Verification of Web-Content: A Case Study on Technical Documentation

Christian Schönberg, Mirjana Jakšić, Franz Weitl and Burkhard Freitag

Department of Informatics and Mathematics, University of Passau
{Christian.Schoenberg, Mirjana.Jaksic, Franz.Weitl, Burkhard.Freitag}@uni-passau.de

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Technical Report, Number MIP-0904
Department of Informatics and Mathematics
University of Passau, Germany
February 2009

Abstract. In this paper, we present the results of a case study on a novel approach to document verification. Employing new techniques of user constraint specification and model checking, our aim is to bridge the gap between logical precision and usability, thus enabling authors and inexperienced users to employ formal verification methods. Based on a technical documentation in the form of a web document, we show that our approach is effective, efficient and has a high usability. Additionally, we argue that document verification is highly relevant for many applications, but especially for web content and hypertext documents.

1 Introduction

Keeping technical documentations in a consistent state – w.r.t. both structure and content – is a hard task. Many documentations today are compiled from a number of separate resources and text fragments, depending on current requirements and priorities. Online documentations complicate matters further because they usually offer more than one (linear) path through the document, rendering content consistency almost impossible to check manually.

At the same time, publishing documents online, in digital formats, is steadily gaining in importance. Most manufacturers make their technical documentations available on the web, while reusing content that is common to more than one product. This further increases the impact and relevancy of automatic document verification [KSS04].

As part of the Verdikt[1] project [WJF09] we propose a framework that employs information extraction, temporal description logics, and model checking to verify consistency criteria on a multitude of document types. In this paper, we will describe the techniques in the context of a case study in the domain of technical documentation.

The results of the case study

- confirm the usability of our method for users not acquainted with formal methods,
- show the limited effort required to prepare a web document for model checking,
- reveal the efficiency of model checking, and
- demonstrate the precision and usefulness of error reports generated by model checking.

As a result, our approach helps bridging the gap between formal precision and usability. This sets it apart from existing work in the field of document verification.

For the purpose of this use case, we have adopted a version of our approach that is simplified in several areas. At the appropriate points, we will describe briefly how our framework goes beyond the techniques described here. Our approach is explained in more detail in [WJF09].

The rest of this paper is organized as follows: Section 2 describes our use case. Section 3 gives an overview of the Verdikt framework. Sections 4 through 6 contain more detailed descriptions of the main components of the framework and exemplify the overall processing based on our use case. Section 7 presents the results achieved, section 8 discusses related work, and section 9 concludes the paper.

[1] This work is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant number FR 1021/7-1.

2 Use Case

The case study presented in this paper concentrates on the domain of technical documentation. As a sample document we picked a manual of a digital satellite receiver published at the manufacturer's web site and anonymized it, basically by replacing company names, product brands, images, and most distinct phrases. This lead to a document containing all the general features, characteristics, and errors of the original but being anonymous in the sense that it cannot be linked to a specific brand or seller. As a result, we obtained a document of 80 printed pages, split into 25 HTML files that is a typical representative of technical documents in terms of content, structure, and size. By convention, the content of each HTML file is called a *chapter* of the document in the sequel.

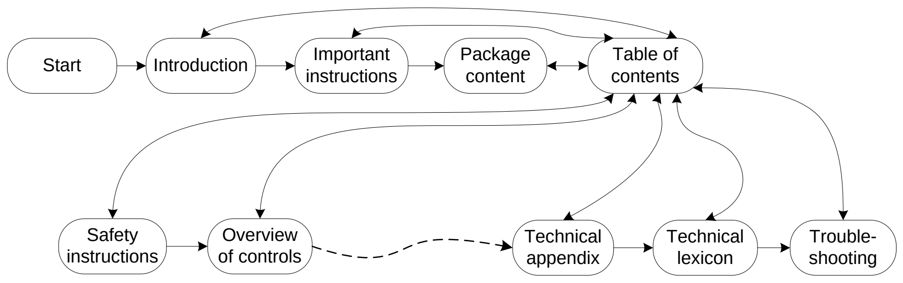


Fig. 1. Structure of the sample document

Fig. 1 presents a part of the basic structure of the web document in the form of a directed graph of HTML pages / chapters (vertices) and links between them (edges). The document begins with a *title page* (start), followed by a short *introduction*, followed by *important instructions*, *package content*, and a *table of contents*. The *table of contents* is succeeded by *safety instructions* and *overview of controls*. Further follow the instructions about all the settings, options, and functions of the receiver that could be used. The manual includes a *technical appendix* – an overview of the complete configuration of the receiver, and a *technical lexicon* – an overview of all abbreviations and technical terms with explanations. The document ends with a *troubleshooting* chapter.

