Formal Verification of Websites [1]

Sonia Flores[2] Salvador Lucas[3] Alicia Villanueva[4]

DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022
Valencia, Spain

Abstract

In this paper, a model for websites is presented. The model is well-suited for the formal verification of dynamic as well as static properties of the system. A website 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 website) are treated as the *environment* of the system. We also present the logic which is used to specify properties of websites, and illustrate the kinds of properties that can be specified and verified by using a model-checking tool on the system. In this setting, we discuss some interesting properties which often need to be checked when designing websites. 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

Internet is an essential component of the modern Information Society. It provides easy and flexible access to information and to resources distributed all around the world. The development of the *Hypertext Markup Language* (HTML) [22] and the *Hypertext Transfer Protocol* (HTTP) in the nineties led to a happy marriage between wires, waves, and software components (which we call now *the Web*) which can be thought of as the main developments driving this change.

The development of websites (which can be often understood as sets of HTML documents) is an important task in modern software engineering. As expected in any software project, the web developers or designers must guarantee that the system (i.e., the website) satisfies some particular requirements. For instance, they might want to ensure that some private resources are only available to a given set of

---

[2] Email: sflores@dsic.upv.es

[3] Email: slucas@dsic.upv.es

[4] Email: villanue@dsic.upv.es

registered users. They could also aim to guarantee that some information (held as HTML documents in the site) is reachable by any user from any point in the site.

To this end, developers can apply validation and verification techniques and tools. Formal methods have proved their usefulness to guarantee (beyond any doubt) that the behavior of a software system corresponds to what has been specified. Formal methods try to establish mathematically provable connections between the formal description of the system (i.e., its semantics) and the properties of interest. Regarding the web, the underlying programming language is (some variant of) HTML. The first problem to solve is to devise a suitable semantics which fits the desired abstraction level and also enables the use of some logic and verification algorithms to express and check the properties. Viewed as software systems, websites can be understood as *reactive* or *interactive* systems [23]. Model checking [7,21], which is a well-known and successful technique for verifying *reactive* and *interactive* systems, provides an appropriate framework for dealing with such kind of systems.

In this paper, we propose a model (in terms of a directed graph) which is suitable for the verification of many interesting properties of websites. Being an essential aspect of the Web, the proposed model focuses on the information in the HTML sources concerning *connectivity* [10]. Following the *semantic web* approach, however, we also consider *annotations* [12,13], which complement the information associated to each node of the website. Such annotations can be thought of as *abstractions* of the (contents of the) HTML document. We show how to verify properties which are specified by using a Linear Temporal Logic (LTL [18]). Then we show how to apply a model-checking tool to the system in order to check the properties in practice. 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 (and eventually fixing) the error.

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 web pages. Thereafter, in Section 4, the formalism for the specification of properties is presented. We illustrate by means of examples the kinds of properties that can be specified and verified in this framework. We briefly describe the verification approach that we have implemented to check such properties in Section 4.3. In Section 5 we discuss the future extensions that potentially can be integrated in our framework. Some related work is discussed in Section 6. Finally, Section 7 presents our conclusions.

2 Introduction to Websites

In this work, we model web pages and their connections as a directed graph whose nodes represent web pages and whose edges represent links among web pages. 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 web pages which are semantically related in some way and hosted at a single machine. Note that we could easily extend the notion of system by considering, for instance, a set of websites instead of a single one. In that case, a preprocess of the web page documents is mandatory in order to adapt the notions, but the contents of this paper would remain valid. It is 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 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 [9] can be adapted to the Web context.

In this section, we introduce some notions related to web pages 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 web pages such as text, images, etc.

The system to be checked consists of a website. Usually, each web page in the website 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 (X)HTML, but the definitions can be parametrized w.r.t. any specification language with a similar expressiveness. The document (web page) 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.

In order to be able to define a link to a specific point within a document, a *label* (or *anchor*) at such point must be defined in the document. This label will later be used in a link to specify the destination point in the web page. Links and labels are the only information about the structure of a web page that our model considers. We discuss later how the model also includes semantic information.

