DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No.: DSIC-II/06/07 **Pages: 16**
Title: Formal Verification of Websites
Author(s): Sonia Flores, Salvador Lucas, Alicia Villanueva.
Date: Marzo 2007
Keywords: Models of the Web, Graph representation, Formal Verification, Model Checking

Vº Bº
Leader of research Group Author(s)

Formal Verification of Websites

Sonia Flores, Salvador Lucas, and Alicia Villanueva

DSIC, Technical University of Valencia,
Camino de Vera s/n,
E-46022 Valencia, Spain,
{sflores,slucas,villanue}@dsic.upv.es

Abstract. In this paper a model for web sites is presented. The model is well-suited for the formal verification of dynamic as well as static properties of the system. A web site is defined as a collection of web pages which are semantically connected in some way. External web pages (which are related pages not belonging to the web site) are treated as the *environment* of the system. We also present the logic which is used to specify properties of web sites, and illustrate the kinds of properties that can be specified and verified by using a model-checking tool. In this setting, we discuss some interesting properties which often need to be checked when designing web sites. We have encoded the model using the specification language Maude which allows us to use the Maude model-checking tool.
Keywords: Models of the Web, Graph representation, Formal Verification, Model Checking

1 Introduction

The influence of Internet and web services in the society motivates the study of formal methods for the verification and validation of websites and web services. Nowadays, a huge amount of transactions or service requests are made by using Internet. In this context, the user needs to be sure that these transactions are secure and satisfy specific properties, for example that her personal data are not sent to unexpected Internet users. In general, websites must also satisfy some specific properties. For example, they might ensure that some private resources are only available to a given set of registered users.

The software developer or designer must guarantee the correctness and safety of the system, and to this end, he can apply some validation and verification techniques. Testing is a very useful technique which is able to detect errors in systems, and the special nature of the Web has made that testing techniques have been adapted to be applied to webpages, websites and web services. However, testing does not guarantee the absence of errors, as it only detects errors when they exist. In order to ensure the *absence* of a specific kind of error, the developer could use formal tools such as *model checkers*, typically used in contexts like embedded or reactive systems.

The first step for the application of any formal verification technique is to obtain a model of the system which captures all the necessary information for the

specific kind of verification. For this reason, in this paper we propose a model (in terms of a directed graph) which is suitable for the verification of many interesting properties. We also show how to verify properties which are specified by using a Linear Temporal Logic (LTL in short [18]). As we will show later, by using the model presented in this paper, more expressive properties can be considered by adapting different extensions of temporal logic.

The proposed model captures the information relative to essential properties such as *connectivity*, *reachability* and *web semantics* [8, 13]. We show how we can combine all of them to verify very interesting properties. Then we show how to apply a model-checking tool to the system in order to formally check a property. If the output from the model checker is *yes*, then we are sure that the system satisfies the property. Otherwise, the model checker provides a counterexample showing a trace (or path in our case) in which the property is *not* satisfied. This counterexample can be very useful for detecting the error.

The system we deal with is any (set of) website(s). Each website is composed of a set of webpages (source documents). The set of webpages not belonging to the website but whose URI appears in any link of the webpages of the site defines the *environment* of the system. Some semantics information might be associated to webpages by using different approaches such as the RDF framework [16], the OWL Web Ontology Language [23], by using the DARPA markup language [3], etc. Our framework captures the semantics of webpages depending on the specification language. In this paper we abstract from these topics assuming that we can handle and access the web semantics of pages. The integration of web semantics into our model allows the verification of properties related to it. Let us recall that the final purpose of this work is to analyze properties enabling the user to ensure the correct behavior of her systems, but also to analyze the structure of the website: how webpages are linked among them, whether webpages regarding the same (or related) topic are linked or not, etc.

Defining formal models for the Web is not a new topic. In [4], a model for a website able to capture information about links and frames of webpages was defined. The model defined in this paper is different from the one of [4] in several ways. First of all, we make the search space finite by restricting the analysis to a finite system; we abstract the model whereas in [4] the abstraction is made at the logic level. Another important difference is the kind of properties we are able to verify since our model captures different information from the source documents of the system. Finally, we want to mention that there exist many tools which verify static properties of websites, such as the size, connections, etc. [1, 2]. Our method is able to analyze, in addition, *dynamic* properties since paths can be explored in order to verify their properties.