The manual contains some problems that severely limit its readability. First, inconsistent notation is used for some technical terms. For example, as a short form of the term **Conditional Access Module**, the notation **CA-Module** has been used in all chapters of the document, except for the *technical lexicon*, where this term is referred to as **CAM**. We also found some abbreviations like USB, EPG, FBAS, which are not explained in the *technical lexicon* at all. Finally, there are some interfaces (like Ethernet) shown in the chapter *overview of controls*, which are not described later on.

Among the objectives of this case study is to demonstrate the usability for the end user and to determine the effort required to apply our framework in a real-world scenario. Both are important factors for the general practicability of the approach. In addition, the efficiency of the system is evaluated by runtime tests.

3 Verdikt Framework

In the context of the Verdikt project, a general framework for document verification has been developed [WJF09], which is divided into three major components: the information extraction and model generation component, the specification and formula generation component, and the model checking and error reporting component.
The first component (information extraction) reads all relevant data from the source document, stores it for further processing, and creates a verification model suitable for model checking. The second component (specification) allows the user to specify criteria to be applied to the input document. To this end, a high level approach to the specification process based on specification patterns is used [JF08]. The third component employs temporal description logics and model checking techniques to verify the specification against the model and to track errors to their origin in the source document. Temporal description logics allow for the concise representation of consistency criteria evaluated along some or all paths through the document the reader can sensibly follow. These paths are subsequently called *reading paths*. For instance, "Start" → "Introduction" → "Table of contents" → "Troubleshooting" is a reading path in the document depicted in Fig. 1.

A detailed overview of the framework is shown in Fig. 2. The user defines a set of consistency criteria to be applied to a document (top left corner). These user level specifications are converted into logical formulae to be used by the model checker (top center). From a document base (bottom right corner), data is extracted and stored as RDF metadata statements, which can be combined with background knowledge about the domain of discourse. The user can choose what metadata is relevant for the current application and should therefore be transferred to the verification model (top right corner), which is then passed on to the model checker. The model checking component, based on temporal description logics (TDL), calculates the verification result according to the specification and the verification model, from which an error report is distilled and presented to the user (bottom center). To make sense to the user, the error report has to take into account the original user specification and has to refer to the source document. Those source references are added to the document metadata.

4 Information Extraction

Extracting the necessary data from the document is performed in three stages: (a) Extract the data from the document sources (in this case HTML), (b) store

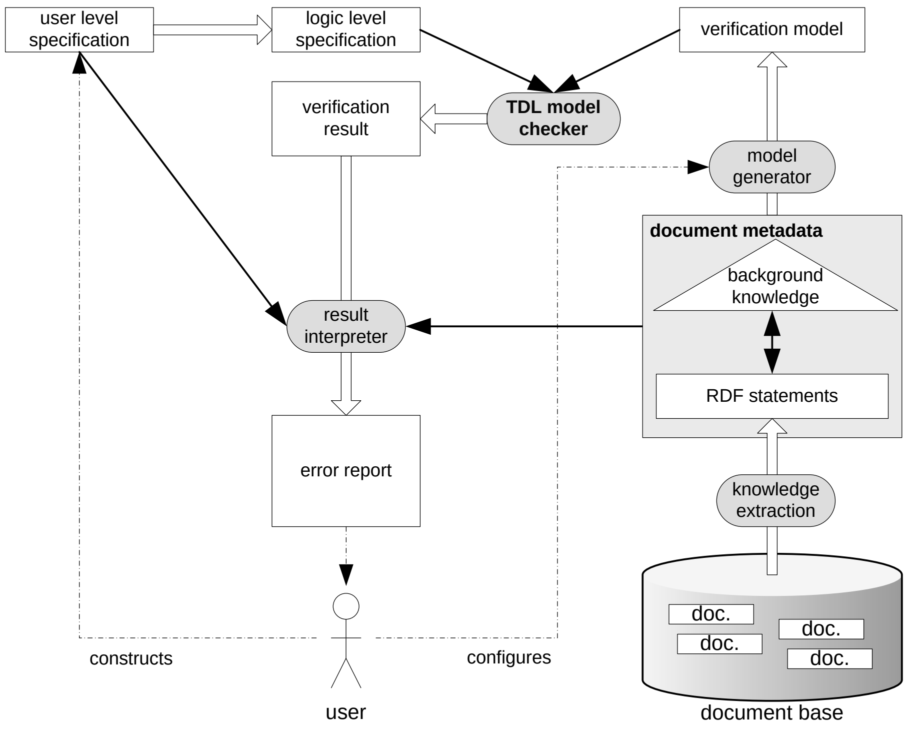


Fig. 2. Overview of the Verdikt framework

the data in an RDF database, and (c) convert the data to a verification model according to the semantics of the logic level specification language $\mathcal{ALC}\mathsf{CTL}$ (see sections 5 and 6). The model generated in step (c) can be adjusted to meet different requirements, for example a fine-grained model describing every paragraph of the document, or a more abstract model on the level of chapters. To facilitate creating different models without having to reprocess the original document, or to allow for ontology inferencing, the data is stored in an RDF database.