Being the basis for the definition of the kind of links our model will deal with, let us recall the definition of URI [22]. It consists of three components:

(i) the mechanism used to access the resource: usually http, https, mailto or ftp,

(ii) the name of the machine hosting the site, and

(iii) 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,ftp,mailto} [5], host is a string defining a domain, and resource is a string representing the path at the host where the web page can be found. resource could refer both to a whole web page (when no label is specified in the string) or to a specific point into the web page.

It is also possible to define relative URIs: those where the information about

---

[5] Note that for the purposes of this work, only http, https and ftp values are considered.
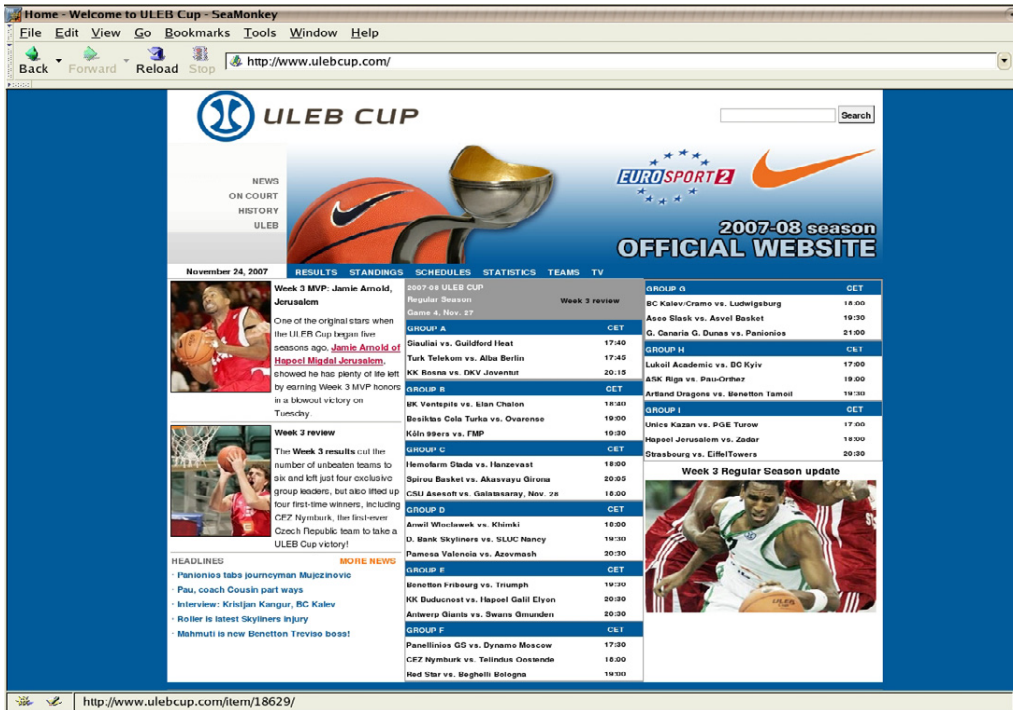
Fig. 1. ULEB Cup basketball competition homepage

mechanism and host is omitted. This means that we can specify the access to a web page in the website by using a relative or an absolute URI. In this paper we assume that, whenever possible, URIs are specified in its relative form. This allows us to classify links depending on the type of URI which is used to specify the resource.

The way in which links are specified depends on the specification language. Typically, links and anchors are defined by means of the LINK and A elements of the 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 a specific point of the document. href can be associated to both A and LINK elements, and contains the URI where the destination of the link is located.