The paper is organized as follows. Section 2 is a brief introduction to some notions related to the Web that will be used along the paper. In Section 3 the formalism used to model websites is presented and motivated. It is also illustrated how the information can be captured from source documents. Thereafter, in Section 4 the formalism for the specification of properties is presented. We also illustrate by means of examples the kind of properties that can be specified

and verified in this framework. In Section 5 we present how our framework is well suitable for new extensions improving expressiveness and efficiency for the formal verification of the Web. Finally, Section 6 presents our conclusions.

2 Introduction to Websites

In this work we model web pages and their connections as a directed graph whose nodes represent webpages and whose edges represent links among webpages. We restrict ourselves to the analysis of a website (which plays the role of the *system* in classical approaches to formal verification). Here, a website is a collection of webpages which are semantically related in some way and hosted at a single machine. Note that we could easily extend this notion of system by considering, for instance, a set of websites instead of a single one. In that case, a preprocess of the source documents is mandatory in order to adapt the notions, but the contents of this paper would remain valid. It's important to note that our focus is not on analyzing the entire Web, but rather an specific part. Therefore, we represent the *environment* (i.e., other parts of the Web) as a single special node. This fact does not limit the applications of the framework presented in this paper since, due to the flexibility of the notion of system, our method allows one to analyze huge parts of the Web. Moreover, as we are dealing with a formal model, optimization techniques of model-checking algorithms such as symbolic representations [6] or abstract interpretation [12] can be adapted to the Web context in order to mitigate the well-known state explosion problem.

In this section we introduce some notions related to webpages and websites that will be used along the paper. We focus on the information that we intend to handle for the verification process. In particular, our model abstracts out from concrete contents of webpages such as text, images, etc.

In this paper we assume (without loss of generality) that the webpages of a website are hosted at the same machine, and can be administrated by the same (set of) administrator(s). Usually, each webpage is defined by means of a document, typically specified in HTML, XHTML or XML markup languages. The examples shown along this paper assume that the specification language is XHTML, but the definitions can be parametrized w.r.t. any specification language with a similar expressiveness. Each webpage could contain links to other webpages. The source document where a link is specified is called *the source of the link* whereas the document where a link points to is called *the destination of the link*. Obviously, these two documents could coincide when the link is a local link (a link from one point in a webpage to a different one in the same webpage).

In order to be able to define a link to an specific point in a source document, a label (or anchor) to such specific point must be defined in the source document. This label will later be used in a link to specify the destination point within a webpage. These labels or anchors are the only information about the structure of a webpage that our model will take into consideration. This means that the relative position (on top, bottom or middle of the page) of an specific content or link is abstracted from. Note that as a direct consequence of considering the

labels in a document, more than one edge between two specific nodes (webpages) could exist in the model, each one referring to a different point on the webpage.

Being the basis for the definition of the kind of links our model will deal with, let us recall the definition of URI [22]. Intuitively, an URI is the description of where the source document of a webpage is hosted. It consists of three components:

1. the mechanism which is used to access the resource: usually http, https, or mailto;
2. the name of the machine hosting the site;
3. and the name of the resource itself, described as a path in the hosting machine.

Therefore, the URI of a document can be represented as a structure of the form uri(mech,host,resource) where mech $\in$ {http,https,mailto}[1], host is a string defining a domain, and resource is string representing the path where the source document can be found at the host. When the string resource contains the character # followed by a label, then the URI does not refer to the top of a webpage, but it defines a precise point within the source document, in particular the point where the label had been specified. Therefore, resource could refer both to a source document (when the label is not specified) and to a more specific point into a document. Let us show an example:

Example 1. The string `http://www.w3.org/2002/ws/sawsdl/#Publications` refers to a document accessible by using the `http` protocol which is hosted at the `www.w3.org` domain. The navigator can access to the source document by following the path `2002/ws/sawsdl/` at the host and, since the author has labeled a specific point in the document by defining the anchor `#Publications`, when the navigator follows the link, it doesn't show the upper point of the webpage but the section identified by the anchor `#Publications`.

The example above shows how an absolute URI can be defined. It is also possible to define relative URIs which are those where the information about mechanism and host are omitted due to the fact that they coincide with the mechanism and host values for the source document of the link. Therefore, the only described information in a relative URI is the path where the destination document is hosted. Again, it is possible to refer to an specific point in the document by using labels as shown above. Note that we could specify the access to a webpage in the website both by using a relative and an absolute URI. In this paper we assume that, whenever possible, URIs are specified in its relative form. This allows us to classify links (local, site or external) depending on the type of URI used to specify the resource.