(a) HTML extraction. First, the HTML source files are preprocessed using JTidy[2] to produce valid XML code. Subsequently, they are converted into RDF triples by means of an XQuery program. Using the Qexo[3] XQuery implementation, we developed and added further functionality to the XQuery program, including an extended context to keep track of current and previously parsed elements, list-like data structures to facilitate a file history and to avoid infinite loops, and several convenience methods to create valid RDF XML code.

[2] Java HTML Tidy, © World Wide Web Consortium
[3] Qexo is part of GNU Kawa, © Per Bothner

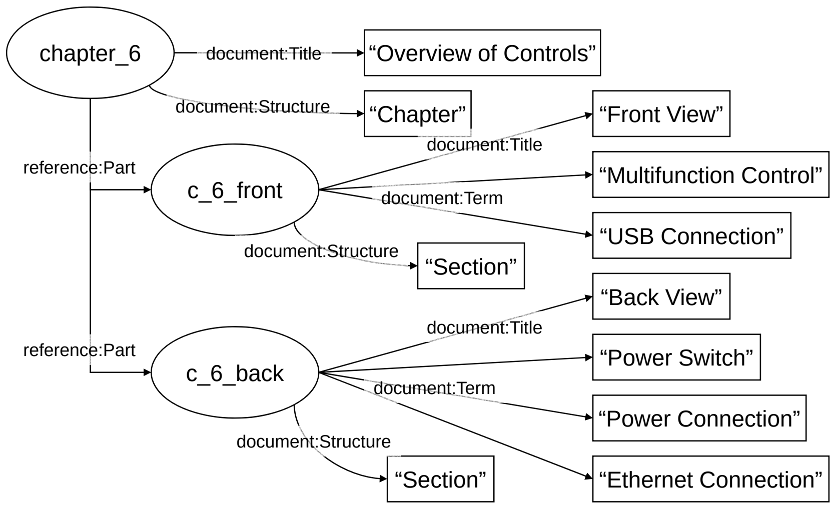


Fig. 3. Small extract of the RDF metadata about the sample document

Using a predefined vocabulary and CSS classes to infer structural information, a metadata description of the sample document is generated, which is represented in RDF (see Fig. 3).
Structural metadata includes information about the document's chapter and subchapter structure, as well as the content type of text units: for example, a section is identified as an *introduction* or as a *technical appendix*. In our case study, evaluating stylesheet classes and rudimentary keyword analysis were sufficient to determine all important attributes. For more complex documents, we have dictionaries and thesauri at our disposal, and we have made some initial experiments based on natural language processing and machine learning.
Similar techniques can be applied to the extraction of content information, which is however the more difficult task. In this case study, we used background knowledge about the important terms, so we could employ elementary grammar rules to find all instances of those terms in the text.

(b) RDF database storage. After a comparative analysis of both the capabilities and the performance of different RDF database systems [SF09a], we decided to employ the Sesame Framework[4], an open source software that supports several relational databases, including PostgreSQL and MySQL, and different query languages, including SPARQL and an extension of RQL. It both outperformed and offered better reliability than its major competitor, the Jena Framework[5].

In our case study, the RDF data amounted to nearly 900 statements. This number can easily grow by one or several orders of magnitude for very large doc-

[4] © Aduna Software
[5] © Hewlett-Packard Development Company, LP

uments or interconnected web pages. In these cases, memory management becomes an issue, and storing the data in a database system is mandatory [SF09a].

(c) Verification model generation. We again use XQuery to generate a verification model from the RDF graph. This gives us more flexibility in customizing the verification model to the requirements of the use case than native RDF query languages would, since these do not support recursive queries required to track paths in the RDF graph [SF09a]. Instead of simply transferring the entire metadata about the document to the verification model, we exclude information irrelevant for the desired specifications. This increases the efficiency of model checking and helps to facilitate a greater ease of use by providing a concise set of vocabulary for user level specifications.

