Formal Modeling of RESTful Systems Using Finite-State Machines

Ivan Zuzak, Ivan Budiselic, and Goran Delac

School of Electrical Engineering and Computing, University of Zagreb,
Unska 3, 10000 Zagreb, Croatia
{izuzak,ibudiselic,gdelach}@gmail.com

Abstract. Representational State Transfer (REST), as an architectural style for distributed hypermedia systems, enables scalable operation of the World Wide Web (WWW) and is the foundation for its future evolution. However, although described over 10 years ago, no comprehensive formal model for representing RESTful systems exists. The lack of a formal model has hindered understanding of the REST architectural style and the WWW architecture, consequently limiting Web engineering advancement. In this paper we present a model of RESTful systems based on a finite-state machine formalism. We show that the model enables intuitive formalization of many REST's constraints, including uniform interface, stateless client-server operation, and code-on-demand execution. We describe the model's mapping to a system-level view of operation and apply the model to an example Web application. Finally, we outline benefits of the model, ranging from better understanding of REST to designing frameworks for RESTful system development.

Keywords: representational state transfer, World Wide Web, software architectural styles, formal model, finite-state machines, hypermedia.

1 Introduction

One of the main reasons for the wide adoption of the World Wide Web (WWW) as a global information system has been its ability to scale and remain reliable with the rapid growth in the number of its users and applications. Enabling this growth is an architecture [1] designed just for the purpose of developing large-scale distributed hypermedia systems such as the WWW. The foundation of this architecture is a set of software design principles named the *Representational State Transfer* architectural style (REST) [2]. In a way, REST describes how a Web application should behave in order to maximize beneficial properties, such as simplicity, evolvability, and performance.

From its introduction in year 2000., REST has not only guided many incremental changes in WWW's continuous evolution, such as the recently standardized HTTP PATCH method [3], but has also been guiding the development of its new dimensions in order to preserve its desirable properties. These efforts include the expansion of the WWW with higher-level applications, interlinked

S. Auer, O. Diaz, and G.A. Papadopoulos (Eds.): ICWE 2011, LNCS 6757, pp. 346–360, 2011.

data, physical devices and real-time access, through mashups [7], the Semantic Web [5], Web of Things [4] and the Real-Time Web [6]. However, as the WWW grows in functionality, it also grows in complexity and is consequently becoming harder to understand and explain at the architectural level. Therefore, understanding REST is essential for engineering the WWW and its future.

However, although defined over 10 years ago, REST's architectural principles have only been semi-formally described using diagrams, tabular techniques and natural language descriptions. Furthermore, although formal models of hypermedia systems in general do exist [9], no such model covers fundamental principles of REST and most techniques are used to model the WWW which includes many unRESTful properties. In result, no formalism for modeling RESTful systems exists today. This lack of formal explanation has increasingly been causing negative effects, such as misunderstanding of REST concepts, misuse of terminology [10] and ignorance of benefits of the REST style. For example, common misunderstandings include the overload in meaning of the word state, such as state, application state, resource state and session state [13], and identifying functionality of REST user agents with Web browsers [14].

In result, WWW researchers and engineers experience difficulty in concisely explaining both small-scale and large-scale WWW patterns or requirements, such as defining Web application interaction [11] and defining future WWW architectural goals [12]. Furthermore, development of systems which adhere to the REST style is difficult due to a lack of software frameworks which guide their implementation [15]. This is especially true for developing machine-driven RESTful clients and their application in machine-to-machine RESTful interaction and service composition [8]. We believe this to be a direct consequence of the absence of formal models which are used as the practical encoding of general architectural principles and serve as the foundation for the software development process in such frameworks.

In this paper we present a finite-state machine (FSM) [16] formalism for modeling RESTful systems, with the primary motivation of contributing to the understanding of the REST style. Our choice of using a FSM formalism was inspired by *The Rule of Least Power* [17] which originally suggests that the least powerful language suitable for expressing constraints or solving a problem should be chosen. Consequently, one of our goals was to explore the possible limits of the FSM formalism for this specific purpose in order to suggest the use of more a powerful model. Furthermore, one of the core principles of the REST style, that resource representation transfers are used for transitioning agents from one state to another, suggest the usage of a state transition system formalism.

Our model is based on the nondeterministic finite-state machine formalism with epsilon transitions (ε-NFA). We first explain the mapping of the model's abstract elements to those of a RESTful system. In order to illustrate model usage, we introduce an example Web application and present its ε-NFA model. Next, we explain how each of REST's style constraints map to the model, including uniform interface, stateless client-server operation, and code-on-demand execution. We show that the transition function of the ε-NFA enables formalization

of the transformation of the system's application state, following the *hypermedia as the engine of application state* principle. Furthermore, we show that nondetermistic transitions of the model enable formalization of the temporally-varying mapping of resources to representations and that ε-transitions enable formalization of code-on-demand execution. The presented model naturally translates to the client-centric view of RESTful system operation with the client storing application state, issuing resource manipulation requests and integrating responses into application state, while the server performs request processing.