The way in which links are specified depends on the specification language. Typically, links and anchors (i.e., labels which are associated to an specific point in the current document) are defined by means of the `LINK` and `A` elements of the

[1] Note that for the purposes of this work, only http and https values must be considered.

language. In order to specify the different characteristics of the link (or anchor), some attributes can be associated to these elements. In this paper we focus on the `name` and `href` attributes. `name` is associated to the `A` element in order to define a label (anchor) to the specific point of the document where the link is specified. `href` can be associated to both `A` and `LINK` elements, and contains the URI where the destination document is located.

Finally, our model considers the information regarding how the resource is being loaded: in the same navigator window, in a different one, in the same frame, etc. This information is specified by using the argument `target`. In this work we consider the following possible values for that argument: $\mathsf{Target} = \{\texttt{\_blank}, \texttt{\_self}, \texttt{\_parent}, \texttt{\_top}\}$. The `_blank` value tells the browser to load the destination webpage in a new window whereas `_self` makes the browser to load the destination in the same window (or frame) where the link is specified.

In the next section we define the graph structure modeling a website. We also illustrate the relation of each part of the graph with the corresponding source code.

3 The Website Model

In any formal verification process, one of the first tasks to be performed is the definition of the model that the verification algorithm will handle. In this section we present a graph structure that models websites. Recall that a website as described above is a collection of webpages that ideally is not merely a set of webpages but an entity with some properties that characterizes the website. Our model is defined in terms of a directed graph whose edges are labeled with some information related to links. Each node of the graph represents a webpage (a source document) whereas each edge represents a link specified in the source node and pointing to the destination node of the edge. The information that our model captures mainly regards links. A link connects two ends (webpages) and has one direction [22]. In the source of the link it must be specified the destination by means of a (relative) URI.

In the following we define the different components of our model. First of all we describe how nodes of the graph are formed. Each node represents a single webpages of the website. Thereafter we will introduce each component of the node. These components will represent the structure and connections that each webpage has. Examples of components are local, site or external links, etc.

Definition 1 (Node). *Let u be the URI associated to the webpage s. The node representing s is defined as the tuple $PageSkeletom(s) = \langle u, lab, loc, site, ext, Sem\rangle$ where* lab *is the set of labels (anchors) defined in s,* loc *is the set of local links defined in s,* site *is the set of site links defined in s,* ext *is the set of external links defined in s, and* Sem *is the set of annotations representing the semantic web information of the webpage s.*

As one can observe, our model is able to represent three kinds of links: local links, site links and external links. A local link relates a webpage with itself

by using a label (anchor) defined in the same document. A site link connects webpages from the same website, i.e., webpages that share the hosting machine and that semantically are related. Finally, an external link connects a specific webpage to any other resource not belonging to the website. We have defined this new representation of the Web to make the model suitable for the kind of properties we intend to verify. We discuss deeply this topic in Section 4.

The set of labels that are specified along the webpage document and that define the structure of the contents of the webpage is defined as follows:

Definition 2 (lab). *Let s be a source document, we denote as $lab(s)$ the set of specified values of attributes* `name` *and* `id` *specified for elements* `A` *or* `LINK` *in s.*[2]

To model a local link we need to specify both the label where the link points to, and the target (how to load the resource):

Definition 3 (Local link). *Let u be the URI of the webpage s. The pair $\langle l, t\rangle$ denotes a local link specified in s whose destination resource is s itself, $l \in lab(s)$, and $t \in$* Target.

Note that we do not need to specify the mechanism, host and resource (path) of the document since they coincide with the ones of s. For example, assuming that at some point along the document we have the following code

```
<a name="Participants">The list of the Working Group's
participants</a> is ...
```

that defines an anchor to that specific point of the webpage (`Participants` $\in$ $lab(s)$). Then, the following local link could appear at any other point of the same document

```
<a href="#Participants">Participation</a>
```

and the following link will be associated to the node $PageSkeletom(s)$

$$\langle l, t\rangle = \langle \texttt{Participants}, \texttt{\_self}\rangle$$

A site link is defined similarly. The difference w.r.t. local links is that the tuple has an additional component: the path from which the document can be retrieved. In order to retrieve the document, we need an auxiliary function doc that, given an URI uri(mech,host,res), retrieves the document associated to it. For example, if we have the URI uri(m,h,r), then doc(m,h,r) $= d$ being d a resource document.

Definition 4 (Site link). *Let* uri(m,h,u) *be the URI of document s (*doc(m,h,u) $=$ *s). The tuple $\langle p, l, t\rangle$ denotes a site link specified in s where* doc(m,h,p) $= d$, *$l \in lab(d) \cup \{\text{""}\}$ and $t \in$* Target.

[2] This definition establishes a direct relation between the model and the specification language. If a different specification language was used, the definition should be parametrized w.r.t. the new language.