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 the following, we present a graph structure modeling websites. Recall that a website is defined as a collection of web pages that is not merely a set of web pages but rather an entity with some properties characterizing the website. The model is defined in terms of a directed graph. Each node in the graph represents a web page whereas edges represent links from the source node to the destination node.
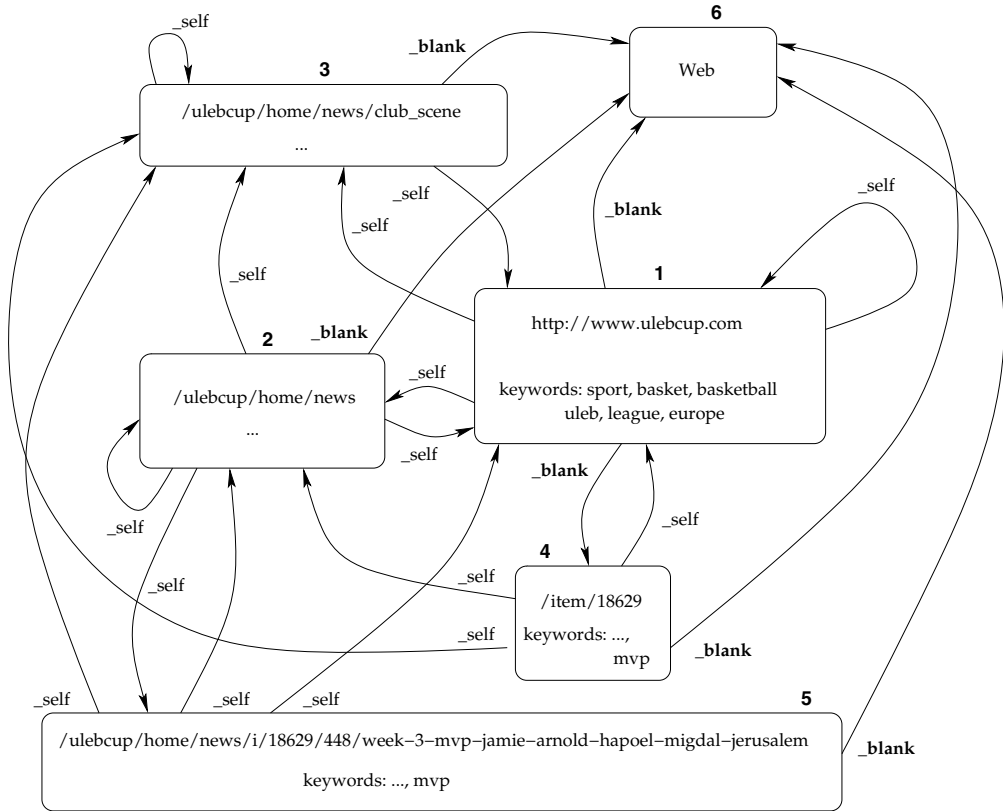
Fig. 2. The ULEB Cup website, 24th November 2007.

3.1   An illustrative example

Consider the website of the ULEB Cup competition (http://www.ulebcup.com [6])
shown in Figure 1. In Figure 2, we partially show the model representing the
website. In order to help to understand the explanations below, the nodes in the
graph have been given numeric labels. We have included 5 nodes representing 5
web pages in the website and a *Web* node (number 6) representing the environment
of the website. Node 1 represents the homepage of the website. Moreover, a menu
with the main web pages of the site is shown at the top of every web page in the site.
For this reason, every node has connections to nodes 1, 2 and 3 (which are part of
these main web pages). They allow us to navigate on competition news, history, on
court information, etc. Nodes 4 and 5 represent more specific information regarding
the *mvp* (most valuable player) of the week. We can directly reach the *mvp* web
page from the homepage, but not directly from any other web page. Note how the
club-scene web page has no link to nodes 4 or 5.

Semantic annotations defined for this site include information regarding the
topics covered by the site: sport, basketball, etc. These keywords are included in

---

the nodes representing web pages. Additional, more specific information could be added in a similar way. For example, we could add the keyword *mvp* in web pages represented by nodes 4 and 5. Finally, labels on the edges represent how the web page will be displayed by the navigation tool: in the same window as the source web page (`_self`), or in a new window (`_blank`). We aim at using all the information represented in the model to verify properties such as whether we can directly reach the homepage from every web page in the site.

3.2   Components of the model

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 web page of the website. Later we introduce how each component of the node is defined. These components represent the structure and connections that correspond to each web page.

Definition 3.1 Let u be the URI associated to the web page s. The node representing s is defined as the tuple $PageSkeleton(s) = \langle u, lab, loc, site, ext, Sem \rangle$ 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 web page s.

Our model can represent three kinds of links: local links, site links and external links. A local link relates a web page with itself by using a label (anchor) defined in the same document. A site link connects web pages from the same website, i.e., web pages that are semantically related. Finally, an external link connects a specific web page to any other resource not belonging to the website. Other approaches such as [1] use a different model. We have defined this new representation to make the model suitable for the kind of properties we intend to verify. We discuss this topic in Section 6. Abusing notation, we denote by $lab(s)$, $loc(s)$, $site(s)$, $ext(s)$ and $Sem(s)$ the obvious projections on the tuple defining the web page s.