The remainder of the paper is organized as follows. In Section 2, we give an overview of related work, focusing on approaches for formalization of RESTful systems, hypermedia systems and Web applications. Section 3 defines our finite-state machine formalism for modeling RESTful systems and presents the model of an example Web application. In Section 4 we conclude the paper, discuss the limitations of the presented model and give directions for future work.

2 Related Work

This Section gives an overview of existing approaches for modeling RESTful systems with the goal of examining the degree of completeness in which REST principles are covered by each model. We first give an overview of related work focused on REST and similar styles and then of related work focused on modeling Web applications and hypermedia systems in general. In summary, our analysis shows several issues with existing formal and semi-formal models that motivate our research. First, most models are not focused on REST, rather on hypermedia applications in general or Web applications, and thus do not include many of REST's principles. Second, most models do not offer an explicit mapping from REST's principles to the chosen formalism, and in general do not use the terminology originally proposed for REST. Third, most models address only REST's static properties or do not offer a mapping of REST principles to a system-level view of operation dynamics. Fourth, some principles of REST are rarely included in models, such as the temporally-varying mapping of resources to representations, code-on-demand execution and steady application states.

In [18], the authors present Alfa, a framework for characterization of architectural styles, based on composing a small set of architectural primitives. The authors use Alfa to describe many architectural styles, including a subset of REST, the layered-client-code-on-demand-cache-stateless-server (LCCOD$SS) style. However, this style does not include the uniform interface constraint, one of the most important and distinct principles of REST, while its model does not explain key REST concepts, such as resources, representations and media types.

In [19] the authors present a definition of RESTful semantic Web Services using a process calculus formalism. In this model, a RESTful system is described as a set of processes, representing origin servers and user agents, which exchange request and response messages over uniquely identified channels. This approach is very promising as it allows that channel names exchanged between processes be used to model the exchange of messages containing resource identifiers. This

property of the model enables formalization of the REST constraint of using hypermedia links as the engine of application state. We encourage further work on using process algebras for modeling RESTful systems which would include a mapping of resources, representations, media types, steady and transition states to such a model together with a generalization of the model which would not be bound to standard HTTP methods as it currently is.

In [20] the authors present a promising formal model for specifying RESTful execution of processes specified by Service nets, a specific class of Petri nets that include value passing. The main advantage of the model is its integration of hypermedia-driven application flow while its main use is in modeling composition of RESTful processes. However, the model is not explicit on where application state is stored and does not explain its transformation in response to initial fetches of resource representations which occur at the beginning of an application flow. Furthermore, the model introduces a notion of static and dynamic ports, metaphors for static and temporary resources, which is not RESTful since clients never know and do not need to know if a resource is static or temporary.

In [21] the authors present the Resource Linking Language, an XML-based language for describing interlinked REST resources and consequently the service that can be accessed by interacting with those resources. The language is based on a RESTful service description metamodel, formalized as a UML class diagram, and which incorporates many REST's concepts, such as resources, representations, media types and links. However, the static metamodel does not explicitly express the important dynamic properties of REST, such as the application state contained on the client side and the effects of the code-on-demand constraint, and does not map REST's concepts to a client-server architecture.

In [22], an agent-based model of RESTful applications is presented. In the model, an agent represents the client side of the application while the environment represents the server side. The agent has several pools of predefined logic, including application, action and protocol logic, formalized as a hierarchical state machine which drives the agent's action selection. Although explaining high-level dynamics of a RESTful system, the model is more descriptive than formal, not providing an explicit mapping of many REST's principles to the model, including code-on-demand execution, resource representations and temporally-varying mapping of resources to representations.

In [23], the authors present a broad overview of modeling methods for Web application verification and testing, using a categorization of criteria for classifying models of Web application. Although some criteria may be regarded as reflections of REST's principles, such as the *dynamic navigation* criterion which asserts the possibility of modeling servers that may nondeterministically return responses for the same requests, this work is focused on Web applications only and most principles of REST were not considered. For example, in [25], the authors introduce a finite-state machine behavioral modeling approach for hypermedia Web applications. The model is based on presenting Web pages as states and links as transitions in a FSM. Furthermore, the authors define multiple types of pages and transitions in order to model activity-initiated transitions and

automatic transitions. However, the model is based on a deterministic FSM and does not explain the temporally-varying mapping of resources to representations in RESTful systems. Furthermore, the model is based on using only "clickable links" for transitions, i.e. only navigation is used for changing application state, which is an incomplete definition of transitions in RESTful systems.