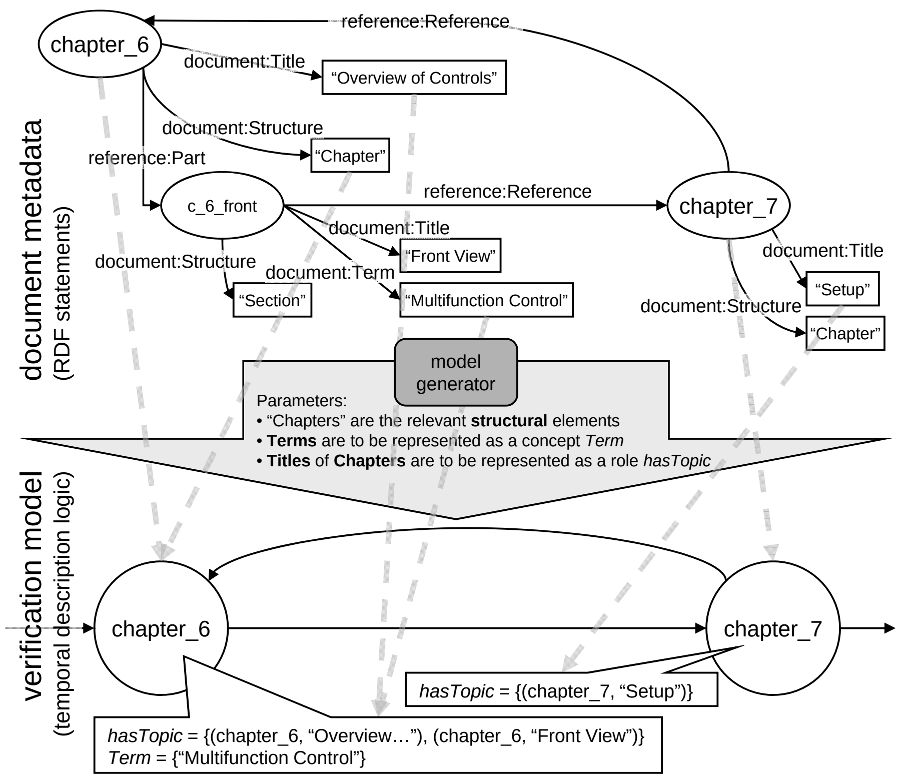


Fig. 4. Generating the verification model from the RDF metadata

Fig. 4 provides an example of how the verification model is generated from the document metadata w.r.t. a set of external parameters. The parameters for this example specify that the relevant structural unit is the "Chapter" (as opposed to e.g. “Section” or “Paragraph”), that any technical terms should be represented by a concept named *Term*, and that the relation between chapters and their titles and subtitles should be represented by a role named *hasTopic*. The verification model is described in detail in section 6.

5 Formal Specification

For the purpose of our case study, we have chosen the following sample consistency criteria:

1. Each abbreviation has to be defined later on.
2. Any technical term used in the document should be explained in the *technical lexicon*.
3. The *safety instructions* should be listed immediately after the *table of contents*.
4. The *package content* should be listed before the *table of contents*.
5. Any interface shown in the *overview of controls* should be explained later on in the *technical appendix*.

To prove these or any other criteria automatically, they have to be expressed in some formal language. For the formal representation of consistency criteria, we use the temporal description logic $\mathcal{ALC}\mathsf{CTL}$, which has been introduced in [Wei08]. $\mathcal{ALC}\mathsf{CTL}$ is a combination of the description logic $\mathcal{ALC}$ [BN03] and the branching time temporal logic CTL [Eme90]. $\mathcal{ALC}$ is expressive for representing structured properties of single content elements. CTL is expressive for representing loose criteria on reading paths through the document. The combination of description and temporal logics provides high expressiveness for content-related criteria w.r.t. reading paths.

Consider the first sample criterion: *Each abbreviation has to be defined later on*, which can be expressed in $\mathcal{ALC}\mathsf{CTL}$ as:

$$\mathit{AbbreviatedTerm} \sqsubseteq \mathsf{EF}\ \mathit{DefinedTerm} \tag{1}$$

AbbreviatedTerm and *DefinedTerm* are concepts representing an abbreviation and a definition of a technical term, respectively. $\sqsubseteq$ expresses that all instances of the concept to its left (in this case *AbbreviatedTerm*) are also instances of the concept to its right (in this case EF *DefinedTerm*). EF (read “some path future”) is a temporal connective representing the set of objects which on some path are eventually an instance of the concept in the scope of the EF quantification. For instance, EF *DefinedTerm* is the set of terms being defined in some text unit reachable from the current text unit on some reading path.

In addition to the atomic concepts *AbbreviatedTerm* and *DefinedTerm* in formula (1), $\mathcal{ALC}\mathsf{CTL}$ provides constructors to form complex concept terms [Wei08] that enable the representation of relevant properties which cannot be expressed by existing propositional temporal logics such as CTL and LTL [Eme90].

Expressing consistency criteria in a temporal description logics like $\mathcal{ALC}\mathsf{CTL}$ requires good mathematical knowledge and usually involves considerable effort. For these reasons, $\mathcal{ALC}\mathsf{CTL}$ is not suitable to be employed directly by the end user. In the context of the case study, we manually prepared natural language formulations of the selected target criteria. Of course, this approach does not scale up to larger scenarios. In addition, criteria expressed in natural language are usually ambiguous. We suggest specification patterns [JF08,WJF09] to support the end user in specifying different types of consistency criteria without having to deal with $\mathcal{ALC}\mathsf{CTL}$ or being limited to a predefined set of natural language sentences.

6 Model Checking and Error Reporting

Within our framework, specifications represented in $\mathcal{ALC}\mathsf{CTL}$ are verified by model checking [Wei08,WJF09]. $\mathcal{ALC}\mathsf{CTL}$ model checking combines high precision with excellent performance [Wei08]. In the case of specification violations, counterexamples are generated that precisely pinpoint the error locations within the document.

