Journal of Symbolic Computation 46 (2011) 183–206

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Verifying the consistency of web-based technical documentations☆

Christian Schönberg[a,1], Franz Weitl[b], Burkhard Freitag[a]

[a] University of Passau, Department of Informatics and Mathematics, 94030 Passau, Germany
[b] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

ARTICLE INFO

Article history:
Received 9 June 2010
Accepted 2 August 2010
Available online 27 August 2010

Keywords:
Document verification
Model checking
Information extraction
Document modeling
Temporal description logics

ABSTRACT

A new framework for document verification is presented which covers the entire process from document analysis through information extraction, document modeling, representation of background knowledge about the domain of discourse, user level and formal representation of consistency criteria, verification by model checking, counterexample generation, and error reporting. Emphasis is placed on employing background knowledge to reduce the complexity and to increase the quality of results in each step. A rule-based approach to information extraction supports the concise definition of extraction rules for document formats based on XML or HTML. The expressiveness of the existing extraction methods is exceeded by supporting rule specialization, integration of external tools, and access to background knowledge represented in ontologies. As a formal basis for representing consistency criteria, the new temporal description logic $\mathcal{ALC}$CTL is proposed. In contrast to the existing formalisms, criteria related to the coherence of content along individual paths of reading can be represented and verified efficiently. The adequacy, performance, and effectiveness of the proposed framework is demonstrated on a case study in technical documentation.

☆ This work is partially funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant number FR 1021/7-2, and by the German Academic Exchange Service (DAAD) under the program "Research at International Science and Technology Centers".

E-mail addresses: Christian.Schoenberg@uni-passau.de (C. Schönberg), weitl@nii.ac.jp (F. Weitl), Burkhard.Freitag@uni-passau.de (B. Freitag).

URLs: http://www.im.uni-passau.de (C. Schönberg), http://researchmap.jp/weitl (F. Weitl), http://www.im.uni-passau.de (B. Freitag).

[1] Tel.: +49 0 851 509 3182.

0747-7171/$ – see front matter © 2010 Published by Elsevier Ltd
doi:10.1016/j.jsc.2010.08.007

1. Introduction

Keeping technical documentations in a consistent state – w.r.t. both structure and content – is challenging. 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, online documents are steadily gaining in importance. Most manufacturers publish their technical documentations on the web, while reusing content common to more than one product. This further increases the impact and relevancy of automatic document verification.

There is a variety of approaches to document verification either based on XML methods like XSLT and XPath such as Schematron (Jelliffe, 2002), on first order logic, e.g., xlinkit (Nentwich et al., 2002) and CDET (Scheffczyk, 2004), or on propositional temporal logic and model checking (Stotts et al., 1998; Sciascio et al., 2005; Flores et al., 2008). Methods based on XML processing or on first order logics are suitable for checking semantic dependencies in the content of a document. However, they are not efficient for checking properties along individual browsing paths, the number of which usually grows exponentially in the size of the document. Propositional temporal logics and model checking are efficient tools for verifying properties on browsing paths. However, propositional formalisms are insufficiently expressive to account for semantic interrelationships among parts and topics of a document.

In this paper, we propose a framework that employs information extraction, temporal description logics, and model checking to verify content- and path-related consistency criteria on a multitude of document types. In contrast to the existing approaches, various sources of background knowledge about the document and its domain of discourse, like ontologies of important terms and their semantic relationships, are exploited to reduce the complexity and to enhance the precision of both the information extraction and the verification process. Moreover, criteria related to the coherence of content along individual paths of reading can be represented concisely by the new temporal description logic $\mathcal{ALC}$CTL and verified efficiently by model checking, both of which were not possible previously. To the best of our knowledge, the presented approach constitutes the first application of a temporal description logic for verification by model checking.

The verification of content- and structure-related properties has to be based on nontrivial methods, because the specification formalism needs to be sufficiently expressive to capture both semantic interrelationships and paths, while remaining tractable, i.e., verifiable in polynomial time. As another dimension of complexity, the verification process strongly depends on both the correctness and completeness of the metadata extracted from the documents to be checked and on the appropriateness of the generated internal representation of the documents. The latter, in turn, is often dependent on application-specific verification goals. These challenging requirements are met by making use of various sources of background knowledge in the information extraction process and by proposing a new temporal description logic that offers a better compromise between expressiveness and computational complexity than the existing formalisms.

The goals and general architecture of our approach have been presented in Weitl et al. (2009) and the system has been demonstrated in Schönberg et al. (2009). This article is an extended version of Schönberg et al. (2009) that provides further conceptual and technical details. Its major new contributions are

- The *information extraction process* is presented in detail. It is shown how *production rules* and different types of *background knowledge* about a document and its domain of discourse are used to generate an RDF-representation of the document.
- It is demonstrated how *verification models* suited for efficiently checking document properties are generated from RDF descriptions. To this end, we propose the use of *SPARQL query templates* that can be adapted by the user.
- The description of $\mathcal{ALC}$CTL in Weitl et al. (2009) is completed by defining the full syntax and semantics of $\mathcal{ALC}$CTL and by illustrating its use for representing the content and properties of a user manual.

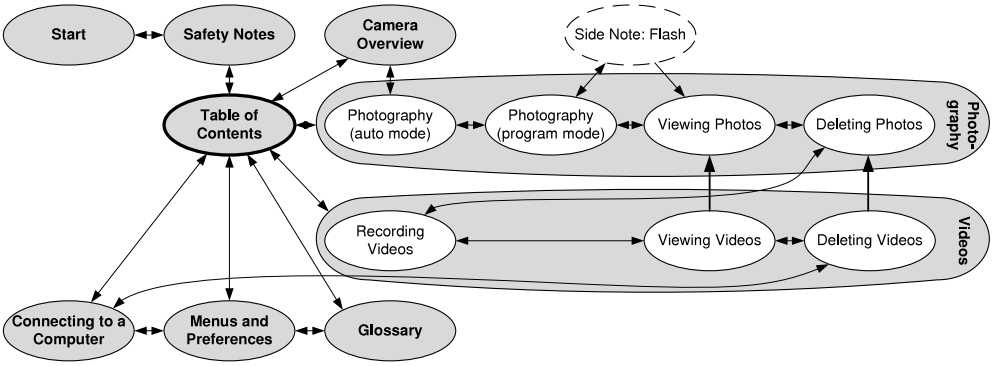

Fig. 1. Use case document: schematic view.