In [9], the authors give a broad systematization of formal and semi-formal reference models for hypermedia systems and a comparison of hypermedia engineering methodologies. Hypermedia reference models capture important abstractions found in hypermedia applications and describe the basic concepts of these systems, such as the node/link structure. Semi-formal models include the Amsterdam Hypermedia model while formal models include the Trellis and Dexter reference models. However, although these models describe the mechanisms by which the links and nodes in the hypermedia network are related, these do not include many principles, concepts and terminology of RESTful systems. For example, the Dexter reference model uses components and instances, while REST uses resources, representations and application state. Furthermore, these models do not offer a dynamic operational system-level view which maps system components to clients, servers and intermediaries.

In [24] the authors present an automata-based model of hyperdocuments with the goal of verifying trace-based properties by model checking. The model is focused on simple hyperlink-based connectedness of hypertext documents for the Trellis hypermedia system, which does not include important properties of RESTful systems, such as the uniform interface constraint. However, two interesting ideas are presented. First, the underlying model upon which a link automaton is constructed is based on place/transition nets in order to allow modeling of parallel execution of hyperdocuments. Second, the authors present a temporal logic for model checking link automatons.

3 A Finite-State Machine Model of RESTful Systems

Finite-state machines (FSMs) are a mathematical formalism for describing processes with a finite number of possible states and sequential state transitions. Although components of a RESTful system may be viewed as separate agents, each driven by a self-contained FSM, we model the operation of the complete system, often called an *application*, as a single FSM. For formalizing RESTful systems, we use a nondeterministic finite-state machine with ε-transitions (ε-NFA), an extension of the basic deterministic FSM model. Our model is focused more on explaining the operation of RESTful systems and less on explaining their static properties. The central part of this view is the application state of a RESTful system, its definition, transformation and relation to other concepts.

In the following subsections we first give an overview of the model, formalizing elements of the ε-NFA in context of RESTful systems. Second, in order to illustrate the usage of the model, we introduce an example Web application and present its ε-NFA formalization. Next, we describe the model in more detail by mapping style constraints of REST to the presented model, including

client-server, stateless, code on demand and uniform interface styles. The presented model does not explicitly formalize the layered and cacheable constraints of REST since these are not essential for understanding the operation of a system from a functional perspective.

3.1 Model Overview

A nondeterministic finite-state machine with epsilon transitions (ε-NFA) is a tuple $(S, \Sigma, s_0, \delta, F)$ where S is a finite, non-empty set of states, Σ is a finite, non-empty set of symbols representing the input alphabet, $s_0 \in S$ is the initial state of the ε-NFA, δ is the state transition function $\delta : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(S)$, where $\mathcal{P}(S)$ is the power set of S, and $F \subseteq S$ is the set of accepting states. A system-level perspective of ε-NFA operation is shown in Fig. 1. The *Input Symbol Generator* module generates an input symbol based on internal rules or environment state (1). Since the generator is not part of the ε-NFA's formal model but is required to properly model its operation, we define that the generator has access to the system's state stored in the *Current State* module. The *Transition Function* accepts the generated input symbol and the current state (2), determines the next state and stores it in the Current State module (3). The described cycle is then repeated. Since the ε-NFA is nondeterministic, formally the Transition Function module returns a set of states, for which the practical meaning is that the system may be in any single state from that set. If at some point at least one state stored in the Current State module is marked as accepting, it is said that the ε-NFA accepts the sequence of input symbols read up to that point. Furthermore, since the ε-NFA includes ε-transitions, the Transition Function does not need to read an input symbol in order to perform some transitions.

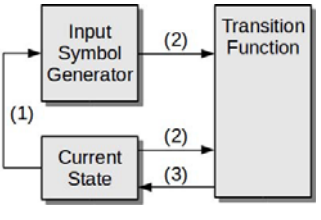

Fig. 1. System-level view of ε-NFA operation

We map a RESTful system [2] to the ε-NFA formal model as follows. Let $ResIDs$ be a finite set of identifiers of system resources, let $Metas$ be a finite set of metadata key-value pairs, let $LTypes$ be a finite set of link types and let $LRels$ be a finite set of link relations. Furthermore, let $Links$ be the finite set of hypermedia links, each defined by a resource identifier, link type and link relation $Links \subseteq ResIDs \times LTypes \times LRels$ and let $MTypes$ be a finite set of media types which determine the set of hypermedia links present in a representation $MTypes = \{MType : Reprs \rightarrow \mathcal{P}(Links)\}$. Finally, let $Reprs$ be the finite

set of resource representations, each consisting of resource metadata and data $Reprs \subseteq data \times \mathcal{P}(Metas)$ where one metadata element defines the media type of the representation. The **set of states S of the ε-NFA** represents the application states of the system, $S = AppStates$, where an application state is defined as a non-empty, ordered set of representations, $AppStates \subseteq \mathcal{P}(Reprs) - \{\}$. Furthermore, **the initial state s_0 of the ε-NFA** represents the initial application state at system startup. Finally, let $Steadys$ be the subset of application states, $Steadys \subseteq AppStates$, for which it holds true that representations of all embeddable resources linked to from the first representation in the application state are also present in the application state. The **set of accepting states F of the ε-NFA** represents the steady application states, $F = Steadys$.