Here, we cannot give a comprehensive introduction into model checking $\mathcal{ALC}\mathsf{CTL}$. Instead, we illustrate $\mathcal{ALC}\mathsf{CTL}$ model checking informally on a simplified part of our use case and refer the reader to [Wei08] and [WJF09] for technical details.

The model checking problem of $\mathcal{ALC}\mathsf{CTL}$ is defined on top of two structures:

- a finite, non-empty set of $\mathcal{ALC}\mathsf{CTL}$ formulae F that are derived from pattern-based specifications [JF08,WJF09] and represent consistency criteria to be met by a document d.
- a finite verification model M_d of document d that is derived from RDF-based metadata (Fig. 4) and represents the structure and content of document d.

Fig. 5 depicts a simplified part of the verification model of the presented use case. The verification model is an annotated graph (S, R, I): Nodes S of the graph represent text units of the document (in our case: web pages), edges R represent links between text units, and annotations I represent the content of each text unit of the document.

In Fig. 5, S contains the nodes "chapter_0" (start), "Table of contents", ... R contains, for instance, an edge from "Table of contents" to "chapter_7" and an edge from "chapter_7" back to "Table of contents" (Fig. 5 center). I maps each node in S onto a set of annotations. For instance, $I(chapter\_7)$ represents the annotations for node "chapter_7" in Fig. 5 center bottom:

$$hasTopic = \{(chapter\_7, \text{“Setup”})\} \tag{2}$$

$$AbbreviatedTerm = \{\text{“LNB”}\} \tag{3}$$

This expresses that "chapter_7" covers the topic "Setup" and mentions the abbreviation "LNB". The technical details of defining the interpretation I are omitted here for brevity. The respective formal definitions can be found in [Wei08].

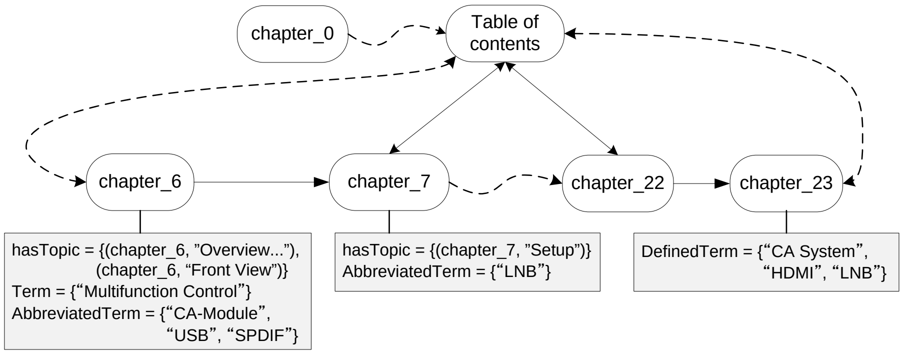

chapter_0
Table of contents
chapter_6
chapter_7
chapter_22
chapter_23
hasTopic = {(chapter_6, "Overview..."), (chapter_6, "Front View")}
Term = {"Multifunction Control"}
AbbreviatedTerm = {"CA-Module", "USB", "SPDIF"}
hasTopic = {(chapter_7, "Setup")}
AbbreviatedTerm = {"LNB"}
DefinedTerm = {"CA System", "HDMI", "LNB"}

Fig. 5. Simplified verification model of the sample document

In contrast to models of propositional temporal logics such as CTL [Eme90], the interpretation I of $\mathcal{ALC}$CTL verification models allows to express relationships among objects of the modeled domain. For instance, the fact that the objects represented by *chapter_7* and "Setup" are in a *hasTopic* relationship in node *chapter_7* (equation (2)) cannot be represented directly by means of propositional formalism such as CTL. Being able to represent semantic interrelationships among parts and topics of a document is vital for verifying content-related properties.

The semantics of $\mathcal{ALC}$CTL [Wei08] defines when an $\mathcal{ALC}$CTL formula f holds at a node $s \in S$ of a verification model $M_d = (S, R, I)$, in symbols $M_d, s \models f$.

Given a verification model $M_d = (S, R, I)$ and an $\mathcal{ALC}$CTL formula f, *model checking* is defined as determining the set

$$Nodes(M_d, f) := \{s \in S \mid M_d, s \models f\}$$

i.e. the set of nodes at which a formula f holds in a model M_d [Wei08].
$Nodes(M_d, f)$ represent the parts of the document that conform to a requirement represented by formula f.