- It is illustrated how error reports are generated from model checking results and how errors are located in the document source files.

The rest of this paper is organized as follows. Section 2 introduces the case that has been used for evaluating the adequacy, performance, and effectiveness of the proposed methods and which serves, in a simplified form, as a running example throughout this paper. In addition, Section 2 outlines the major research problems addressed in this paper. Section 3 sketches our approach to solve the problems stated. Section 4 through 7 contain detailed descriptions of the major components of our approach and exemplify the overall processing based on our use case. Section 8 presents results obtained in experiments, Section 9 discusses related work, and Section 10 concludes the paper with a summary of achievements and an outlook on future research issues.

2. Problem definition and motivation

2.1. Use case and sample scenario

As a running example, we will use an HTML version of a technical documentation for a fictional digital camera *VDK1109*. The document has been created using several genuine documentations for digital cameras as guidelines. It represents their features and characteristics in condensed form. It is also anonymized in the sense that it does not contain any company or brand names. It consists of 15 chapters and subchapters, each located in a separate HTML file.

Fig. 1 shows a schematic view of the document. Each ellipse represents a chapter or a subchapter of the documentation (chapters are indicated in **bold face** and with a grey background color). The document begins with a title page (*Start*) and ends with a *Glossary*. There are links from each (sub)chapter to both its successor and its predecessor, as well as to the *Table of Contents* (most links to the *Table of Contents* are not depicted in Fig. 1 for reasons of clarity). The *Table of Contents* links to each chapter, but not to their subchapters (e.g., there is a link to *Photography*, but not to *Viewing Photos*). The *Photography* chapter contains a side note on *Flash*, an extension that can be read optionally. There are also cross references from two subchapters in the *Videos* chapter back to corresponding subchapters in the *Photography* chapter. It is possible to traverse the document from start to finish on different paths. There is one path that visits each (sub)chapter exactly once, but there are also paths that skip subchapters or chapters, and there are paths that visit (sub)chapters more than once. As the document structure contains cycles, not all paths are finite, and the number of paths is infinite.

The manual contains errors and inconsistencies that are typical for documents of this kind. Each type of error has been observed in real-world documentations on at least one occasion. One frequent problem is that not all technical terms and abbreviations are explained in the document (e.g., in the *Glossary*). Another problem is that critical warnings (or other critical sections) must not be bypassed by the reader. This means that there must not be any path through the document that skips chapters

containing such warnings. Specifically, side notes should only contain warnings that are also available in the main content of the document. A third kind of problem is related to the sequence of content on paths: no concept should be discussed before it has been introduced, and some chapters (like the *Safety Notes*) should be read before other chapters (like the subchapter on using the *Flash*). Finally, there is the problem of incorrect references. Links that point to an inappropriate target may confuse the reader even more than links whose targets do not exist. Links whose targets have no relation to at least one of the topics discussed in the referring chapter, are considered inconsistent. In the course of this article, we will discuss ways to detected these problems.

2.2. Problem statement

Consider the following consistency criteria:

C1 Any referred chapter covers topics *related to* those of the referring chapter.
C2 *Safety-related* topics are not skipped on *any path* through the document.
C3 Components in an overview are described *later on every path* through the document.

To verify criteria **C1** and **C2**, it is necessary to have (background) knowledge about which topics are related to each other and which topics are related to "safety". For checking **C2** and **C3**, it is necessary to consider all possible paths of reading through the document. **C3** is a requirement on the coherence of content along paths. It expresses a semantic relationship between overview sections and detailed descriptions on paths through the content. The following research questions arise:

Q1 How can information relevant to verifying criteria be extracted from the document as completely and precisely as possible, and how can it be combined with background knowledge to support the verification of criteria such as **C1** and **C2**?
Q2 How can the user be supported in setting up a generator of abstract yet sufficiently precise document models that can be verified efficiently?
Q3 How can a possibly infinite number of paths through a document be examined efficiently?
Q4 Which specification formalism is suitable to represent coherence criteria, i.e., requirements on semantic relations between objects on paths through the content?
Q5 How can the user be supported in formalizing path- and content-related criteria on documents?

In the course of this article, we will address questions **Q1** through **Q4**, with emphasis on **Q1** and **Q2**. In Weitl et al. (2009), our model for representing knowledge about documents has been defined. In this article, we rather focus on the *process* of generating knowledge representations and discuss how this process can be customized to the use case at hand. **Q3** and **Q4** have been addressed in Weitl et al. (2009), but are discussed in more detail in this article. Question **Q5** is an important issue within our approach, but will not be addressed in this paper because of space limitations. Instead, we refer the reader to Weitl et al. (2009), Jakšić and Freitag (2008, 2009).

2.3. Assumptions and prerequisites

In the context of this paper, a *document* is a set of one or more files that presents coherent information on a specific matter. A document can, for example, be a book, a documentation, a manual, a specification, lecture notes, or an e-learning script. A document consists of a set of one or more *fragments*. Fig. 2 shows a more detailed view of the *Photography* chapter, revealing its internal structure.

Fragments are the result of a sensible partitioning of a document into smaller parts, usually along structural lines (e.g., each file or each chapter becomes a fragment). A fragment can be broken down into (sub)fragments, which in turn can be fragmented again. We assume that each fragment is annotated with metadata describing it. Possible metadata annotations include *topic*, *structuralType* (the structural level of the fragment, e.g., chapter, section, etc.), *functionType* (the function of the fragment for the document, e.g., definition, example, introduction, hint, etc.), *author*, and intended *audience*. Fig. 3 shows a detailed view of the side note on *Flash*, how it is composed of subfragments, and what metadata is annotated to each fragment.