Next, let Ops be a finite set of resource manipulation methods, and let $Reqs$ be the finite set of valid resource manipulation requests $Reqs \subseteq Ops \times ResIDs \times Reprs$. The **set of input symbols Σ of the ε-NFA** represents requests $Reqs$ and their corresponding link types $LTypes$ for manipulating resources, $\Sigma \subseteq Reqs \times LTypes$. Furthermore, let $Resrcs$ be a finite set of resources, mappings from a resource identifier to a representation $Resrcs : \{Resrc : ResIDs \rightarrow Reprs\}$, and $Resps$ be a finite set of resource manipulation responses $Resps \subseteq ResIds \times Reprs$. The **transition function δ of the ε-NFA** represents the translation of input symbols into requests, processing of requests into responses and integration of response representations into the next application state, $\delta : AppStates \times (Reqs \times LTypes) \rightarrow \mathcal{P}(AppStates)$. Furthermore, since the ε-NFA includes ε-transitions, for some application states the transition function may change the application state without reading an input symbol, i.e. without the system generating a resource manipulation request.

3.2 Example Web Application

In order to illustrate concepts presented in this paper, we introduce a weather forecast Web application. The resources comprising the Web application are shown in Fig. 2. The base URI of the application is `http://weather.example.com` and we identify its resources using relative addressing. The main Web page, located at `/main`, contains an `<a>` link to the details Web page and a `<script>` element pointing to a JavaScript script located at `/script`. The script periodically highlights the `<a>` link to the details page by changing the color from blue to red. The details Web page, located at `/details`, contains an `<img>` tag pointing either to `/cloudy` or to `/sunny` depending on the current weather. Furthermore, the details page contains a `<script>` element with inline JavaScript code which uses XmlHttpRequest for periodically fetching the current temperature from `/temp` and displaying it in the Web page. Finally, the details page contains an `<a>` link pointing to the main page. The media types of the resources are `text/html` for `/main` and `/details`, `image/png` for the `/sunny` and `/cloudy`, and `text/javascript` and `text/plain` for `/script` and `/temp`, respectively. We assume that the user of the application is using a modern Web browser.

The ε-NFA model of the example Web application is shown in Fig. 3, with numbers denoting states and letters denoting input symbols. The initial state

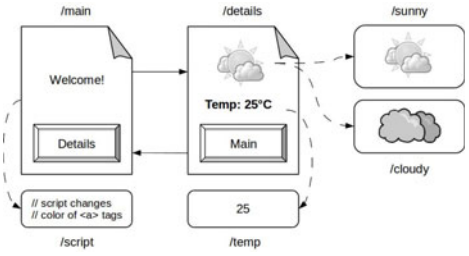

Fig. 2. Example Web application

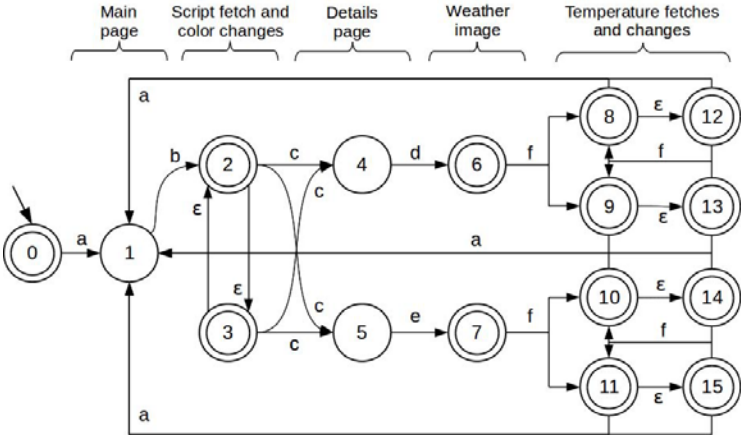

Fig. 3. ε-NFA model of example Web application

`0` contains a representation with an `<a>` link to the `/main` page. After a GET request is issued (`a`), the server returns a response containing the representation of the `/main` page which becomes the current state. State `1` is not steady since the representation contains a `<script>` link to `/script` which must be fetched. A GET request is issued to fetch the script (`b`) and the application then enters the steady state `2`. Because the script periodically changes the color of the link pointing to the `/details` page, the application's steady state may change between states `2` and `3` without issuing a request (ε).