The set of labels (anchors) specified within a web page is defined as follows:

Definition 3.2 Let s be a source document. We denote as $lab(s)$ the set of values of attributes `name` and `id` specified for elements `A` or `LINK` in s. [7]

Since we assume that links are specified in relative form whenever possible, a local link just specifies the label where the link points to.

Definition 3.3 Let u be the URI of the web page 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 $t \in \mathsf{Target}$ [8].

---

[7] 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.

[8] Values in `Target` represent the way in which the page is loaded by the navigator. Typical values for this component are `_blank`, `_self` or `_top`.

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

```
<a name="GroupB">Standings Group B</a> ...
```

defining an anchor to a specific point of the web page ($\texttt{GroupB} \in lab(s)$). Then, the following local link could appear at any other point of s

```
<a href="#GroupB">Group B Standings</a>
```

This is modeled as follows:

$$\langle l, t \rangle = \langle \texttt{GroupB}, \_\texttt{self} \rangle \in loc$$

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 a URI $\mathsf{uri}(\mathsf{mech},\mathsf{host},\mathsf{res})$, retrieves the document associated to it. For example, if we had the URI $\mathsf{uri}(m,h,r)$, then $\mathsf{doc}(m,h,r) = d$ denotes that d is the document associated to such URI.

Definition 3.4 Let $\mathsf{uri}(m,h,u)$ be the URI of document s ($\mathsf{doc}(m,h,u) = s$). The tuple $\langle p, l, t \rangle$ denotes a site link specified in s if $\mathsf{doc}(m,h,p) = d$, $l \in lab(d) \cup \{\text{""}\}$ and $t \in \mathsf{Target}$.

The specification of the anchor is non compulsory. We represent this case by using the value $l = \text{""}$. The following code is the specification of a site link in a document s.

```
<a href="/ulebcup/home/news"> News </a>
```

Following the Definition 3.4, the above example is represented by the tuple

$$\langle p, l, t \rangle = \langle \texttt{"/ulebcup/home/news"}, \texttt{""}, \_\texttt{self} \rangle \in site$$

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 defined as a tuple composed of five components:

Definition 3.5 Let $\mathsf{uri}(m,h,u)$ be the URI of the web page s. Then the tuple $\langle m', h', p, l, t \rangle$ denotes an external link specified in s if $\mathsf{doc}(m',h',p) = d$, $l \in lab(d) \cup \{\text{""}\}$ and $t \in \mathsf{Target}$.

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

```
<link href="http://www.euroleague.net/uleb/domestic-leagues",
      target=_blank>
```

$$\langle m', h', p, l, t \rangle = \texttt{<"http","http://www.euroleague.net",}$$
$$\texttt{"/uleb/domestic-leagues","",\_blank>}$$

Note that the extension of the framework to deal with *a set of* hosts is straight-

forward. We just need to add some preprocess to adapt the notions of local and external link.

We have already described the five first components of a node: the URI, the set of anchors and the three allowed kinds of links. The last component of each node models web semantics, i.e., the semantic annotations which are associated to the web page and that must be defined in the document; for instance, in the `meta` element as keywords. We do not pay attention to *how* a site is given semantic annotations. In our graph structure we represent the annotations associated to a web page by using a set of pairs consisting of an attribute (`type`) and its assigned value (`cont`), which is a list of keywords.

Definition 3.6 The semantics of a web page s is defined as the set *Sem* of pairs $\langle type, \{cont\} \rangle$.

Now we show an example of meta data which are defined as part of the semantics of the web.

```
<meta name="keywords" content= "sport, sports, basket, basketball,
          ULEB, league, europe, teams, players, uleb"/>
```

This information is included in the component *Sem* of the node as a pair:

```
<"keywords",{"sport", "sports", "basket", "basketball","ULEB",
          "league","europe", "teams", "players", "uleb"}>
```

Let us now define the edges which link the nodes. Recall that each node is a web page, and that edges represent links among web pages 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 a link.