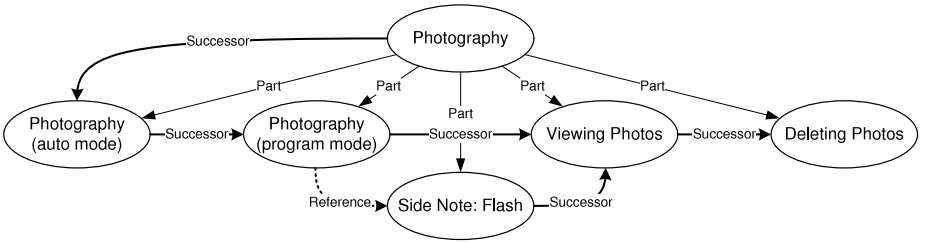

Fig. 2. Fragments of the *Photography* Chapter

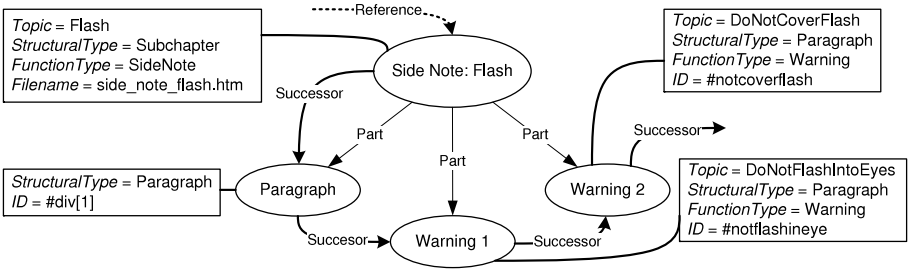

Fig. 3. Fragments of the Subchapter *Side Note: Flash*

There are three types of relationships between fragments of a document:

(1) Subfragments are *part* of their parent fragments.
(2) A fragment can cross-*reference* other fragments.
(3) Each fragment can have one or more *successor*s.

Both Fig. 2 and Fig. 3 show examples of these connectives. The parent fragment (e.g., *Photography* or *Side Note: Flash*) is linked via *part* relations to its subfragments. Fragments can *reference* other fragments (e.g., *Photography (program mode)* references *Side Note: Flash*). And fragments have one or more *successors* as indicated, for instance, by a "next page" link on a page. The successor relations between fragments from the same file (HTML page) are defined implicitly by the document order. Successors define a not necessarily linear structure that a reader is encouraged to follow (Schönberg and Freitag, 2009).

For the proposed framework to be applicable, documents must fulfill five minimum requirements:

D1 The document needs to be *fragmentable*, i.e., it must be divisible into fragments.
D2 The set of these fragments needs to cover the entire document.
D3 The fragments must not overlap, but a fragment may be completely contained in another fragment, i.e., as a subfragment.
D4 The fragmentation must be defined by a set of rules or criteria, so that it can be automatically detected during information extraction.
D5 The document must either contain semantic annotations, or background knowledge about the document and its domain of discourse must be available.

The requirements **D1** through **D3** are met by semi-structured document formats such as XML or HTML, so that effective information extraction processes can be applied in these cases. For instance, the XML-based standards for technical documentations DocBook (Walsh et al., 2005) and DITA (Priestley and Hackos, 2005) are well supported. In addition, we applied our framework successfully to LaTeX and Microsoft Word documents, by converting them into XML files in a preprocessing step.

3. Overall verification framework

In the context of the Verdikt research project, a general framework for document verification has been developed (Weitl et al., 2009). The major components of the framework are ① information

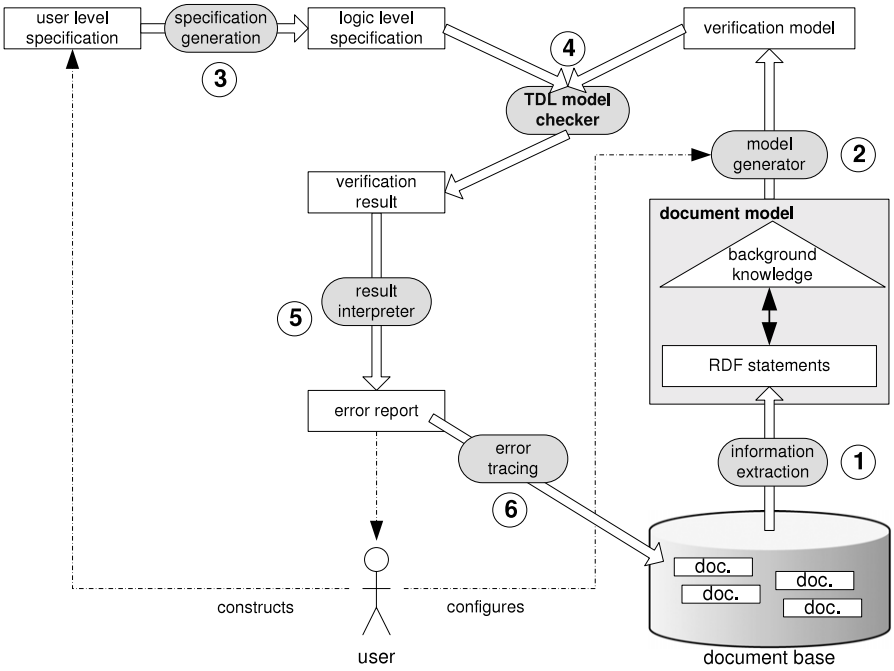

Fig. 4. Overview of the verification framework.

extraction (Fig. 4 rhs bottom), ② model generation (rhs top), ③ specification generation (lhs top), ④ model checking (center top), ⑤ result interpretation (center), and ⑥ error tracing (rhs bottom).

The first component (*information extraction*) reads all relevant data from the source document and stores it for further processing (→ **Q1** in Section 2.2: Problem Statement). The *model generator* creates a verification model suitable for model checking (→ **Q2**). The *specification generation* component supports the user in specifying criteria to be applied to the input document (→ **Q5**) and translates *user level specifications* into *logic level specifications* based on a temporal description logic (TDL) (→ **Q4**). The *TDL model checker* checks verification models against TDL formulae (→ **Q3**). The *result interpreter* generates meaningful error reports from counterexamples returned by the model checker in the case of violated formulae. *Error tracing* highlights the discovered errors in the source documents.

We propose TDL as a fundamental formalism for representing consistency criteria on documents because TDL allows for the concise representation of consistency criteria evaluated along some or all paths through the document the reader can follow (→ **Q4**). These paths are subsequently called *reading paths*. For instance, *Start* → *Safety Notes* → *TOC* → *Camera Overview* is the prefix of a reading path in the document depicted in Fig. 1. Since formalizing criteria in TDL is difficult, we added a user level specification method based on specification patterns, examples, and visualizations (Jakšić and Freitag, 2008; Weiß, 2009). The user is guided by an incremental specification process in constructing a specification that is both intuitive to understand and unambiguous (→ **Q5**).

The general approach and the proposed data structures for knowledge representation and property specification have been presented in Weitl et al. (2009). In this article, we detail the components ① and ②, and sketch the components ⑤ and ⑥ of Fig. 4.