When the user follows the link to the `/details` page (c), the application makes a nondeterministic transition to transient states `4` and `5` because the representation contains an `<img>` link pointing either to `/sunny` or `/cloudy`. After fetching the linked image using a GET request (`d` or `e`), the application enters a steady state `6` or `7`. Furthermore, the representation of the details page contains an inline script which periodically makes a GET request for the current temperature (`f`). Since the returned temperature may have two possible values (25C or 30C), the application nondeterministically enters states `8` and `9` if the weather was cloudy, or states `10` or `11` if the weather was sunny. The script

then inserts the temperature into the page without issuing requests (ε), moving the application into states `12` and `13`, or `14` and `15`, depending on the weather. Because the details page contains a link to the main page, the user may at any time follow the link (`a`) and bring the application back to state `1`.

3.3 Client-Server Style and Stateless Style Constraints

Figure 4 shows the system-level view of a RESTful system as a set of modules, their mapping to the elements of the ε-NFA and distribution between client and server components. Because client-server interaction in RESTful systems must be stateless, the *Application State* module which stores the current application state is located on the client. This is also true for modules that generate input symbols: the *Media Type Processor*, *Application-level Logic* and *Hypermedia-level Logic*. However, the transition function is divided between the client and server in order to satisfy the client-centric description of RESTful system operation in which the server is responsible only for mapping from requests to responses. Therefore, the *Request Preprocessor*, *State Integrator* and *Code-on-demand Engine* reside on the client, and only the *Request Processor* on the server.

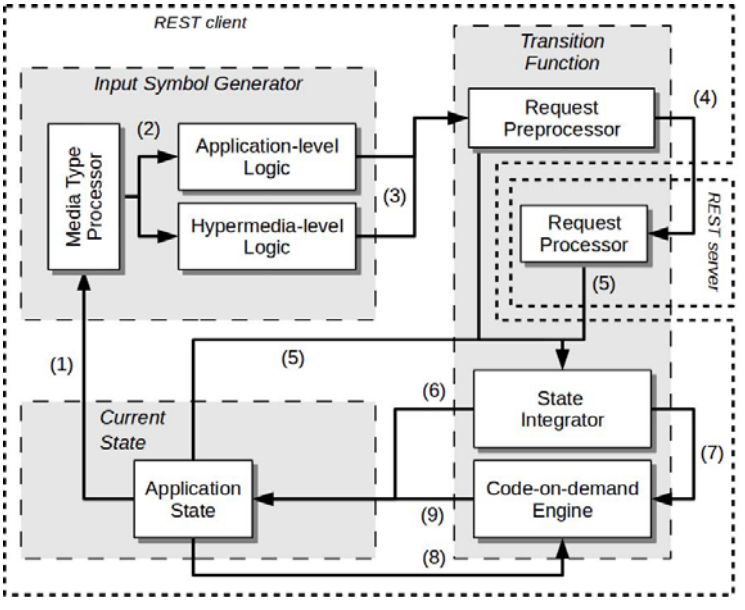

Fig. 4. Mapping of client-server and stateless REST constraints to the ε-NFA model

The interaction of the system's modules is defined as follows. The resource representations comprising the current application state are read by the Media Type Processor (1) in order to determine the set of available hypermedia links. For example, the `/main` page of the example Web application has the

`text/html` media type which enables that the `<a>` link to the `/details` page and `<script>` link to the `/script` script be recognized. The set of links and application state are passed to the Application-level Logic and Hypermedia-level Logic (2) so that one of the links may be chosen as the basis for the next input symbol. Hypermedia-level Logic is responsible for generating input symbols which guide the system to a steady state. Because steady states are determined exclusively from the media types of the representations in the current application state, Hypermedia-level Logic functions independently of Application-level Logic. On the other hand, Application-level Logic is responsible for generating input symbols based on application-specific goals, which are derived either from user input or from application-specific rules encoded in the module. For example, after the `/main` page has been fetched, two links may be followed, `<script>` for embedding the `/script` script and `<a>` for navigating to the `/details` page. The former link would be selected by Hypermedia-level Logic for downloading the script, while the latter link would be selected by Application-level Logic, but only in response to the user clicking on the link.

The input symbol generated by either of these modules consists of a resource manipulation request and the link type of the chosen link (3). The Request Preprocessor stores and removes the link type of the input symbol and adds a request identifier to the request before forwarding it to the Request Processor on the server (4). The server's response therefore contains the request identifier and a representation of the identified resource. Although request identifiers are a conceptual requirement for coupling responses with requests, they are not currently used on the Web since requests and responses are related through the TCP connection by which they are sent and received. The State Integrator uses the link type connected with the request, the corresponding server response and the current application state (5) to synthesize the next state (6). For example, if the `/main` page was fetched and the `/script` script was fetched afterwards via the `<script>` link, the script representation would be added to the application state. On the other hand, if the `/details` page was fetched afterwards via the `<a>` link, the received representation would replace the existing application state.