The specification of the anchor is non compulsory and we represent this case by using the value $l =$ "". Let us show an example. The following code is the specification of a site link in the document s.

```
<a href="/../Activity"> Web Services Activity statement </a>
```

Following the Definition 4, the above example is represented by the following tuple which will be contained in the set of site links of $PageSkeletom(s)$.

$$\langle p, l, t\rangle = \langle \texttt{"/../Activity","", \_self}\rangle$$

The next example shows the correspondence between a site link referring to a specific label (`#groups`) and its model:

```
<a href="/../#groups"> About Web services </a>
```

$$\langle p, l, t\rangle = \langle \texttt{"/..","groups",\_self}\rangle$$

The third considered class of link is external links. We assume that these links point to resources hosted at any machine. These pages are not part of the system, i.e., part of the website, but they are part of the environment of the system. External links are also defined as a tuple. This time the tuple has two additional components:

Definition 5 (External Link). *Let* $\mathsf{uri}(m,h,u)$ *be the URI of the webpage* s*. Then the tuple* $\langle m', h', p, l, t\rangle$ *denotes an external link specified in* s *where* $\mathsf{doc}(m',h',p) = d$*,* $l \in lab(d) \cup \{\text{""}\}$*, and* $t \in \mathsf{Target}$*.*

An example of external link and its corresponding representation in the model is shown below:

```
<link href="http://www.dsic.upv.es/users/elp/elp.html"
      target=_blank>
```

$\langle m', h', p, l, t\rangle$ =
`<"http","www.dsic.upv.es","users/elp/elp.html","",_blank>`

Note that the extension of the framework to deal with *a set of* hosts is straightforward. We should need a preprocess to handle the new notion of local and external link.

The last component of nodes in the graph represents web semantics, i.e., the semantics associated to each webpage and that ideally are defined in the source document. In this paper we do not aim to study how the semantic web of a site must be inferred or defined: we assume that web resources have already had their semantics defined. In our graph structure we represent the semantics associated to a webpage by using a set of pairs. Pairs consist of two components: the name of the attribute type and its assigned value cont where the assigned value will typically be a list of keywords.

Definition 6 (Page Semantics). *The semantics of a webpage* s *is defined as the set* Sem *of pairs* $\langle \mathsf{type}, \{\mathsf{cont}\}\rangle$*.*

Next we show an example of meta data interpreted as part of the semantics of the web.

```
<meta name= "keywords" content="Web Services,
      Web Services Activity, SOAP, SOAP 1.2, WSDL,
      Web Services Architecture,Choreography,
      Web Services Choreography,WS-CDL, CDL, WSDL 2,WSDL 2.0,
      WS-CDL 1.0,WS-Addressing,Web Services Addressing,
      WS-Addressing 1.0,Semantic, annotation"/>
```

This information will be included in the *Sem* set of the node as the pair:

$\langle \mathsf{type}, \{\mathsf{cont}\}\rangle$ = `<"keywords",{"Web Services, "Web Services Activity",`
`"SOAP","SOAP 1.2","WSDL","Web Services Architecture",`
`"Choreography","Web Services Choreography",`
`"WS-CDL","CDL","WSDL 2","WSDL 2.0","WS-CDL 1.0",`
`"WS-Addressing","Web Services Addressing",`
`"WS-Addressing 1.0","Semantic","Annotation"}>`

Up to this point we have defined how nodes of the graph are formed. Next we define edges linking nodes. Recall that each node is a webpage, thus edges represent links among webpages and are defined depending on the components of each node. In order to show how edges are defined in the graph, we need an auxiliary function which, given a link, retrieves the node corresponding to the destination document of such link.

Definition 7 (Auxiliary function node). *We call N to the set of nodes in the graph and Web a special node representing all the nodes outside the system. Given a mechanism m, a host h, and a path p,* $\mathsf{node}(m,h,p)$ *returns*

- *the node $n \in N$ such that $n = \langle u, lab, loc, site, ext, Sem\rangle$ and $u = (m, h, p)$,*
- *or the node Web in case there exist no node in N corresponding to the URI (m,h,p).*

Note that the case when the function node returns the node *Web* corresponds to the case when the link is an external link. External webpages are represented by the special node *Web*.