A typical use case of the framework runs as depicted in Fig. 4. First, a set of consistency criteria to be applied to a document is constructed (top left corner). These user level specifications are converted into logical formulae for the model checker (top center). From a document (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 chooses what metadata is relevant for the current specification 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 verifies the verification model

against the logic level specification and returns verification results, from which an error report is distilled and presented to the user (bottom center). The error report refers to the original user level specification and to the source document rather than to the logic level verification and verification model. Source references added to the document metadata enable the re-mapping of model checking results onto user level data.

4. Information extraction

Extracting the relevant data from a document and preparing it for model checking is done in two stages: first, the document structure and content data is extracted from the source files. The resulting document model represents the data and structure that can be used for verifying criteria. This model has a graph structure and includes properties for nodes that represent fragments. It is represented in RDF (Klyne and Carroll, 2004), and it can be stored in RDF-capable database systems for future reuse.

The second step is to define a view on the document model that is most advantageous for verifying a selected specification, and to materialize the view to obtain a verification model. The latter step is described in more detail in Section 5. Technical details about the document model can be found in Schönberg and Freitag (2009). Question **Q1** (cf. Section 2.2) is addressed in this section, while **Q2** is addressed in the next section.

The *information extraction process* (cf. Fig. 4 rhs bottom) makes use of the internal representation of a document's source files. For XML-based formats, the DOM tree is processed, starting at the root DOM object and traversing through the tree structure down to its leaves. Using pattern matching, the DOM objects are selected according to conditions based on their name and context, executing predefined actions that have two main objectives: to insert new DOM objects (usually the object's child elements) into the process, and to successively construct the annotated graph structure that represents the document model. For the latter objective, background knowledge is employed to support metadata recognition, to control the mapping of metadata and structural data onto the RDF graph of the document model, to cope with imprecise terminology, and to deal with circumstances that depend on the application or content domain. Both the extraction process and the usage of background knowledge will be described in more detail below.

A rule language is used to specify the extraction logic in terms of patterns and actions, which are then processed by a rule engine. Such a language needs to meet the following requirements:

- It must be possible to cover differently structured documents as well as multiple file formats with a single set of rules, like HTML with embedded SVG images.
- Access to external tools and resources like preprocessors to clean up HTML code, dictionaries of important terms, ontologies of the domain of discourse, or reasoners for deriving implicit knowledge about the document must be possible.
- Rule specialization must be supported to extend generic rules for specific cases.

We define the information extraction logic using the JBoss Drools language (Proctor, 2007), which is an object-oriented extension of production rules. Production rules are pairs of condition and action definitions, matched against a set of Java objects – the so-called *fact base* – by the Drools rule engine. For objects that meet the condition, the action will be performed, which may insert, remove, or modify objects in the fact base. To identify objects that match to one or more rules, the Drools engine adopts the Rete algorithm (Doorenbos, 1995). When more than one rule matches a given object, or when more than one object matches to rules, it uses a strategy based on rule priority and LIFO ordering to resolve the conflict by creating a custom execution order (Bost et al., 2007).

The Drools rules meet the first two requirements because they are built upon the Java object model, and because they allow access to external Java methods and libraries in both the condition and the action part of rules. They satisfy the third requirement by allowing multiple rules with different conditions to match the same object.

Example 1 (*Rule Specialization*)**.** Consider a JBoss rule R_1 whose condition matches Java objects of class C_1. Then a rule R_2 can be seen as an extension to rule R_1 if the condition of R_2 entails the condition of rule R_1, for instance if R_2 matches objects of a subclass C_2 of C_1. Then objects that are instances of

class C_1 but not of class C_2 only match rule R_1, while objects of the more specific class C_2 will match both rules. Thus, the action of rule R_2 will be performed in addition to the action of rule R_1 for objects of class C_2, i.e., both rules are executed.

4.1. Background knowledge

The quality of the verification results depends on the precision and recall of the information extraction process. In our context, *recall* is the percentage of discovered constructs (e.g., definitions, successor relations, or terms) of a certain type relative to their actual number in the document. *Precision* is the percentage of constructs whose type has been detected correctly relative to the total number of detected constructs of a certain type (cf. Remark 1). Both values can be used to evaluate the quality of the verification model used in the verification process. Background knowledge can enhance the recall of information extraction by providing additional 'clues' to the extraction process, and it can help to improve precision by reducing best-guess approaches.

Example 2 (*Impact of Precision and Recall on the Verification Process*)**.** The criterion "For every definition, there has to be a matching example" cannot be checked reliably if some definitions have not been detected (poor recall). In this case, matching examples that are missing may go unnoticed, which results in false positives. False negatives may arise from paragraphs being recognized as definitions by mistake (poor precision).

We distinguish three kinds of background knowledge:

B1 Knowledge about the document format. This is used to define the general outline of the extraction rules.

B2 Knowledge about specific customizations or parameterizations of the document format. Examples are predefined templates in Microsoft Word, stylesheet classes in HTML, and keywords like 'definition' or 'example'. These templates, style classes, and keywords are syntactical indicators of semantic information about fragments, such as structure or function type (cf. Section 2.3: Assumptions and Prerequisites). Their meaning is represented by a mapping onto a controlled vocabulary (cf. Example 3). This mapping helps to increase the recall of extraction rules. The use and meaning of syntactical indicators depend on the *application domain*, i.e., the document's context.

B3 Knowledge about the content of the document, such as important terms or relations between terms. This knowledge is dependent on the document's *domain of discourse*. Extraction rules query this domain knowledge at runtime to detect relevant topics and content relations between document fragments.

Remark 1 (*Computing Precision and Recall*)**.** Factors that can introduce errors into the extraction results are incomplete detection of relevant resources, and erroneous detection of resources that should have been ignored. In our case, precision and recall are computed by comparing the extraction results of the document in its regular form with the extraction results from a version of the document with a manually applied complete tagging of all relevant resources.[2] For instance, the recall for our use case before the introduction of domain knowledge can be calculated as

$$\frac{|\{\text{all relevant resources}\} \cap \{\text{resources found}\}|}{|\{\text{all relevant resources}\}|} = \frac{77}{104} = 0.74$$

Example 3 (*Syntactical Indicators*)**.** Fig. 5 shows an example of a mapping between syntactical indicators and controlled vocabulary. On the left, a set of indicators like keywords or CSS class names is listed. On the right, a controlled vocabulary in the form of a taxonomy is given. The dotted arrows between the two define the mapping. For example, if the keyword "illustration:" is detected in the

[2] These results are obtained without using background knowledge, which is not required since all relevant resources have already been tagged and thus do not need to be inferred. The manual tagging ensures that no resources remain undetected, while the exclusion of background knowledge which might contain errors ensures that no superfluous resources are included in the result.