Finally, if the resource representation is an executable script, the integrator passes the script to the Code-on-demand engine for execution (7). The script may then examine and change the current application state (8) (9) without issuing requests to the server. For example, after the `/script` script is fetched and executed using a JavaScript engine, it periodically modifies the application state by changing the color of a link in the representation of the `/main` page.

3.4 Uniform Interface Style Constraint

REST is defined by four *uniform interface* constraints: identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of application state. In this section we formalize these constraints in the context of our model. *Resource identification* is supported in the model through resource identifiers which are used explicitly in input symbols

and application states, where an input symbol consists of a resource manipulation request and link type, where the request contains a resource identifier, a method and a representation. For example, an input symbol $IS_{\text{toDetails}}$ for navigating to `/details` in the Web application example could be represented as:

$$IS_{\text{toDetails}} = (\ \mathit{Request} : (\mathit{Method} : \text{“}\mathit{GET}\text{”}, \mathit{ResourceId} : \text{“}\mathit{/details}\text{”}, \\ \mathit{Representation} : \text{“”}), \mathit{LinkType} = \text{“} < a > \text{”})\ ,$$

and the application state AS_{main} of a completely loaded `/main` page as:

$$AS_{\text{main}} = [(\mathit{metadata} : \text{“...”}, \mathit{data} : \text{“}\mathit{/maincontents}\text{”}), \\ (\mathit{metadata} : \text{“...”}, \mathit{data} : \text{“}\mathit{/scriptcontents}\text{”})]\ .$$

Due to lack of space, we do not include full listings of representation data and metadata. On the Web, metadata in general consists of HTTP headers, while the data is the body of HTTP message.

Manipulation of resources through representations is supported in the model through explicit usage of representations in input symbols and application state. One of REST's foundations is the temporally-varying mapping of resources to representations, which is supported through the nondeterminism of the transition function. For example, because the `/details` page contains an `<img>` link to either `/cloudy` or `/sunny`, the navigation from `/main` to `/details` in the example Web application could be represented with the following transition:

$$\delta(AS_{\text{main}}, IS_{\text{toDetails}}) = \{AS_{\text{detailsCloudy}}, AS_{\text{detailsSunny}}\}\ \text{, where}$$

$$AS_{\text{detailsCloudy}} = [(\mathit{metadata} : [\mathit{mediaType} : \text{“}\mathit{text/html}\text{”}], \\ \mathit{data} : \text{“}\mathit{/details\ content\ with\ link\ to\ /cloudy}\text{”})]\ ,$$

$$AS_{\text{detailsSunny}} = [(\mathit{metadata} : [\mathit{mediaType} : \text{“}\mathit{text/html}\text{”}], \\ \mathit{data} : \text{“}\mathit{/details\ content\ with\ link\ to\ /sunny}\text{”})]\ .$$

The *self-descriptive messages* style constraint is supported in the model through stateless interaction, the limitation of using finite sets for system methods, media types, link types and link relations, and explicitly using these elements in the input symbols and application state. For example, the $IS_{\text{toDetails}}$ input symbol shown above has a link type of `<a>` while the representations in states $AS_{\text{detailsCloudy}}$ and $AS_{\text{detailsSunny}}$ are of the `text/html` media type.

Hypermedia as the engine of application state is supported in the model through the transition function which advances the system from one state to another. Specifically, the output of the transition function should be defined only for pairs of states and input symbols for which the input symbol may be derived from the current state. In other words, the current state must contain a hypermedia link used to generate the next input symbol's link type and resource manipulation request. For example, the transition function in the model of the

example Web application is undefined for state $AS_{\text{detailsCloudy}}$ and $IS_{\text{toDetails}}$ because the `/details` page does not contain an `<a>` link to itself:

$$\delta(AS_{\text{detailsCloudy}}, IS_{\text{toDetails}}) = \{\} \ .$$

Furthermore, we define that the initial application state is a single representation containing links to the resources which are the stable entry points for the system. For example, since the example Web application's entry point is the `/main` resource, the initial state of the model AS_{init} could be represented as:

$$\begin{aligned} AS_{\text{init}} = \ & [(metadata : [mediaType : \text{“}text/html\text{”}], \\ & data : \text{“}HTML\ page\ with\ an\ <a>\ link\ to\ /main\text{”})] \ . \end{aligned}$$

Finally, *steady and transient application states* are supported in the model through accepting and unaccepting states. The acceptance of a state is determined from the media type of the first representation in the application state. For example, in the example Web application the first representation in an application state is always of the `text/html` media type meaning that all embedded resources, such as resources linked to using `<img>` and `<script>`, should be fetched in order for the system to be in a steady state. Therefore, the state AS_{main} is accepting (steady), while the state $AS_{\text{detailsCloudy}}$ is unaccepting (transient).