Definition 3.7 Let N be the set of nodes in the graph and *Web* be a special node representing all the nodes outside the system. Given a mechanism m, a host h, and a path p, `node`(m,h,p) returns either

- 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* when no node in N corresponds to the URI (m,h,p).

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

Definition 3.8 Let N be a set of nodes. The set of edges E for the graph is given as follows: Given nodes $n, m \in N$ of the form $n = \langle u, lab, loc, site, ext, Sem \rangle$ and $m = \langle u', lab', loc', site', ext', Sem' \rangle$:

 (i) For each $\langle l, t \rangle \in loc$, if $l \in lab$ then $(n, n, l, t) \in E$

 (ii) For each $\langle p, l, t \rangle \in site$, $u = (mech, host, path)$, if `node`$(mech, host, p) = m$ and $l \in lab'$, then $(n, m, l, t) \in E$

(iii) For each $\langle m, h, p, l, t \rangle \in ext$, if `node`$(m,h,p) = Web$, then $(n, Web, l, t) \in E$

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

Definition 3.9 Given a set of web pages (documents) P forming the website S, we define the model of the website S as $SiteModel(S) = \langle N, E \rangle$, where N is the set of $PageSkeleton(p)$ for each $p \in P$ and E is the set of edges as defined in Definition 3.8 on the set N.

As an example, we show the components of the node representing the homepage of the ULEB Cup website shown in Figure 1 (node 1 in Figure 2).

```
⟨u = uri(http,"www.ulebcup.com",""),
 lab = {}
 loc = {}
 site = {⟨"/ulebcup/home/news",_self⟩,
         ⟨"/ulebcup/home/news/club-scene",_self⟩,
         ⟨"/ulebcup/home/news/on-court",_self⟩,
         ⟨"/ulebcup/home/news/sportlight",_self⟩,
         ⟨"/ulebcup/competition/results",_self⟩, ...}
 ext = {⟨"http://www.euroleague.net",_blank⟩, ...}
 Sem = {⟨"keywords",{"sport", "sports",
         "basket","basketball",
         "europe", "uleb", "ULEB"}⟩, ...}⟩
```

4 Verification of Properties

In this section, we develop a formal framework for the specification of properties. We use the well-known Linear Temporal Logic (LTL) [18,20] for the specification of properties. First, we 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 (which can be identified with a boolean variable), $\mathcal{U}$ is the *until* operator, $\mathcal{G}$ is the *always* operator, and $\mathcal{X}$ is the *next* operator. Other classical connectives can be defined in terms of the above ones as usual, in particular $\varphi \rightarrow \psi \equiv (\neg \varphi) \vee \psi$.

As usual, the LTL semantics is given in terms of Kripke Structures which can also be seen as a directed graph. Formally, given a set AP of atomic propositions, a Kripke Structure is a tuple $M = (S, S_0, T, L)$ where S is a set of *states*, $T \subseteq S \times S$ is a *transition relation between states*, $S_0 \subseteq S$ is a set of *initial states*, and L is a labeling function $L : S \rightarrow \wp(AP)$ that determines the propositions which are satisfied in each node (i.e., state). 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. Note that $\pi^0 = \pi$.

A Kripke Structure M satisfies a given formula φ ($M \models \varphi$) if and only if for each initial path π of M, it holds that $\pi \models \varphi$. Formally, $\models$ is inductively defined as follows:

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

Note that the labeling function L is essential to determine which formulas are satisfied by a model. The classical model-checking technique [7] takes a Kripke Structure as the model of the system and a temporal formula as the property to be checked. The verification algorithm explores the model guided by the formula and answers whether the model satisfies it or not.

4.1   Web sites as Kripke structures

Let us formally show the correspondence between the model defined in the previous section and a Kripke Structure. The relation is straightforward except for the labeling function L. First of all, labels on the edges of the graph can be integrated into the labeling of Kripke Structures by means of well-known transformation techniques obtaining a semantically equivalent graph; Regarding temporal operators, they are interpreted on the edges of our model, i.e., on the links between web pages. Finally, we define the labeling function of the Kripke Structure in terms of the information stored in each node of our model, i.e., the semantic annotations, the URI and the kind of links. Moreover, we define the atomic propositions of the logic in such a way that semantic keywords can be specified in formulas.