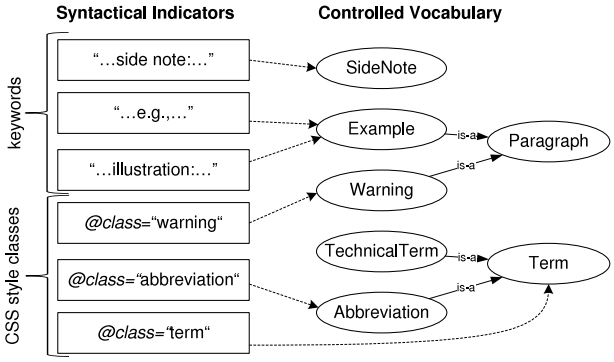

Fig. 5. Mapping between syntactical indicators and controlled vocabulary.

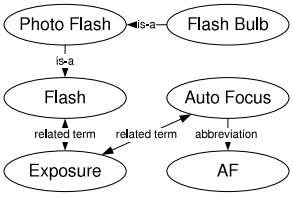

Fig. 6. Content domain ontology.

title of a paragraph, this paragraph is annotated with the function type "Example". The keywords can easily be extended to regular expressions to match phrases like "illustration 6.2". The knowledge about warnings is used as data for the role *safety-related* in criterion **C2**. In the presented use case, the mapping was represented as a set of indicator/vocable pairs in an XML file.

Remark 2 (*Obtaining Background Knowledge*)**.** If formalized knowledge does not exist for a domain or topic, obtaining it can introduce additional overhead. This overhead can be reduced as follows: Knowledge about the document format and about its customizations can be derived from format specifications. Domain knowledge is obtained from ontologies that cover the topic domains, from dictionaries of specialized terms like abbreviations, from lexical databases such as WordNet (Miller, 2006), or from ontologies extracted from social collaboration projects (Schönberg et al., 2010). Domain knowledge can also be extended during the extraction process, for example by dynamically adding terms extracted from documents in the domain to a dictionary of terms.

Example 4 (*Domain Knowledge*)**.** Fig. 6 shows an example of knowledge about the domain of discourse. It represents the interrelationships of some topics about *Flash*. This knowledge is used as data for the role *related to* in **C1**. In the presented use case, it was sufficient to represent this information by means of three XML files, one for each of the relations *is-a*, *related term*, and *abbreviation*. In more complex settings, OWL is preferable since it enables inference services such as classification and consistency checks.

Remark 3 (*Word Forms*)**.** When matching terms in a document with a controlled vocabulary, stemming algorithms and dictionaries help recognize terms that are not in their basic form (e.g., an occurrence of 'Photo Flashes' in the text should still be recognized as an instance of the term 'Photo Flash').

4.2. Information extraction process

The general rule-based information extraction process using JBoss Drools works as follows. First, objects are inserted into the fact base of the Drools rule engine. Then, the pattern matching algorithm (Rete) of the engine determines the set of rule/object pairs that match (i.e., where the object satisfies all conditions of the rule), and that have not yet been processed. These pairs are called activations. If there

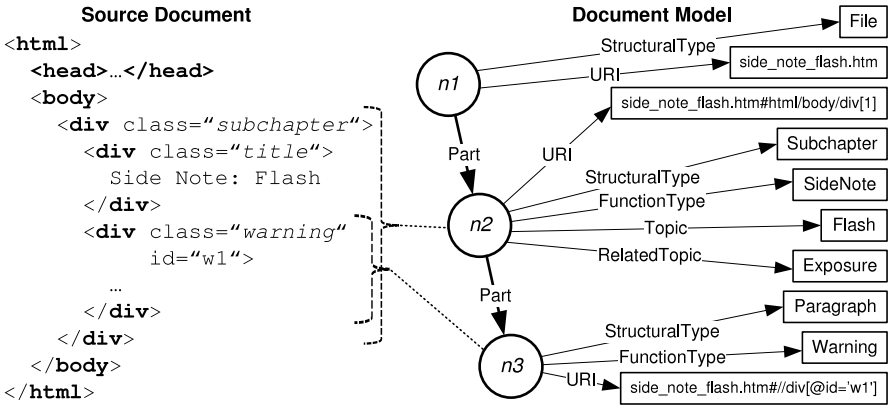

Fig. 7. Information extraction: example.

is more than one activation, a conflict resolution algorithm is triggered that selects a single activation. Finally, the selected activation is processed: the actions associated with its rule are performed. After that, the matching algorithm continues to detect new activations, until no activations remain (Bost et al., 2007).

(Meta-)information is extracted by processing the DOM tree of the XML source files of the documents to be processed. The objects being inserted into the fact base of the rule engine are instances of the processing context, representing the status of the currently processed DOM element.

Example 5 (*Information Extraction*)**.** Fig. 7 shows an example of how the structure and content of a clipping of a source file (lhs) is represented in the document model (rhs). Each node (*n1* through *n3*) on the rhs represents a fragment of the file.

To start the extraction process, a new *context object* containing the root element of the file (`<html>`) and the file's URI is inserted into the fact base, triggering one or more rules that match this context object. The actions of the selected rule for contexts of root elements create a new node *n1* in the document graph (Fig. 7 center top). Node *n1* is annotated with a 'StructuralType' property 'File', and with the 'URI' property of the current context object. Finally, the rule creates a new context objects for each of the child elements (`<head>` and `<body>`) and inserts them into the fact base. These context objects also contain a reference to their parent fragment *n1* in the document model.

Inserting new context objects into the fact base of the engine triggers new rules, which successively process the respective elements of the DOM tree. A rule matching contexts of `<body>` elements does nothing but insert contexts of its child elements (`<div>`) into the engine. Different rules are used for processing `<div>` elements, depending on the values of their class attributes. The rule processing `<div>` elements of class "subchapter" creates node *n2* and annotates it with both a URI property and with the respective structural type (rhs center). It also creates the "Part" connection between *n1* and *n2*, knowing from the context that *n1* is the current parent fragment. It then inserts the context objects of the element's children into the fact base, each containing a reference to parent node *n2*.