3.5 Code-on-Demand Style Constraint

The *code-on-demand* style constraint is defined [2] as client-side execution of downloaded scripts together with the possibility that these scripts extend the functionality of the client. We formalize the code-on-demand constraint in the model through ε-transitions i.e. if a script executing on the client may change the application state from A to B without issuing a request, then this change is modeled with an ε-transition as $\delta(A, \varepsilon) = \{B\}$. In the example Web application, the `/script` script changes the color of the `<a>` link of the `/main` page which can be modeled as $\delta([main_{\text{link\_blue}}], \varepsilon) = [main_{\text{link\_red}}]$ and $\delta([main_{\text{link\_red}}], \varepsilon) = [main_{\text{link\_blue}}]$, where $main_{\text{link\_red}}$ and $main_{\text{link\_blue}}$ are the representations of the `/main` page in which the link is colored red and blue, respectively.

4 Conclusion and Future Work

The study of architectural styles is an essential part for understanding and improving information systems. As the World Wide Web is the most important global information system, the study of its foundational architectural style, Representational State Transfer (REST), is of equal importance from both a theoretical and a practical perspective. Our analysis of previous research in this field has shown that formal models of RESTful systems are unresearched, consider only few core principles of RESTful systems while ignoring others and are focused on modeling hypermedia systems in general and not RESTful systems.

In this paper we propose a formalism for modeling RESTful systems based on nondeterministic finite-state machines with epsilon transitions (ε-NFA). We

show that ε-NFAs are a natural fit for modeling REST's principles which are primarily concerned with exchange of representations using states and transitions. Specifically, the states of the ε-NFA represent the application states which the system may be in at some point of execution, where each application state is a set of resource representations. The input symbols of the ε-NFA represent the set of resource manipulation requests while the transition function models the hypermedia links between resources i.e. the request which may be issued at some state. In order to support the time-varying nature of resource representations, transitions may be nondeterministic, while ε-transitions are used to model client-side execution of code-on-demand scripts. The client-server style of REST is therefore naturally modeled by storing the current ε-NFA state and generating input symbols on the client while the transition function is divided between the client and server. The client is responsible for transforming input symbols into requests and integrating resource representations into the application state, while the server is responsible for processing requests into responses. Representation media types determine which states of the ε-NFA are accepting or not, representing respectively steady and transient states of the system. Therefore, with respect to the presented model, a system may be called RESTful if it can be represented with an ε-NFA and if sequences of generated input symbols do not lead the ε-NFA into an empty error state.

Our research gives the following insights and areas for future work. First, although we have shown an example of using the presented formalism in modeling a simple Web application, the formalism should be applied to more systems, including complex Web applications, widget-based Web applications, Web APIs and mashups. A special focus of this effort would include the modeling of composite RESTful applications. However, the goal would not be to extend the model so that it supports modeling of all properties of all systems, but rather to use it for reasoning about which properties of such systems are RESTful.

Second, it would be useful to explore the possibility of extending the presented formalism so that it explicitly accounts for currently unaddressed principles of RESTful systems. For example, this could include the layered and cacheable style constraints, and resources that map to a set of representations with different media types, with the goal of modeling content negotiation.

Third, one specific issue which may be raised is that of the finite set limitations of the presented model. Because RESTful systems are not restricted to a finite number of states or input symbols, finite-state machines are not a completely suitable model. The ε-NFA model may be relaxed so that the set of states may be infinite or even not countable or, alternately, other kinds of models for describing infinite-state machines may be used. One possible candidate are labeled state transition systems [26], a formalism similar to finite-state machines which permits that the number of states and transitions be infinite.

Fourth, we will explore practical applications of the model. One possible direction is to use the ε-NFA model as the basis for designing a framework for development of RESTful systems, and specifically RESTful Web applications.

Existing Web application development frameworks do not support the implementation of many RESTful constraints, such as the uniform interface constraint, shifting that burden to the developer. We believe this to be a direct consequence of nonexisting models with system-level mappings and that the model in this paper is simple, yet powerful, and understandable by developers. This premise is supported by the recently developed Restfulie framework [27] which imposes state transitions as the underlying application development model.

Finally, in order to avoid the state explosion problem for models of complex systems, we will consider aggregating similar states of models into a single state. However, this requires that a suitable method for aggregation be chosen. Currently, we are researching an approach based on abstracting representations into two parts: the constant application-level data and the variable set of hypermedia links. This would enable that states which correspond to the same set of representations with the same set of links and which differ only by the data would be aggregated into a single state, significantly reducing the number of states.