In our opinion, one of the most interesting things of our model is the fact that, thanks to the integration of semantic annotations within the labeling function, it is possible to reason about web semantics. Let's clarify how this information is included into the Kripke Structure.

Definition 4.1 Given $n = \langle u, lab, loc, site, ext, Sem \rangle \in N$ be a node of the graph, we call n' to a corresponding node in the Kripke Structure. Assume that Sem is indexed by I. Then, $L(n') = \bigcup_{i \in I} Sem_i|_2$ where Sem_i is the i-th element of Sem and $|_2$ is the projection on the second component of the pair Sem_i.

4.2   Specification of properties

Temporal logic makes possible the specification of *safety* properties (ensuring that nothing bad will happen) and also of *liveness* properties (ensuring that something good will eventually happen). These properties are related to the infinite behavior of a system. In our context, the temporal view of the classical model-checking approach is transformed into an *accessibility* view, since the modal operators are interpreted on the links connecting web pages. Therefore, we check properties related to paths among web pages. Let us consider some interesting properties that can be modeled.

4.2.1 Privacy/secure access

In some cases, it is important to guarantee that private web pages 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 following example.

Example 4.2 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 web pages. In particular, there exists a `type` *scope* whose values can be any of `{public,private,access}` meaning respectively that the page can be accessed by any user, only by registered users, and that we are dealing with a public page used to register users.

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

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

Note that, thanks to the definition of the labeling function in Kripke's model, we can use the semantic annotations as atomic propositions. The property says that it cannot exist an initial path along which we pass from one public page to a private one. In fact it should first pass through the `access` page.

The following version of the formula has a different (more restrictive) meaning since it checks whether the formula is satisfied by *any* path (not only initial paths).

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

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.

4.2.2 Immediate links

Let us now check whether a web page has any link to itself, or whether there exists a web page which is reachable by a direct link from any other web page. The last property is specially interesting since we could infer some information regarding the structure of the website from it. The node which is reachable from any other node by a direct edge could often be thought of as representing the *Homepage* of the website. Let ϕ be a characterization of a given web page. The formula

$$\mathcal{G}((\neg(\texttt{web} \vee \texttt{leaf})) \ \rightarrow \ (\mathcal{X}\phi))$$

is satisfied whenever all nodes except the one representing the environment and *final* web pages, have a direct link to the web page ϕ.

4.2.3 Connectivity

Finally, a relevant property that is commonly checked in the literature (see, e.g., [14]) concerns connectivity, i.e., ensuring that every node in the website is reachable from an initial one. In order to deal with this problem, we should verify, for each

node of the model, whether it is reachable from an initial state $s \in S_0$. In particular, we should verify that the formula

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

is **not** satisfied[9], assuming that for every initial state s, $\texttt{init} \in L(s)$ and that ϕ identifies the state we are analyzing.

Similarly, to ensure that every node is reachable from any other, it would not be enough to statically check whether the URI of every node is used in at least one node different from itself since there could be sets of nodes without any connection to other set of nodes. Actually, we should check a similar property to the previous one but not only for the initial nodes, but for every considered node.

4.3   The prototype

In order to experiment with website model checking, we have encoded the graph representing the model of the system in Maude [8]. The current version of our prototype parses HTML web pages to models (as defined along this paper) encoded in Maude. As we are dealing with LTL properties, we can use the Maude model checker [11] to verify the above mentioned properties on the website model. Maude is a specification language based on rewriting logic 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 (web page) to another.

Recall that a model checker is a tool able to formally verify whether a property is satisfied by a model, and in case it was not satisfied, then it provides the counterexample found. In the classical approach, this allows the programmer to detect where the error is located and to fix it. In our case, the counterexample helps the designer to identify a flaw in the website design.

As the experiments with our prototype have provided very interesting results, we think it is worth extending the system in several ways. First of all, we plan to develop parsers for other markup languages (not only HTML). We think that it is an important issue since the correct definition of the model is a crucial step on the verification process. Writing the model by hand is an error-prone process than we should avoid.

The features regarding the flexibility of the notion of system can be integrated in the prototype. We plan to implement the preprocess which considers and analyzes the web pages' URIs for determining the kind of link (local, site or external), 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 language commonly used by programmers and designers.