The rule matching context objects of `<div>` elements whose class is "title" processes the title of the parent fragment, which is *n2* in the given example. First, it checks the title against the list of syntactical indicators (cf. Fig. 5) and detects, in this case, the key phrase "side note:", which corresponds to the function type "SideNote" in the controlled vocabulary. As a result, the current fragment is annotated with this function type and the remaining part of the title "Flash" is annotated as a "topic" property to *n2*. Now, the content domain ontology (Fig. 6) is checked for terms related to the topic "Flash". The term "Exposure" is found and annotated as a related topic to *n2* (rhs center).

Yet another rule matches context objects of `<div>` elements whose class is "warning". This rule creates a new node *n3* and annotates it with the function type property "Warning" (rhs bottom) according to the mapping of syntactical indicators in Fig. 5. Similar to *n1* and *n2*, *n3* is also annotated with a structural type and a 'URI' property, and inserted as a part of *n2* into the document model.

The following clipping shows sample code whose general structure is common to many of the extraction rules:

```
1  rule "traverse document"
2  when   $context : Context(element.tagName == "div" && some attributes... )
3         $backgroundknowledge : Knowledgebase(name == "terms")
4  then   Fragment $frag ← new Fragment($context);
5         $context.parentFragment.addPart($frag);
6         $frag.structuralType = "Subchapter";
7         $frag.addTerms($backgroundknowledge.match($context.element.text));
8         ▷ wrap child elements into context objects and insert them into fact base
9  end
```

The rule condition (line 2) contains the tag name of an XML element and optionally one or more attribute values. $*context* is a variable that is instantiated with an object from the fact base matching the condition pattern after the colon. "Context(…)" indicates that only objects of type `Context` can be matched, while the expression within parentheses defines conditions on the attributes of the context object, e.g., to check the tag name of the DOM element. Line 3 matches a background knowledge object that is used in the conclusion of the rule.

Lines 4 through 8 contain the actions of the rule that are executed for the matching $*context* object. First, rule-dependent actions like creating nodes for fragments in the document model (line 4), defining the part-of hierarchy of fragments (line 5), or annotating properties such as the structural type to nodes (line 6, cf. Example 5) are performed. Then, the text content of the current element is matched against the known terms in the background knowledge, and all detected terms are assigned to the fragment node (line 7). Finally, all child elements of the current element are wrapped in individual context objects and inserted into the Drools fact base (line 8). Rules that match the pattern for link elements (<a> tags in HTML) add a new context object to the fact base that contains the root element of the referenced file. By keeping track of already processed files in a global list it is ensured that a source file is not processed more than once.

Remark 4 (*Termination of the Information Extraction Process*)**.** The information extraction process terminates if the extraction rules adhere to the following conditions:

(1) rules only insert new context objects of elements that are either children of the current element, or that are root elements of other files.
(2) rules matching root elements of files that have already been processed do not write to the fact base.

Advantages of using rules to specify and control the extraction process include a decreased complexity of the extraction logic, because a rule language promotes a clear separation between condition and action, and a concise and explicit notation for both. This also reduces the effort required for maintenance and for adapting the extraction logic to new scenarios. This is a distinct advantage over the Java-based approach presented in Schönberg and Freitag (2009), while maintaining its flexibility. A rule-based specification also removes the *need* to manually define the control flow, while maintaining the *possibility* to do so when appropriate. This, combined with improved runtime results, is an advantage over the XQuery-based approach presented in Schönberg et al. (2009).

5. Model generation

The information extraction process as described in Section 4 generates a graph consisting of RDF statements (Fig. 4 rhs center). The RDF-representation describes the content and structure of the document as detailed and complete as possible to support the verification of a wide variety of possible specifications.

In the *model generation* process, the metadata relevant for verifying a *given* specification is selected and a *verification model* is constructed that fits to the semantics of the temporal logic $\mathcal{ALC}$CTL — the logic level specification formalism of our approach (Fig. 4 rhs top). Hence, the verification model can

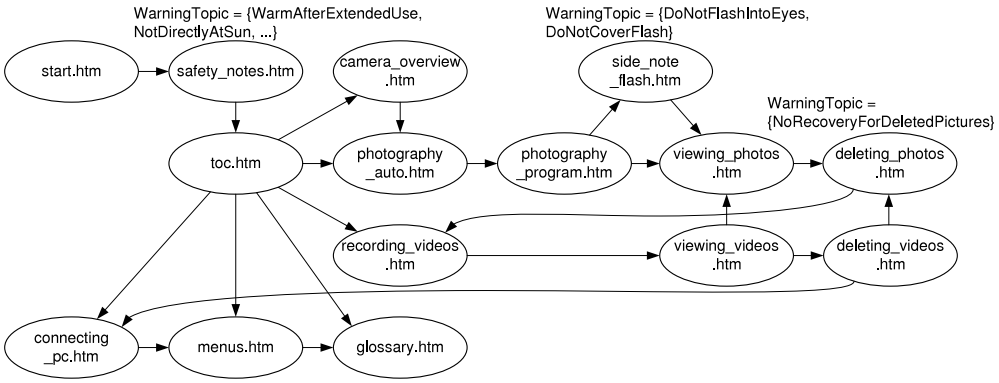

Fig. 8. Generated verification model (role annotations omitted).

be regarded as a specific *view* on the RDF graph of the document to enable the efficient verification of a given set of criteria.

5.1. Verification model

Fig. 8 shows a simplified version of the verification model of our sample scenario.

The verification model is an annotated directed graph whose nodes represent fragments of the document at a distinct structural level (e.g., subchapters). An edge between node a and node b in the verification model expresses that after having read a it is sensible to go to b. Hence, paths in the verification model represent sensible ways of reading the document (cf. Fig. 1). Note that not all links in a document must be represented as edges in the verification model. For instance, links back to the table of contents in Fig. 1 are omitted in the verification model of Fig. 8, since they are not interpreted as a recommendation of how to continue reading.