Acknowledgments. The authors acknowledge the support of the Ministry of Science, Education, and Sports of the Republic of Croatia through the *Computing Environments for Ubiquitous Distributed Systems* (036-0362980-1921) research project. Furthermore, the authors thank Sinisa Srbljic, Dejan Skvorc, Miroslav Popovic, Klemo Vladimir, Marin Silic, Jakov Krolo and Zvonimir Pavlic from the School of Electrical Engineering and Computing, University of Zagreb.

References

1. Jacobs, I., Walsh, N.: Architecture of the World Wide Web, Volume One. W3C Recommendation, WWW Consortium (2004), `http://www.w3.org/TR/webarch/`
2. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002)
3. Dusseault, L., Snell, J.: PATCH Method for HTTP. In: Proposed Standard, Internet Engineering Task Force, IETF (2010), `http://tools.ietf.org/html/rfc5789`
4. Trifa, V., Trifa, V., Guinard, D., Bolliger, P., Wieland, S.: Design of a Web-based Distributed Location-Aware Infrastructure for Mobile Devices. In: 1st IEEE International Workshop on the Web of Things, Mannheim, Germany, pp. 714–719 (2010)
5. Alarcon, R., Wilde, E.: Linking Data from RESTful Services. In: 3rd International Workshop on Linked Data on the Web, Raleigh, North Carolina, USA (2010)
6. Fitzpatrick, B., Slatkin, B., Atkins, M.: PubSubHubbub protocol, `http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html`
7. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing 12(5), 24–31 (2008)
8. Pautasso, C.: RESTful Web service composition with BPEL for REST. Data & Knowledge Engineering 68(9), 851–866 (2009)
9. Koch, N.: Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process. Ph.D. dissertation, Ludwig-Maximilians-University of Munich, Germany (2000)

10. Fernandez, F., Navon, J.: Towards a Practical Model to Facilitate Reasoning about REST Extensions and Reuse. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 31–38 (2010)
11. Rees, J.: ACTION-434: Some notes on organizing discussion on WebApps architecture. In: W3C TAG Mailing List (2010), http://lists.w3.org/Archives/Public/www-tag/2010Oct/0061.html
12. ISSUE-60: Web Application State Management. W3C TAG Issues List, http://www.w3.org/2001/tag/group/track/issues/60
13. Fielding, R.T.: ACTION-434: Some notes on organizing discussion on WebApps architecture. In: W3C TAG Mailing List (2010), http://lists.w3.org/Archives/Public/www-tag/2010Oct/0100.html
14. Kemp, J.: AWWW and the Web interaction model. In: W3C TAG Mailing List (2010), http://lists.w3.org/Archives/Public/www-tag/2010Jun/0034
15. Vinoski, S.: RESTful Web Services Development Checklist. IEEE Internet Computing 12(6), 95–96 (2008)
16. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing, Reading (1979)
17. Berners-Lee, T., Mendelsohn, N.: The Rule of Least Power W3C TAG Finding (2006), http://www.w3.org/2001/tag/doc/leastPower.html
18. Mehta, N.R.: Composing style-based software architectures from architectural primitives. Ph.D. dissertation, University of Southern California, California, USA (2004)
19. Hernandez, A.G., Moreno Garcia, M.N.: A Formal Definition of RESTful Semantic Web Services. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 39–45 (2010)
20. Decker, G., Luders, A., Overdick, H., Schlichting, K., Weske, M.: RESTful Petri Net Execution. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 73–87. Springer, Heidelberg (2009)
21. Alarcon, R., Wilde, E., Bellido, J.: Hypermedia-driven RESTful Service Composition. In: 6th Workshop on Engineering Service-Oriented Applications, San Francisco, California (2010)
22. Charlton, S.: Building a RESTful Hypermedia Agent, Part 1 (2010), http://www.stucharlton.com/blog/archives/2010/03/building-a-restful-hypermedia
23. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modeling methods for web application verification and testing: state of the art. In: Software Testing, Verification & Reliability Archive, vol. 19(4), pp. 265–296. John Wiley and Sons Ltd., Chichester (2009)
24. Stotts, P.D., Furuta, R., Cabarrus, C.R.: Hyperdocuments as Automata: Verification of Trace-Based Properties by Model Checking. ACM Transactions on Information Systems 16(1), 1–30 (1998)
25. Dargham, J., Al-Nasrawi, S.: FSM Behavioral Modeling Approach for Hypermedia Web Applications: FBM-HWA Approach. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, pp. 199–199 (2006)
26. Trybulec, M.: Labelled State Transition Systems. Formalized Mathematics 17(2), 163–171 (2009)
27. Parastatidis, S., Parastatidis, S., Webber, J., Silveira, G., Robinson, I.S.: The role of hypermedia in distributed system development. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 16–22 (2010)