Now we are ready to define how edges of the graph are defined:

Definition 8 (Edges). *For each node $n, n' \in N$ being (respectively) of the form $n = \langle u, lab, loc, site, ext, Sem\rangle$ and $n' = \langle u', lab', loc', site', ext', Sem'\rangle$, we define the set of edges E for the graph as follows:*

1. *For each $\langle l, t\rangle \in loc$, if $l \in lab$ then $(n, n, l, t) \in E$*
2. *For each $\langle p, l, t\rangle \in site$, $u = (mech, host, path)$, if $\mathsf{node}(mech,host,p) = n'$ and $l \in lab'$, then $(n, n', l, t) \in E$*
3. *For each $\langle m, h, p, l, t\rangle \in ext$, if $\mathsf{node}(m,h,p) = Web$, then $(n, Web, l, t) \in E$*

Finally, we formally define the graph structure representing the system:

Definition 9 (Website Model). *Given a set of webpages (documents) P defining the website S we define the model of the website site as $SiteModel(site) = \langle N, E\rangle$ where N is the set of $PageSkeletom(p)$ for each $p \in P$ and E is the set of edges as defined in Definition 8 on the set N.*

3.1 An Illustrative Example

In Fig. 1 we can see the webpage associated to the URI `www.w3.org/2002/ws/`.

Fig. 1. Web Page of W3C Web site

The model obtained for the website associated to such page is partially shown in Fig. 2.

Next we show the components of the node representing the above webpage which in the graph is depicted at the middle of the figure:

$\langle u =$ `uri(http,"www.w3.org","/2002/ws/")`,
$lab =$ {`"News","Events","Wiki and tools","Technical",`
`"Documents","Groups","drafts","discussion","related"`}
$loc =$ {$\langle$`"News",_self`$\rangle$,$\langle$`"Events",_self`$\rangle$,
$\langle$`"Wiki and tools",_self`$\rangle$,$\langle$`"Technical",_self`$\rangle$,
$\langle$`"Documents",_self`$\rangle$,$\langle$`"Groups",_self`$\rangle$}
$site =$ {$\langle$`"/../../sawsdl",_self`$\rangle$,$\langle$`"/../../../cg/",_self`$\rangle$,
$\langle$`"/../../chor",_self`$\rangle$,$\langle$`"/../../../addr",_self`$\rangle$,
$\langle$`"/../../../desc",_self`$\rangle$,$\langle$`"/../../../policy",_self`$\rangle$
$\langle$`"/../../../databinding",_self`$\rangle$}
$ext =$ {$\langle$`"http://search.w3.org/2000/xp/group",_self`$\rangle$}

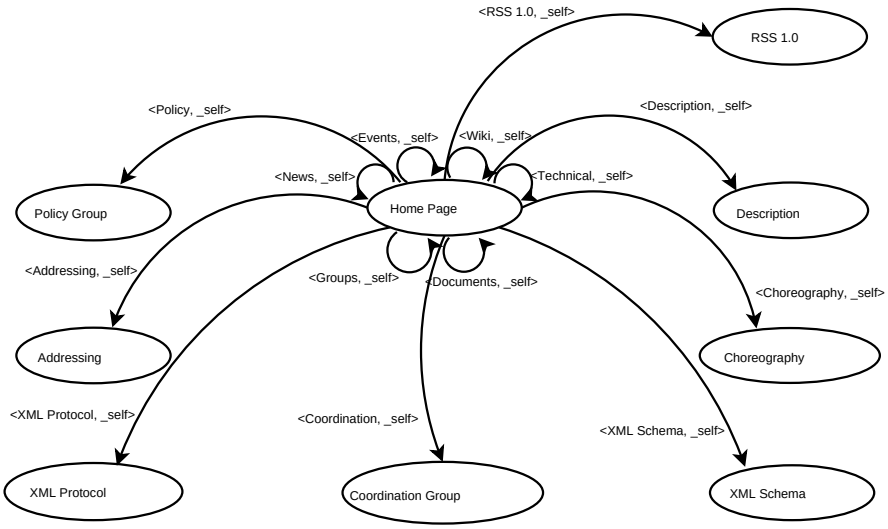

Fig. 2. Model of `www.w3.org/2002/ws/`

$$Sem = \{\langle \texttt{"keywords"},\{\texttt{"Web Services"},\texttt{"Web Services Activity"},\ \texttt{"SOAP"},\texttt{"SOAP 1.2"},\texttt{"WSDL"},\texttt{"Web Services Architecture"},\ \texttt{"Choreography"},\texttt{"Web Services Choreography"},\ \texttt{"WS-CDL"},\texttt{"CDL"},\texttt{"WSDL 2"},\texttt{"WSDL 2.0"},\texttt{"WS-CDL 1.0"},\ \texttt{"WS-Addressing"},\texttt{"Web Services Addressing"},\ \texttt{"WS-Addressing 1.0"},\texttt{"Semantic"},\texttt{"annotation"}\}\rangle,\ \langle \texttt{"revision"},\{\texttt{"\$Id: Overview.html"},\ \texttt{"v 1.230 2007/01/29 18:08:38 ylafon Exp\$"}\}\rangle\}\rangle$$