---

[9] We want to the *existence of a path* satisfying the property $\texttt{init}\ \mathcal{U}\ \phi$. LTL formulas are checked for *all paths*; thus, we have to use the equivalent strategy that checks whether the negation is not satisfied (for all paths).

5   Extensions of the framework

The guide [19] establishes some principles for the easy and intuitive navigation inside a website. The rules that the guide proposes regard the style of navigation in terms of the size and the architecture of sites. The guide specifies some properties that cannot be specified by using the LTL logic, in particular these where quantification is needed.

In order to be able to check such kind of properties, a real-time temporal logic is needed, i.e., not a qualitative but a quantitative temporal logic where counting the number of time instants is possible [4,5]. To deal with this kind of logic, the model-checking algorithm of Maude should be adapted. In our opinion, an interesting extension of our framework is to consider such kind of properties and to adapt a model-checking algorithm to deal with the new logic and with our model. Note that, in any case, the model presented in this paper should not be modified.

Other extensions of logics can be considered. For example, the first order version of the LTL logic can be useful to prove a typical property that many approaches check: whether there exist broken links. In our model, a broken link is characterized by a finite path whose last node has more elements in its *local*, *site* and *external* components that edges from it. We can check the property as a static property in the sense that it depends on the information in a given node and the edges from such node, thus we can check it for every reachable node in the graph:

Example 5.1 Assuming that $\texttt{web} \in L(\textit{Web})$ and the information about the URI ($\texttt{uri}$) of web pages and links are considered in the labeling function of nodes, the following formula models the property:

$$\mathcal{G}((\neg\texttt{web}) \rightarrow (\forall l \in \textit{links}, \neg(\mathcal{X}(\neg\texttt{uri} = l))))\,^{10}$$

We also plan to consider not only quantifying on the number of steps, but also on the number of paths. Properties regarding the number of links pointing to one page, or the number of links defined in a document can be very useful for a web designer, for example to restrict the number of links defined in (to) a web page. To be able to check such kind of properties, we need to quantify on the number of possible paths, thus a branch temporal logic able to count branches is needed.

Finally, in our opinion, 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). The inferred information could be added to the semantics of the model such that more sophisticated properties (assuming the inferred information) could be checked.

---

[10] In this example the notation for links has been simplified.

6   Related work

Defining formal models for the Web is not a new topic. In [1], a model for a website able to capture information about links and frames of web pages was defined. Model Checking [7,21] 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, [1] 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 infinite component intrinsic to the Web nature. Our proposal is different from [1] in the sense that, first of all, we define a different, more precise model for the formal verification. Moreover, we restrict the search space by restricting the analysis to a finite system (a website), whereas in [1] this abstraction is made at the temporal logic level redefining the μ-calculus. Another example on how to restrict the search space is found in [14], where LTL properties are defined to be checked w.r.t. a subset of states modeling an excerpt of the Web.

Another important difference w.r.t. [1] is the kind of properties we are able to verify, implied by the fact that our model captures different information from the site. Finally, we want to mention that there exist many tools which verify static properties of websites, such as the size, connections, etc. Our method is able to analyze *dynamic* properties in the sense that paths can be analyzed in order to check their properties.

In [17], a different formalism is used in order to check the web. Term rewriting techniques are used in order to model the dynamic behavior of the Web. Then, properties regarding reachability can be verified under some restrictions that are imposed due to the decidability results of reachability for the different term rewriting theories. Our approach improves the one in [17] in the sense that we are able to verify more expressive properties by using an effective algorithm. Finally, note that [17] uses Maude as a specification model for rewriting theories whereas in this work, we use Maude to specify the defined graph model.

The rule-based approach has also been used in [2,3], where static properties regarding both syntax and semantics of the web are verified. A new technique inspired in declarative debugging algorithms has been developed in order to check errors regarding correctness as well as completeness of websites w.r.t. a given specification. The main difference between this approach and the framework presented in this paper is that we are interested on dynamic properties that can be specified by using temporal logics and that can be checked by using a model-checking algorithm.