In [Wei08] we have shown that the $\mathcal{ALC}$CTL model checking problem is decidable and has a polynomial runtime complexity. Further, we have defined a sound and complete algorithm for determining the set $Nodes(M_d, f)$. This algorithm runs in $\mathcal{O}(|d|^3 \cdot |f|)$ where $|d|$ denotes the size of the document d, and $|f|$ denotes the size of the formula f to be verified. Documents of several thousands of web pages are verified in less than 5 seconds. For comparison, the state-of-the-art CTL model checker NuSMV [CCG$^+$02] takes more than 30 seconds for only 500 pages under similar conditions [Wei08].

For an illustration of model checking, let M_d be the verification model depicted in Fig. 5 and f be the $\mathcal{ALC}$CTL formula

$$AbbreviatedTerm \sqsubseteq \mathsf{EF}\ DefinedTerm$$

expressing that each "abbreviated term" is on some path within the graph (S, R) eventually a "defined term" (compare section 5).

Then *chapter_7* $\in$ $Nodes(M_d, f)$ because for the (only) abbreviated term "LNB" in *chapter_7* (Fig. 5 center bottom) there is a path to the node *chapter_23* where "LNB" is a defined term (Fig. 5 rhs bottom). However, *chapter_6* $\notin$ $Nodes(M_d, f)$ because there are abbreviated terms "CA-Modules","USB", and "SPDIF" in *chapter_6* (Fig. 5 lhs bottom) that are not defined terms in any node reachable from *chapter_6*.

The nodes in $S \backslash Nodes(M_d, f)$ represent the *error locations* within document d w.r.t. formula f, i.e. the parts of the document d that do not conform to the criterion represented by formula f. By applying appropriate naming conventions, these nodes can be re-mapped easily onto the respective parts of the document. For instance, node *chapter_6* represents the part of the document that is contained in *chapter_6.html* (cf. Table 1, first data row).

error location	**violating terms**
chapter_6.html	"CA-Modules","USB", "SPDIF"
chapter_10.html	"USB", "CIM"
chapter_11.html	"CA"
...	...

Table 1. Error report of verifying formula $AbbreviatedTerm \sqsubseteq \mathsf{EF}\ DefinedTerm$

Based on the model checking results, an error report as sketched in Table 1 is generated. The first data row of Table 1 expresses that the terms "CA-Modules", "USB", and "SPDIF" used in web page *chapter_6.html* are not satisfying the formula $AbbreviatedTerm \sqsubseteq \mathsf{EF}\ DefinedTerm$.

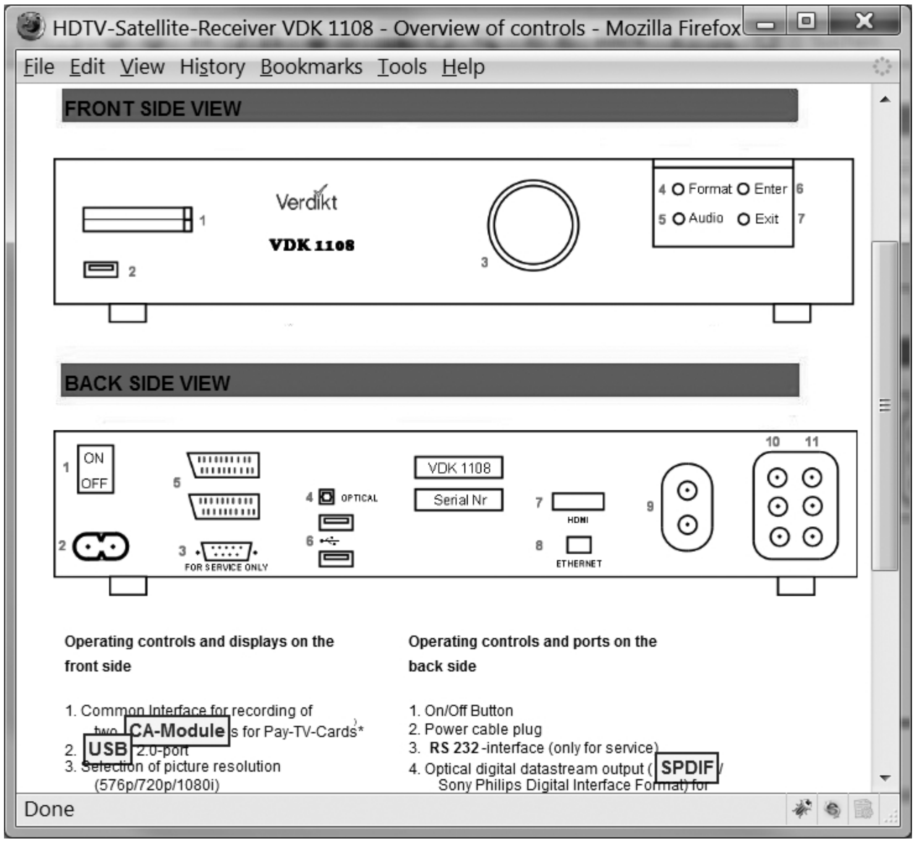
In addition to the error report, a CSS file is generated that highlights the violating terms within an error location of the document. Fig. 6 shows the presentation of "chapter_6.html" with violating terms "CA-Module", "USB", and "SPDIF" being highlighted (bottom of Fig. 6).