4 Verification of Properties

In this section we present a formal framework for the specification of properties. We use the well-known Linear Temporal Logic (LTL) [19] for the specification of properties. Let us recall the syntax of LTL formulas:

$$\varphi ::= p \mid (\neg\varphi) \mid \varphi \wedge \varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \mathcal{G}\,\varphi \mid \mathcal{X}\,\varphi$$

Where p is an atomic proposition, $\mathcal{U}$ is the *until* operator, $\mathcal{G}$ is the *always* operator, and $\mathcal{X}$ is the *next* operator.

The semantics of the logic are given in terms of a Kripke Structure which can also be seen as a directed graph whose nodes are labeled with the atomic propositions satisfied in each node. Formally, given a set of atomic propositions AP, we denote as $M = (S, S_0, T, L)$ a Kripke Structure where S is the set of states, $T \subseteq S \times S$ is the transition relation between states, $S_0 \subseteq S$ is the set of initial states, and L is a labeling function from S to the $\wp(AP)$. A sequence of states $\pi = s_0 \cdot s_1 \cdot \ldots$ is called a path of M whenever $(s_i, s_{i+1}) \in T$ for all

$i \geq 0$. When $s_0 \in S_0$, then we say that π is an initial path of the system. $\pi^i = s_i \cdot s_{i+1} \cdot \ldots$ denotes the suffix of the sequence π starting at the position ith. The semantics of LTL formulas is defined as follows:

$\pi \models p \quad \Leftrightarrow p \in L(s_0)$
$\pi \models \neg\varphi \quad \Leftrightarrow \pi \not\models \varphi$
$\pi \models \varphi \wedge \phi \Leftrightarrow \pi \models \varphi$ and $\pi \models \phi$
$\pi \models \mathcal{X}\varphi \quad \Leftrightarrow \pi^1 \models \varphi$
$\pi \models \mathcal{G}\varphi \quad \Leftrightarrow$ for all $i \geq 0$, $\pi^i \models \varphi$
$\pi \models \varphi \mathcal{U} \phi \Leftrightarrow$ exists $j \geq 0$ and for all $0 \leq i \leq j$ $\pi^i \models \varphi$ and $\pi^j \models \phi$

A Kripke Structure M satisfies a given formula φ ($M \models \varphi$) if and only if for each initial path π of M, $\pi \models \varphi$. The model-checking technique [10] takes one Kripke Structure as the model of the system and a temporal formula as the property to be checked. The verification algorithm then explores the model guided by the formula and answers whether the model satisfies it or not. In the following we show the correspondence between the model defined in the previous section and a Kripke Structure.

The relation between the two formalisms is straightforward except for the labeling function, i.e., the function that determines which propositions are satisfied in each node. Nodes and edges of both formalisms coincide. Labels on the edges of the graph can be integrated into the labeling of Kripke Structures by using well-known transformation techniques. Therefore, the temporal operators of the logic are interpreted on the edges of our model, i.e., on the links between webpages. Regarding the labeling function L, we define it in terms of the information stored in each node of our model, i.e., the semantics of the web, the URI and the defined links. The more information we include in the labeling function, the more precise the formulas can be.

We think that one of the most interesting things of our model is the fact that it allows reasoning about the web semantics. For this reason, next we are going to formalize how this information is included in the Kripke Structure. Given $n \in N$ a node of the graph, we call n' to the corresponding node in the Kripke Structure. Let Sem be the last component of the tuple n (i.e., the representation of the semantics of the webpage), $L(n') = \bigcup_t Sem_t|_2$ being Sem_t the tth element of Sem, and $|_2$ denoting the projection on the second component of the pair Sem_t.

4.1 Specification of properties

Temporal logic allows specification of properties such as safety properties (ensuring that something bad never happens) and liveness properties (ensuring that something good eventually happens). These properties are related to the infinite behavior of a system. Our main goal in this paper is to check properties related to the path among web pages. For example, it is important to guarantee that private webpages are available only to a given set of registered users. This property is crucial for many websites and we can model it as shown in the next example.

Example 2 (Registered User). Let S be a website and $SiteModel(S) = \langle N, E \rangle$ its corresponding model. Assume that the semantics of nodes contains information about the privacy of webpages. In particular, there exists a `type scope` whose value can be `public`, `private`, or `access`, meaning respectively that the page can be accessed by any user, only by registered users, or that it is a public page where users log-in.