Remark 5 (*Types of References*)**.** Apart from the references between chapters that describe the main reading path, there are other types of references such as cross references (e.g., from *viewing_videos.htm* to *viewing_photos.htm*), references to side notes such as from *photography_program.htm* to *side_note_flash.htm*, or references to the table of contents. These types are distinguished during information extraction by using background knowledge about the documents, e.g., all references to the table of contents are labeled "Content" in the source document, or references along the main reading path are labeled "Next". Types are represented in the RDF graph as specializations of references, making use of the RDF Schema "subPropertyOf" feature: for example, the reference types "Successor", indicating a successor along the main reading path, and "TocReference" are both specializations of the general "Reference" role. An RDF Schema definition of the roles used in the RDF graph is provided for querying the graph.

Within the proposed framework of temporal description logic, the nodes of the verification model are interpreted as *states* and the edges as *transitions* of a temporal structure (cf. Definition 2). States of the verification model are annotated with representations of the content of the corresponding document fragments. There are two different kinds of annotations: *concept* annotations and *role* annotations.

Concept annotations represent local properties of states by assigning sets of instances to concepts. For example, the state *side_note_flash.htm* in Fig. 8 has the annotation

$$\mathit{WarningTopic} = \{\mathit{DoNotFlashIntoEyes}, \mathit{DoNotCoverFlash}\}$$

This expresses that *DoNotFlashIntoEyes* and *DoNotCoverFlash* are instances of the concept *WarningTopic* and represents that the respective fragment (here: *Side Note: Flash*) contains warnings on "do not flash into eyes" and on "do not cover the flash".

Role annotations (omitted in Fig. 8 for clarity) represent semantic relations of objects in certain states. For instance, the following is a role annotation of the state *side_note_flash.htm* representing that the corresponding fragment covers both the topics *DoNotFlashIntoEyes* and *DoNotCoverFlash*.

$$\begin{aligned} \mathit{hasTopic} \;=\; \{&(\mathit{side\_note\_flash.htm}, \mathit{DoNotFlashIntoEyes}),\\ &(\mathit{side\_note\_flash.htm}, \mathit{DoNotCoverFlash})\} \end{aligned}$$

Remark 6 (*Concept and Role Annotations*)**.** Concepts and roles such as *WarningTopic* or *hasTopic* form the basic terms upon which formal specifications are built (see Section 6.1). To be sufficient for verifying a given specification S, the verification model must contain annotations of each concept and role occurring in S. For reasons of efficiency, it should not contain annotations of concepts and roles that do not occur in S.

5.2. Model generation process

To generate a verification model for the verification of a given specification, it needs to be decided which fragments in the document model form the set of *states*, which relations between fragments form the *transitions* between states, and which *concept* and *role annotations* are constructed from the document model. These decisions have a large impact on the type of criteria that can be verified, on the precision and usefulness of error reports obtained from model checking, and on the performance of the verification process. To support the construction of verification models tailored to a wide range of different application scenarios and specifications, verification models are generated by executing a set of interrelated SPARQL queries (Prud'hommeaux and Seaborne, 2008), which can be adapted flexibly to the verification scenario at hand. However, the model designer does not have to write SPARQL queries directly. Instead, a graphical user interface provides available options for generating the transition relations based on the extracted information and background knowledge about different types of references in the document (cf. Remark 8).

The subsequent sample queries illustrate the general model generation process based on SPARQL. They are, for the sake of clarity, simplified variants of the queries used to generate the verification model of Fig. 8. The role names used in the queries are defined in RDF Schema for the RDF graph.

5.2.1. States

Fragments that represent states are determined by a SPARQL query on the RDF graph.

Example 6 (*State Query*)**.** The query "*Select all objects that are part of files and whose structural type is 'Subchapter'*" is expressed as a SPARQL query as follows. The result is the set of subchapters of the document as a set of states of the verification model.

```
SELECT ?p WHERE {
?f StructuralType 'File'. ?f Part ?p. ?p StructuralType 'Subchapter'}
```

Remark 7 (*State Query*)**.** As SPARQL does not support recursive queries (Prud'hommeaux and Seaborne, 2008), following deeper part-of structures is usually more verbose than depicted here.

5.2.2. Transitions

In a second step, the transition relation is constructed by querying the successor states of each state returned by the state query.

Example 7 (*Transition Query*)**.** The query "*Select all fragments that are successors of the fragment whose URI is x*" is expressed as a SPARQL query as follows. The result is the set of successor nodes of the state that is identified by URI *x*.

```
SELECT ?s WHERE { ?f URI x. ?f Successor ?s.}
```

The query is executed for each state returned by the state query in the first step. In each instance of the query, the parameter *x* is replaced by a URI that has been generated in the information extraction process and identifies a certain state. To obtain the transition relation of Fig. 8, the targets of cross references are retrieved by an analogue query.

5.2.3. Concept and role annotations

For each concept and role to be included in the annotations of states, a SPARQL query is defined that retrieves their instances and pairs of instances, respectively. Care must be taken that for each concept and role occurring in a given specification, a suitable query is defined. Similar to generating the transition relation, each concept and role query is instantiated and executed for each state returned by the state query.

Example 8 (*Concept Query*)**.** The query "*Select all topics of warnings which are part of fragment x*" is expressed as a SPARQL query as follows. The result are the instances of the concept *WarningTopic* for the state that is identified by URI *x*.

```
SELECT ?t WHERE {
?f URI x. ?f Part ?p. ?p FunctionType 'Warning'. ?p Topic ?t}
```

Example 9 (*Role Query*)**.** The query "*Select all pairs whose first component is a URI of a part of fragment x and whose second component is a topic of the part*" is expressed as a SPARQL query as follows. The result are the pairs of instances of the role *hasTopic* for the state that is identified by URI *x*:

```
SELECT ?u, ?t WHERE { ?f URI x. ?f Part ?p. ?p URI ?u. ?p Topic ?t.}
```

Remark 8 (*State, Successor, Concept, and Role Queries*)**.** As already mentioned, the end user does not need to define the SPARQL queries for generating the verification model directly, but can make use of a GUI. This GUI ensures that only queries that conform to the given RDF model can be created, and queries can be created without knowledge about the underlying selector language (SPARQL). The corresponding SPARQL constructs are then generated automatically. This is achieved by configuring the program with both the SPARQL syntax grammar and with the predicate vocabulary of the RDF model.

End users can also make use of a pattern-based approach for specifying criteria (Jakšić and Freitag, 2010).