7 Results

7.1 Quantitative Results

Table 7.1 summarizes the quantitative results of the case study. We checked a manual of a satellite receiver, consisting of 25 HTML files, against a set of five criteria each of them being represented by a single $\mathcal{ALC}\mathsf{CTL}$ formula (first and second row in Table 7.1).

18 of 25 web pages had errors ("# error locations" in Table 7.1). These web pages contained, in total, 48 terms that violated one of the specified properties ("# violating terms" in Table 7.1). This is surprising since the manual has



Operating controls and displays on the front side

1. Common Interface for recording of two CA-Modules for Pay-TV-Cards*
2. USB 2.0-port
3. Selection of picture resolution (576p/720p/1080i)

Operating controls and ports on the back side

1. On/Off Button
2. Power cable plug
3. **RS 232** -interface (only for service)
4. Optical digital datastream output (SPDIF / Sony Philips Digital Interface Format) for

Fig. 6. Violating terms highlighted within the verified document

# chapters of manual / HTML pages	25
# formulae	5
# violated formulae	3
# error locations	18
# violating terms	48
total runtime	9.1 s
time taken by knowledge extraction	4.4 s
model generation	4.5 s
model checking	0.1 s
report generation	0.1 s

Table 2. Results for verifying an online manual of a satellite receiver

not been verified in a pre-release stage but has already been published by the manufacturer.

The runtime results listed in Table 7.1 have been obtained on a desktop computer with Intel Pentium IV processor at 3.2 GHz and 2 GB RAM running Windows XP and Java Version 6. The verification system has been implemented in Java.

The entire verification process took about 9 seconds (Table 7.1 center). The major portion of runtime was consumed by the knowledge extraction and model generation process (Fig. 2) that analyzes HTML markup, generates an RDF description of the relevant parts of the document, and finally delivers a verification model as described in sections 4 and 6. Note, however, that the verification model can be generated off-line in a preprocessing step and re-used across different verification runs.

Checking the verification model against the $\mathcal{ALC}$CTL-based specification and generating an error report each took just 1% of the total runtime. This ensures a quick response of the system when constructing and testing different specifications interactively.

The *application costs* for our verification system arise from

- *preparing the document.* In our case study, the document has been converted from PDF to HTML format by using the HTML export function of Adobe Acrobat. The resulting HTML code has been cleaned up, anonymized, and annotated manually which took about eight hours. Documents in a more structured original format would require considerably less effort.
- *preparing the specification.* Five criteria for the content of the document have been expressed in natural language and then formalized in terms of $\mathcal{ALC}$CTL. This took about two hours.
- *configuring the system.* The knowledge extraction, model generation, and error reporting components have been configured and parameterized according to the format and markup of the document, which took about two hours in total.

Altogether, the setup cost of our verification system amounts to about 12 hours of manual effort. This initial effort amortizes quickly when a document is changed frequently or parts of it are re-used in different contexts which is typically the case for technical documentations.

7.2 Qualitative Results

Intelligent preprocessing of the document, an expressive yet concise specification method, and a well-structured GUI result in a high degree of usability that is unmatched by existing verification or validation methods for documents (see section 8).

The usability of the approach has been demonstrated at a large exhibition of the University of Passau targeted at the general public. Visitors that did not have any knowledge of verification techniques or technical documentation were able to use the system and understand its verification results after being given a brief introduction.

We have to add that the presented system was limited w.r.t. the type of specifications that could be verified. Currently, we are successively increasing the flexibility of the system without jeopardizing its usability. An intelligent specification assistant leads the user from a first vague idea of a criterion to a precise, unambiguous specification using a structured base of predefined specification patterns and application examples [JF08].

8 Related Work

Schematron [Jel02] and xlinkit [NCEF02] are powerful tools for validating the consistency of XML documents. Our approach is different from these and other XML validation techniques in the following aspects. First, our method is not limited to XML documents but can be applied to other formats, e.g. HTML, Microsoft Word, or LaTeX, to name a few (cf. [Wei08,SF09b]). Second, properties of reading paths are hard to express and inefficient to check using XPath, which is fundamental both to Schematron and xlinkit. Finally, Schematron and xlinkit are not designed to be used by authors without detailed knowledge about XML processing.

There are several approaches (e.g. [SFC98,SDM$^+$05,FLV08]) using some propositional temporal logics (CTL, LTL), which enable the specification of complex properties along browsing paths in hypermedia structures. Although $\mathcal{ALC}\mathsf{CTL}$ exceeds the expressive power of these formalisms regarding semantic relationships within the modeling domain, we achieve a better usability by adding a user specification layer on top of the formal core. In addition, the higher expressiveness of $\mathcal{ALC}\mathsf{CTL}$ results in richer and more precise error reports that clearly pinpoint problems within the document.