The property that establishes that a private page can only be viewed by registered users is defined in LTL as follows:

$$\neg(\texttt{public}\, \mathcal{U}\, \texttt{private})$$

This means that it cannot exist an initial path along which we pass from one public page to a private one. We should pass first the `access` page. Note that the formula

$$\mathcal{G}\neg(\texttt{public}\, \mathcal{U}\, \texttt{private})$$

has a different (more restrictive) meaning since it checks whether the formula is satisfied by *any* path (not only initial paths). This prevents the system from passing from a private page to a public one and then again to a private page. For some websites this property could be very interesting.

A typical property that many approaches check is whether a link is broken [1, 2]. This is a static property in the sense that it do not depend on the paths of the model. We can express this property as illustrated in the next example.

Example 3 (Broken link). In our model, a broken link is characterized by a finite path whose last node is different from the *Web* node. We can express this property as follows:

$$\mathcal{G}(\mathit{true}\, \mathcal{U}\, (\texttt{web} \vee \texttt{final}))$$

assuming that $\texttt{web} \in L(\mathit{Web})$ and that the labeling function of each webpage with no defined link contains the value `final`. Note that this formula ensures that for any path in the graph, there exists an additional transition until the *Web* node (or a leaf) is reached. It is also modeled the case when the *Web* node is not reached due to an infinite loop among website pages.

Other properties that can be checked are whether a webpage has any link to itself, or whether there exist a webpage reachable by a direct link from any other webpage of the website. This last property is specially interesting since we could infere information regarding the structure of the website from it. The node reachable from any other node by a direct edge could be representing the *HomePage* of the website. Let ϕ be a characterization of a given webpage. The formula

$$\mathcal{G}((\neg(\texttt{web} \vee \texttt{final}))\, \mathcal{U}\, (\mathcal{X}\phi))$$

is satisfied whenever all the nodes except the one representing the environment have a direct link to the webpage ϕ.

An important property that is commonly checked regards connectivity [7, 20], i.e., to check whether every node in the website is reachable from another one.

We can statically check whether the URI of every node is used in at least one node different from itself but this wouldn't ensure that every node is reachable from any other since there could exist sets of nodes with no connection to some other nodes. To solve this problem we should verify for each node of the model whether it is reachable from at least an initial state (a state in S_0). We should verify that the formula

$$\neg(\texttt{init}\ \mathcal{U}\ \phi)$$

is **not** satisfied[3] assuming that for every initial state s, $\texttt{init} \in L(s)$ and that ϕ identifies the state we are analyzing.

4.2 The prototype

Model Checking [9, 21] has been proved as a very effective mechanism to formally verify dynamic properties. As we have said before, it is based on a temporal logic which analyzes not only the states of the system, but also the possible traces or paths that an execution can take. In fact, [4] is the first example of how model checking techniques can be adapted to the verification of the Web. In particular, special attention is paid on how to avoid the huge nature intrinsic to the Web. Our proposal is different from [4] in the sense that, first of all, we define a different, more precise model for the formal verification, and second, we use the traditional temporal logic for model checking.

In order to experiment with web site model checking, we have encoded the graph representing the model of the system in Maude [11]. As we are dealing with LTL properties, we can use the Maude model checker [14] to verify the above mentioned properties. Maude is a specification language with some features that make it suitable for the encoding of this kind of systems. We have modeled the graph and defined the transition between nodes of the graph as transition rules that move from one state to another.

5 Extensions of the framework

Some principles for the easy and intuitive navigation inside a web site must be respected. Rules that the literature proposes regard the style of navigation in terms of the size and the architecture of the site. There are properties that cannot be specified by using the LTL logic, in particular these where quantification is needed, for example for studying the cost to reach a given node. In order to be able to check that kind of properties, a real-time temporal logic is needed, i.e., not a qualitative but a quantitative logic where counting the number of time instants is possible. Several real-time temporal logic have been previously defined in the literature [5, 15], but in order to use them, the model checking algorithm of Maude must be adapted.

[3] We want to check if there exist a path satisfying the property $\texttt{init}\ \mathcal{U}\ \phi$ which is equivalent to checking whether the negation is not satisfied in every path.

Another example of useful property that we could check with a real-quantified temporal logic is the number of links pointing to one page, or the number of links defined in a document. For that kind of properties, we need to quantify on the number of possible paths, not only on the number of time instants, thus a branching temporal logic able to count branches should be needed. Note that this property is useful when we want to limit the number of links or self-links defined in (to) a webpage.

As the experiments with our prototype have been very interesting, we think it's worth extending the system in several ways. First of all, we plan to define a transformation method to automatically obtain the model from the source code of webpages depending on the specification markup language. The correct definition of the model is a crucial step on the verification process. Making it by hand is an error-prone process thus we should avoid it.

At the transformation level, the features regarding the *flexible* notion of system can be integrated. We can implement the preprocess which considers and analyzes the webpages URIs for determining which kind of link is each one, and whether pages belong or not to the site. Since the source document can be written in any markup language, the transformation method must be defined for each commonly used language.

Finally, we think that one of the most interesting extensions that can be done is to enrich the behavior of the method by defining heuristics for inferring information from the structure of the website (see the HomePage property described in the previous section).

6 Conclusions

In this paper we have defined a new model for websites which is more precise and flexible than other approaches. In our framework, the modeled and analyzed system can be a single website, but also a set of websites. The presented model is different from others in the literature such as [4] since it captures different information from the source documents. In particular, we are more precise when dealing with links and we can handle web semantics. Moreover, we restrict the state space by defining the notions of system and environment in the Web. Other approaches limit the search space at different levels, for example in [4] it is done at the logic level. Note that for applying classical formal methods it is necessary to have a finite state space. Another example on how to restrict the search space is found in [17], where LTL properties are defined to be checked w.r.t. a subset of states modeling an excerpt of the Web.

We also have integrated the web semantics (and other interesting characteristics of webpages) into the labeling function of the model. In such a way, we can verify properties related to the meaning of webpages, for example by checking whether pages with information about the same topic are accessible among them. We have also shown how it is possible to infer some properties from the structure of the website, which we think is a very important feature.

References

1. Link Checker Pro. `http://www.link-checker-pro.com`.
2. The Escape. `http://www.the-escape.co.uk/`.
3. The DARPA Agent Markup Language Homepage. `http://www.daml.org`, 2006.
4. L. de Alfaro. Model Checking The World Wide Web. In *Proc. 13th Int'l Conf. on Computer Aided Verification*, vol. 2102 of *LNCS*, pp. 337–349. Springer, 2001.
5. R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.
6. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} and beyond. *Information and Comp.*, 98(2):142–170, 1992.
7. J. Carrière and R. Kazman. WebQuery: Searching and Visualizing the Web through Connectivity. In *Proc. of the 6th WWW Int'l Conf.*, pp. 701–711, 1997.
8. S. Casteleyn, P. Plessers, and O. de Troyer. Generating Semantic Annotations During the Web Design Process. In *Proc. of the 6th Int'l Conf. on Web Engineering*, pp. 91–92. ACM Press, 2006.
9. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. of Work. on Logic of programs*, vol. 131 of *LNCS*, pp. 52–71. Springer-Verlag, 1981.
10. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 1999.
11. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The maude 2.0 system. In *Proc. of Rewriting Techniques and Applications*, vol. 2706 of *LNCS*, pp. 76–87. Springer-Verlag, 2003.
12. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Prog. Languages*, pp. 238–252, 1977. ACM Press.
13. D. Doernhoefer. Surfing the Net for Software Engineering Notes. *ACM SIGSOFT*, 31(3):8–16, 2006.
14. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. In *Proc. 4th Work. on Rewriting Logic and its App.*, vol. 71 *ENTCS*, 2002.
15. E. A. Emerson and R. Trefler. Parametric Quantitative Temporal Reasoning. In *Proc. 14th Annual IEEE Symp. on Logic in Computer Science*. IEEE Press, 1999.
16. RDF Core Working Group. Resource Description Framework (RDF). `http://www.w3.org/RDF`, 2004.
17. M. Haydar, S. Boroday, A. Petrenko, and H. Sahraoui. Properties and Scopes in Web Model Checking. In *Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pp. 400–404, 2005. ACM Press.
18. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, 1992.
19. A. Pnueli. The Temporal Logic of Programs. In *Proc. of the 18th IEEE Symp. Foundations of Computer Science*, pp. 46–57, 1977.
20. J. Punin and M.S. Krishnamoorthy. WWWPal System - A System for Analysis and Synthesis of Web Pages. In *Proc. of the WebNet 98 Conf.*, 1998. AACE.
21. J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th Int'l Symp. on Programming*, vol. 137 of *LNCS*, pp. 337–350, 1982. Springer-Verlag.
22. D. Raggett, A. Le Hors, and Jacobs I. *HTML 4.01 Specification*. W3C, 1999.
23. World Wide Web Consortium (W3C). OWL Web Ontology Language. Overview. `http://www.w3.org/TR/owl-features`, 2004.