6. Formal specification

These criteria refer to semantic relationships in the content on paths through the document. Since the number of paths to consider usually grows exponentially in the document size or is even infinite, standard methods for document validation such as Schematron (Jelliffe, 2002) do not scale to application-relevant problem sizes. Propositional temporal logics are well suited for capturing and verifying path-based properties. Their expressiveness, however, is insufficient for representing semantic relationships of objects on paths. We propose the temporal description logic $\mathcal{ALC}$CTL (Weitl, 2008) for representing content- and path-related consistency criteria. $\mathcal{ALC}$CTL is a combination of the description logic $\mathcal{ALC}$ (Baader and Nutt, 2003) and the branching time temporal logic CTL (Emerson, 1990). $\mathcal{ALC}$ is expressive for representing relationships among document parts and their topics. CTL is expressive for representing properties of paths through the document. The combination of description logics and temporal logics provides high expressiveness for content-related criteria w.r.t. reading paths.

6.1. Syntax and semantics of $\mathcal{ALC}$CTL

$\mathcal{ALC}$CTL is a language built upon the following application-dependent symbols which consists of

- a set of atomic concepts **C**. Each atomic concept $C \in \mathbf{C}$ represents a set of objects. For instance, the concept *Warning* $\in \mathbf{C}$ may represent the set of "warnings" contained in (a part of) the document.
- a set of atomic roles **R**. Each atomic role $R \in \mathbf{R}$ represents a binary relation. For instance, the atomic role *hasTopic* $\in \mathbf{R}$ may represent the binary relation between document parts and their topics.

Definition 1 (*Syntax of $\mathcal{ALC}$CTL*)**.** Let **C** be a set of atomic concepts and **R** be a set of atomic roles. The set of $\mathcal{ALC}$CTL *concepts* on **C**, **R** is the minimal set of *state* concepts generated by the following rules:

(SC0) $\top$ and $\bot$ are state concepts;
(SC1) each atomic concept $A \in \mathbf{C}$ is a state concept;
(SC2) if C, D are state concepts then so are $C \sqcap D$, $C \sqcup D$, $\neg C$;
(SC3) if C is a state concept and $R \in \mathbf{R}$ an atomic role then $\forall R.C$ and $\exists R.C$ are state concepts;
(PC1) if C, D are state concepts then $\mathsf{X}C$ and $C \,\mathsf{U}\, D$ are *path* concepts;
(SC4) if C is a path concept then $\mathsf{E}C$, $\mathsf{A}C$ are *state* concepts.

The set of $\mathcal{ALC}$CTL *formulae* is the minimal set of *state* formulae generated by the following rules:

(SF0) *true* and *false* are state formulae;
(SF1) if C, D are $\mathcal{ALC}$CTL concepts then $C \sqsubseteq D$ and $C \doteq D$ are state formulae;
(SF2) if p, q are state formulae then so are $p \wedge q$, $p \vee q$, and $\neg p$;
(PF1) if p, q are state formulae then $\mathsf{X}p$ and $p \,\mathsf{U}\, q$ are *path* formulae;
(SF3) if p is a path formula then $\mathsf{E}p$ and $\mathsf{A}p$ are *state* formulae.

Remark 9 (*Syntax of* $\mathcal{ALC}$CTL)**.**

- We use the following abbreviations as defined in Emerson (1990): Let α, β be either $\mathcal{ALC}$CTL concepts or formulae. $\mathsf{F}\alpha$ ("eventually α") abbreviates $\top \,\mathsf{U}\, \alpha$ if α is a concept and *true* $\mathsf{U}\, \alpha$ if α is a formula; $\mathsf{AG}\alpha$ ("all paths globally α") abbreviates $\neg\mathsf{EF}\neg\alpha$, $\mathsf{EG}\alpha$ ("some path globally α") abbreviates $\neg\mathsf{AF}\neg\alpha$, $\mathsf{A}(\alpha \,\mathsf{B}\, \beta)$ ("all paths α before β") abbreviates $\neg\mathsf{E}((\neg\alpha) \,\mathsf{U}\, \beta)$, and $\mathsf{E}(\alpha \,\mathsf{B}\, \beta)$ ("some path α before β") abbreviates $\neg\mathsf{A}((\neg\alpha) \,\mathsf{U}\, \beta)$.
- The binding precedence of connectives introduced above is as follows: the connectives used as concept constructors have higher binding power than the connectives used to build formulae. The precedence of concept constructors from highest to lowest binding power is as follows: path operators A, E, temporal operators F, G, X, U, B, non-temporal concept constructors $\forall$, $\exists$, $\neg$, $\sqcap$, $\sqcup$. The binding precedence of connectives in formulae is as follows: path operators A, E, temporal operators F, G, X, U, B, non-temporal formula connectives $\sqsubseteq$, $\doteq$, $\neg$, $\wedge$, $\vee$.

To get an intuition for the semantics of the connectives introduced in Definition 1, consider the following consistency criterion: "Topics, which are subject to warnings, are included in the glossary that is reachable on some path".
This criterion can be represented by the $\mathcal{ALC}$CTL formula

$$\mathit{WarningTopic} \sqsubseteq \mathsf{EF}\, \exists \mathit{topicOf}.\mathit{Glossary}$$

"Every topic of a warning ($\mathit{WarningTopic} \sqsubseteq$) is on some path eventually (EF) topic of a glossary ($\exists \mathit{topicOf}.\mathit{Glossary}$)"

WarningTopic and *Glossary* are atomic concepts representing the sets of warnings and glossaries contained in a unit of the document. *topicOf* is an atomic role representing a binary relation between topics and document units. The quantified role expression $\exists \mathit{topicOf}.\mathit{Glossary}$ (cf. (SC3) in Definition 1) represents the set of topics which are topic of some glossary. The CTL operator $\mathsf{EF}\, p$ (some path eventually p) expresses that property p holds eventually on some path. Thus, $\mathsf{EF}\, \exists \mathit{topicOf}.\mathit{Glossary}$ represents the set of topics which are on some path eventually topic of some glossary, i.e., for which a glossary entry is *reachable* from the current document unit. The $\mathcal{ALC}$ connective $\sqsubseteq$ expresses a subsumption relationship: Each instance of *WarningTopic* is an instance of $\mathsf{EF}\, \exists \mathit{topicOf}.\mathit{Glossary}$.

Definition 2 (*Temporal Structure*)**.** $\mathcal{ALC}$CTL formulae are evaluated on *temporal structures* $M = (S, R, I)$ where

- S is a non-empty set of states.
- $R \subseteq S \times S$ is a *left-total* binary relation on S assigning to each state in S at least one successor in S.
- