user who is

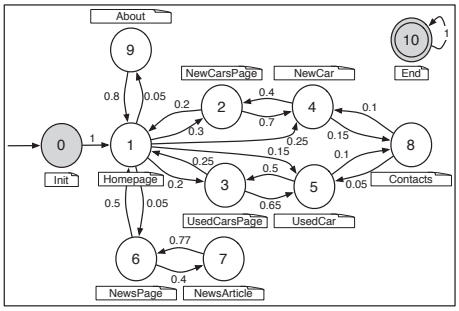

Fig. 2. An example of DTMC inferred for the mycardealer.com application.

browsing an announcement of a new car (state 4) decides to contact a sales agency is equal to 0.15 (transition from state 4 to state 8). The model brings two fundamental advantages: allows the computation of probabilistic properties much more complex than the one just exemplified, support the automatic analysis of the model as illustrated in the next step of the approach. Before illustrating the analysis of the model, we discuss in details the specification of the properties that have been postponed for the sake of readability.

C. Specifying the properties

We now discuss how the application owners specify the properties to be verified at runtime to trigger suitable adaptation strategies. This step, together with the state identification step, is performed at design time. The application owners shall specify the properties of interest in probabilistic computational tree logic (PCTL) [6], a formalism suitable for PMC. Let us consider for example that owners are interested in measuring the probability that users search for new cars instead of used ones. The application owners may later use these values to customise the homepage sorting the announcements of new and used cars to maximise the likelihood of capturing the attention of the users. The more the users are interested in buying new cars, the more the homepage shall include announcements concerning new cars. Notice that these values are difficult to obtain a priori, and may change over time, for instance, because of seasonal or economic trends that may impact on the users attitude. Such values represent a good example of user preferences that can be detected only dynamically by an adaptive Web application. The probability that users browse announcements of new cars versus the probability that users browse announcements of used cars (notice that the two probabilities may not sum to one since users may browse announcements of new as well as used cars) may be computed relying on PCTL and its *finally* operator (F), as shown by the following queries **P1** and **P2**:

$$\textbf{P1} : P =?[F(NewCarsPage \mid NewCar)]$$
$$\textbf{P2} : P =?[F(UsedCarsPage \mid UsedCar)]$$

Since we use PRISM [10] as PMC engine, we used the PRISM syntax to express the properties. The first formula computes the probability of reaching a state labeled either *NewCarsPage* or *NewCar* that indicate a request to a URL corresponding to the page that lists new cars on sale. The second formula is similar but focuses on used cars. The formulae are defined only over the states identified by labels in the state identification step. Engineers may express them at design time without knowing the structure of the DTMC that is instead inferred at runtime in the model inference step. In addition, engineers can express such properties also in structured english that can be automatically translated into PCTL properties [11]. They can be completely agnostic of the complexity of the formal and sounds tools such as PMC and PCTL used in the proposed approach.

```
@Adaptation
public static void reorganizeHomepage(AnalysisResults results){
  float p1Value = results.getPropertyValue("P1");
  float p2Value = results.getPropertyValue("P2");
  if(p1Value>p2Value){
        //put proportionally more announcements
        //of new car in the homepage
  }
  else{
        //put proportionally more announcements
        //of used in the homepage
  }
}
```

Listing 2. Adaptation Script

D. Analysing the model

The Inference Engine periodically invokes a PMC engine with two input parameters: the inferred model and the PCTL properties as specified by the application owner at design time. As previously stated, we use PRISM [10] as a PMC engine, but the approach works also with other engines, such as MRMC [12], and ad-hoc mechanisms conceived for efficient runtime analyses, like the approach introduced by Filieri et al. [13]. PRISM quantitatively evaluates the PCTL properties and produces either a numerical or a boolean result, depending on the property. For example, if we consider the properties specified in the previous paragraph and the DTMC shown in Figure 2, PRISM evaluates the query **P1** to 0.674, while **P2** to 0.437. The results of the PRISM evaluation are used in the adaptation step explained hereafter.

E. Adaptating the application

The adaptation step closes the run-time feedback loop of the approach as illustrated in Figure 1. The inference engine requires two input parameters: a set of regular expression (mapped to labels) and the PCTL properties to be verified. At start-up time engineers also need to define a set of Adaptation Scripts in the form of annotated Java methods. The inference engine invokes such methods depending on the results obtained from the PMC engine. The application owners may exploit this mechanism to implement the desired adaptation strategies. Referring to our working example and to **P1**-**P2**, application owners may write an adaptation script (see Listing 2) that organises the homepage of the mycardealer.com to show a proportional amount of announcements of used versus new cars compliant with the users preferences as estimated over time from the inferred model. By defining ad-hoc regular expressions and by relying on PCTL, engineers may specify several properties and adaptation scripts to be used to adapt the application at runtime. Let us consider for example the scenario in which we apply a discount on a specific set of used cars depending on the current interest of the users. We simply capture the URLs of the announcements of the used cars with a suitable regular expression, and write a PCTL query that monitors the

probability that users browse these announcements. We can couple this property with an adaptation script that dynamically adapts the discount displayed in the announcement to associate a low discount to frequently browsed cars and a high one to cars that users look at rarely. In this way, cars that attract a lot of users are associated with a progressively decreasing discount and vice-versa. Notice that even if we explained the approach by relying on a simple application, the potential of adaptations is extremely effective because of the flexibility and expressiveness of the formal tool and logic we adopted. It is important to notice that the proposed solution brings two fundamental advantages: *incrementality* and *retroactivity* as discussed hereafter. Incrementality is guaranteed by the specific inference mechanism in which, for each transition of the DTMC, we incrementally process log lines by updating the counters $c_{i,j}$ and t_i. Given these counters it is possible to synthesise transition probabilities periodically by performing a simple division among them. This incremental process yields to a very efficient approach that does not interfere with the normal operation of the system. Retroactivity is guaranteed by the continuous process of logging performed by Web servers. Indeed, by exploiting log files, it is possible to apply the proposed solution seamlessly to newly deployed REST systems as well as to legacy systems already in operation.

V. Prototype implementation

To experimentally validate the new approach, we developed a prototype implementation of the core mechanisms developed in Java obtaining promising initial results. The current implementation addresses Apache Web servers [14], and is configurable to support log files generated by other Web servers. To investigate the concrete applicability of the approach, we experimented the inference engine on the log file of an existing publicly available REST application. The experiment indicates that the inference mechanism is extremely efficient, being able to process a log file of more than 40.000 rows with 18.500 matches to regular expressions (state identification step) in less than 10 seconds. The prototype tool was able to process up to 5.000 rows per second on a common laptop[3] generating a model of 17 states. It is important to notice that the incrementally of the inference process implies that log entries may be processed as soon as they are appended in the log file. This feature guarantees the scalability of the approach to large and complex Web applications. We believe in scientific research supported by publicly available tools. As a consequence, the prototype implementation of the inference mechanism has been released as an open source software[4].

VI. Related Work

The problem of capturing and analysing the user behaviours in Web applications through the analysis of log files has been address by many approaches as reported in the survey of Facca et al. [15]. For example, Liu and V. Kešelj combine the analysis of Web server logs with the contents of the requested Web pages to predict users future requests [16]. They capture the content of Web pages by extracting character N-grams that are combined with the data extracted from the log files. Alternatively, Schechter et al. use a tree-based data structure to represent the collection of paths inferred from the log file and predict the next page access [17]. Finally, it is important to mention that Markov models have been used also in other domains with promising results such as software testing. For example Mariani et al. [18] infer finite state machines via dynamic analysis to diagnose integration faults. However Markov models are the most commonly adopted framework to represent users' navigation. Indeed, such models provide an approximate abstraction of user behaviours that, when applicable in a given domain [19], balance well complexity and expressiveness. Borges and Levene propose a Markov model for representing user navigation sessions inferred from log files and modelled with hypertext probabilistic grammars whose higher probability strings correspond to the users navigation patterns [20], [21], [22]. Sarukkai relies on Markov chains for link prediction and path analysis by using an inference mechanism similar to the one proposed in this paper [23]. From a model-driven perspective, we can mention [24] and [25]. The former focuses on functional adaptations and investigates the adoption of aspect oriented programming to weave new system configurations together with run-time models used to validate them. The latter also exploits aspect oriented programming, but focuses on non-functional properties of systems. It describes a middleware for run-time adaptation that chooses between alternative application variants. None of the solutions investigated so far explicitly focus on adaptivity of Web applications. Indeed none of them support the specification and verification of properties combined with any adaptation mechanisms. Differently from existing work, we allow the specification properties by means of a formal logic (PCTL), support their runtime verification by means of PMC, and provide a mechanism that uses the analysis result to trigger adaptation mechanisms implemented as code snippets. These peculiarities allow the design and implementation of user-aware adaptive Web systems and support the enhancement of existing Web applications with adaptive capabilities by simply requiring the availability of log files. Finally, this paper builds on top of a long running research stream on adaptive system. For example in [26], [27] we conceived a framework for run-time adaptation based on probabilistic model checking, while in [13] we refined the previous approach with more efficient model checking techniques conceived ad-hoc.

VII. Conclusions and Future Work

We presented an approach for adaptive REST applications. The approach infers a model of users navigation choices from the log files of the Web application under analysis. We use probabilistic model checking to analyse the model at runtime, and we use the analysis results to trigger adaptation strategies, thus shaping a feedback control loop among model and running implementation. The paper lays the foundations of a novel model-based approach for adaptive Web systems specifically conceived for REST architectures, and represents an early attempt to bring into practice probabilistic model checking, which so far has raised mostly theoretical interest. We are currently refining the approach, experimenting on log files of existing REST systems, and extending the approach to infer other models from the log files to capture different aspects of the users' navigation.

[3] 2 GHz Intel Core i7, 8GB RAM at 1600 MHz DDR3. Java(TM) SE 1.6.0.
[4] http://giordano.webfactional.com/?page_id=22

ACKNOWLEDGMENTS

This research has been funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom and FP7-PEOPLE-2011-IEF, Project 302648-RunMore.

REFERENCES

[1] S. Pertet and P. Narasimhan, "Causes of failure in web applications (cmu-pdl-05-109)," *Parallel Data Laboratory*, p. 48, 2005.

[2] R. Fielding and R. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[4] "Google Analytics," http://www.google.com/intl/en/analytics/, [Online; accessed 19-July-2012].

[5] E. Wilde and C. Pautasso, *REST: From Research to Practice*. Springer, 2011.

[6] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, 1994.

[7] S. Ross, *Stochastic Processes*. Wiley New York, 1996.

[8] J. Friedl, *Mastering regular expressions*. O'Reilly Media, Incorporated, 2006.

[9] M. DeGroot and M. Schervish, *Probability and Statistics-International Edition*. Addison-Wesley. Publishing. Company., Reading, Massachusetts, 2001.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[11] L. Grunske, "Specification patterns for probabilistic quality properties," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 31–40.

[12] J. Katoen, M. Khattri, and I. Zapreevt, "A markov reward model checker," in *Quantitative Evaluation of Systems, 2005. Second International Conference on the*. IEEE, 2005, pp. 243–244.

[13] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 341–350.

[14] "Apache HTTP Server Project," http://httpd.apache.org, [Online; accessed 19-July-2012].

[15] F. Facca and P. Lanzi, "Mining interesting knowledge from weblogs: a survey," *Data & Knowledge Engineering*, vol. 53, no. 3, pp. 225–241, 2005.

[16] H. Liu and V. Kešelj, "Combined mining of web server logs and web contents for classifying user navigation patterns and predicting users future requests," *Data & Knowledge Engineering*, vol. 61, no. 2, pp. 304–330, 2007.

[17] S. Schechter, M. Krishnan, and M. Smith, "Using path profiles to predict http requests," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 457–467, 1998.

[18] L. Mariani, F. Pastore, and M. Pezzè, "Dynamic analysis for diagnosing integration faults," *Software Engineering, IEEE Transactions on*, vol. 37, no. 4, pp. 486–508, 2011.

[19] F. Chierichetti, R. Kumar, P. Raghavan, and T. Sarlós, "Are web users really markovian?" in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 609–618.

[20] J. Borges and M. Levene, "Data mining of user navigation patterns," *Web usage analysis and user profiling*, pp. 92–112, 2000.

[21] ——, "Generating dynamic higher-order markov models in web usage mining," *Knowledge Discovery in Databases: PKDD 2005*, pp. 34–45, 2005.

[22] ——, "Evaluating variable-length markov chain models for analysis of user web navigation sessions," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 4, pp. 441–452, 2007.

[23] R. Sarukkai, "Link prediction and path analysis using markov chains," *Computer Networks*, vol. 33, no. 1, pp. 377–386, 2000.

[24] B. Morin, O. Barais, G. Nain, and J. Jezequel, "Taming dynamically adaptive systems using models and aspects," in *ICSE*. IEEE Computer Society, 2009, pp. 122–132.

[25] S. Lundesgaard, A. Solberg, J. Oldevik, R. France, J. Aagedal, and F. Eliassen, "Construction and execution of adaptable applications using an aspect-oriented and model driven approach," in *Distributed Applications and Interoperable Systems*. Springer, 2007, pp. 76–89.

[26] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 111–121.

[27] C. Ghezzi and G. Tamburrelli, "Predicting performance properties for open systems with kami," *Architectures for Adaptive Software Systems*, pp. 70–85, 2009.