Finally, other models for the web have been defined along the years such as the different versions of the random graph models [15,16]. The principal aim of these works is not to apply formal methods to the analysis or verification of websites. Random graph models are useful in order to *measure* the web in the sense that they try to model the web and perform a number of search algorithms that counts, for example, the number of links that contains a web page, or the number of links that points to a given web page. The numeric results are analyzed in order to

identify clusters regarding a topic, *hub* web pages, etc. However, our main purpose is the verification of properties by applying formal methods, in particular model-checking algorithms, thus the model must differ. Note that it is possible to quantify and *measure* the web by using quantitative logics such as the real-time logics as mentioned in Section 5. In conclusion, we are able to analyze different properties by using an unified formalism: temporal logics, whereas in [15,16] different algorithms must be run in order to study different aspects of the web. Finally, we note that some of the search algorithms described in these works abstract the environment of the system similarly as we do.

7 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 intended as a collection of web pages semantically related, but also a set of websites. We have demonstrated that the model captures enough information to apply formal methods such as model checking or inference of properties. We have established a concrete relation between our model and Kripke Structures, what makes possible to apply the model-checking technique to the Web. The most important issue in such relation is the definition of the labeling function of the Kripke Structure. In particular, we have shown how the labeling function must be defined in order to handle web semantic information of websites. This allows us to check properties related to the semantic information of web pages such as the topic, the language in which it is written, etc.

We have illustrated which kind of properties can be specified and verified by using the LTL logic, and which logics should be considered in order to specify more sophisticated properties. We have also described the possible extensions of our framework, which we plan to address as future work. A very interesting extension is to develop a methodology to infer information from the properties.

As future work, we also plan to improve the current version of the prototype, adding new features such as parsers for other specification languages, making flexible the notion of systems, etc.

References

[1] L. de Alfaro. Model Checking The World Wide Web. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 337–349. Springer-Verlag, 2001.

[2] M. Alpuente, D. Ballis, and M. Falaschi. Rule-based Verification of Web Sites. *Software Tools for Technology Transfer*, 8(6):565–585, 2006.

[3] M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, and D. Romero. A Fast Algebraic Web Verification Service. In *First International Conference on Web Reasoning and Rule Systems*, volume to appear of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[4] M. Alpuente, M. M. Gallardo, E. Pimentel, and A. Villanueva. A Real-Time Logic for tccp verification. *Journal of Universal Computer Science*, 12(11):1551–1573, 2006.

[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 Computation*, 98(2):142–170, 1992.

[7] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 1999.

[8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude – A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer Verlag, 2007.

[9] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, New York, 1977. ACM Press.

[10] M. Cutler, Y. Shih, and W. Meng. Using the structure of html documents to improve retrieval. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[11] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. In Fabio Gadducci and Ugo Montanari, editors, *4th Workshop on Rewriting Logic and its Applications (WRLA'02)*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[12] S. Handschuh and S. Staab. *Frontiers in Artificial Intelligence and Applications - Annotation for the Semantic Web*. IOS Press, 2003.

[13] S. Handschuh, S. Staab, and A. Maedche. CREAM - Creating relational metadata with ontology driven annotation framework. In *ACM*, 2001.

[14] M. Haydar, S. Boroday, A. Petrenko, and H. Sahraoui. Properties and Scopes in Web Model Checking. In *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering*, pages 400–404, Long Beach, California, USA., November 2005. ACM Press.

[15] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–17, 1999.

[16] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. S. Tomkins, and E. Upfal. The Web as a graph. In *Proceedings of the 19th ACM SIGACT-SIGMOD-AIGART Symposium on Principles of Database Systems, PODS*, pages 1–10. ACM Press, 15–17 2000.

[17] S. Lucas. Rewriting-based navigation of Web Sites. In *Proceedings of the First International Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, volume 157 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2005.

[18] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, New York, 1992.

[19] F. Millerand and Martial O. Guide Practique de Conception et d'evaluation Ergonomique de sites Web. Technical Report 25, Centre de Recherche Informatique du Montreal, Montreal QC, August 2001.

[20] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium Foundations of Computer Science*, pages 46–57, 1977.

[21] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–350, Berlin, 1982. Springer-Verlag.

[22] D. Raggett, A. Le Hors, and Jacobs I. *HTML 4.01 Specification*. W3C, MIT, Inria, Keio, 1999.

[23] K. Schneider. *Verification of Reactive Systems. Formal Methods and Algorithms*. Springer-Verlag, Berlin, 2004.