A formal consistency management component based on description logics is proposed in [ESS05] as an extension to the content management system for technical documentation Schema ST4 [Gru06]. Extensive tool support ensures a good usability – at least for authors experienced in technical documentation. However, description logics on their own are not sufficiently expressive for representing criteria on reading paths through the document (cf. [Wei08]).

A powerful and flexible framework for checking the consistency of collections of interrelated documents has been proposed by [Sch04]. The formal basis is full first order logics interpreted over a language defined in terms of the functional programming language Haskell. While the suggested formalisms are very expressive they are also very complex both in terms of computation and application costs. Our approach offers a better compromise between high expressiveness and formal precision on the one hand, and efficiency, usability, and low application costs on the other.

9 Conclusion

We have sketched a new verification framework for web-based documents and presented the results of a case study on an online manual of a satellite receiver.

The core of the verification framework consists of flexible RDF-based metadata representation, configurable generation of verification models, a temporal description logic as an expressive specification language, and formal verification of specifications by model checking.

The case study shows that the modularity and flexibility of the framework reduces its application cost. Temporal description logics and model checking have been demonstrated as powerful, efficient, and precise methods for the verification of web documents. Error reports generated from model checking results pinpoint error locations precisely within the document and highlight problematic terms. The system offers a high degree of usability and can – in a restricted version – be applied instantly by users without any pre-knowledge in the area of document verification. As compared to existing approaches, a better compromise between expressive power, low application costs, and usability has been found. Thus, an important step towards closing the gap between the power of formal methods and their practical applicability has been achieved.

The results of this study have also raised some issues worth to be examined in more detail.

For one, while the efficiency of the model checking algorithm is already satisfactory, the efficiency of the metadata extraction process still needs to be increased. We believe that, among other things, using more native RDF database methods may help in that regard. A first prototype has shown encouraging results.

To identify important terms and topics, we used a small amount of background knowledge during the extraction process. What remains to be investigated, however, is the trade-off of model quality vs. reasoning effort.

References

[BN03] Franz Baader and Werner Nutt. Basic description logics. In *The Description Logic Handbook - Theory, Implementation and Applications*, chapter 2, pages 47–100. Cambridge University Press, 2003.

[CCG$^+$02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proceedings of Computer Aided Verification (CAV 02)*, volume 2404 of *LNCS*. Springer, 2002.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, pages 996–1072. Elsevier, 1990.

[ESS05] U. Egly, B. Schiemann, and J. Schneeberger. Tech. documentation authoring based on semantic web methods. *Künstliche Intelligenz*, 2:56–59, 2005.

[FLV08] S. Flores, S. Lucas, and A. Villanueva. Formal verification of websites. *Electronic Notes in Theoretical Computer Science*, 200:103–118, 2008.

[Gru06] Schema Gruppe. Schema ST4 Leistungsbeschreibung. SCHEMA Electronic Documentation Solutions GmbH, 2006.

[Jel02] Rick Jelliffe. The schematron assertion language 1.6. http://xml.ascc.net /resource/schematron/Schematron2000.html, 2002. last visited Feb. 2009.

[JF08] M. Jakšić and B. Freitag. Temporal patterns for document verification. Technical Report MIP-0805, University of Passau, Germany, 2008.

[KSS04] Reiner Kuhlen, Thomas Seeger, and Dietmar Strauch, editors. *Grundlagen der praktischen Information und Dokumentation.* K. G. Saur Verlag, 2004.

[NCEF02] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)*, 2(2):151–185, 2002.

[Sch04] Jan Scheffczyk. *Consistent Document Engineering.* Dissertation, Universität der Bundeswehr München, 2004.

[SDM$^+$05] E. Di Sciascio, F. M. Donini, M. Mongiello, R. Totaro, and D. Castelluccia. Design verification of web applications using symbolic model checking. In D. Lowe and M. Gaedke, editors, *Proc. of the 5th Internat. Conf. of Web Engineering ICWE 2005*, volume 3579 of *LNCS*, pages 69–74. Springer, 2005.

[SF09a] C. Schönberg and B. Freitag. Evaluating RDF querying frameworks for document metadata. Technical Report MIP-0903, University of Passau, Germany, 2009.

[SF09b] C. Schönberg and B. Freitag. Extracting and storing document metadata. Technical report, University of Passau, Germany, 2009. to appear.

[SFC98] P. David Stotts, Richard Furuta, and Cyrano Ruiz Cabarrus. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *Information Systems*, 16(1):1–30, 1998.

[Wei08] Franz Weitl. *Document Verification with Temporal Description Logics.* PhD thesis, University of Passau, 2008.

[WJF09] F. Weitl, M. Jakšić, and B. Freitag. Towards the automated verification of semi-structured documents. *Journal of Data & Knowledge Engineering*, 68(3), 2009.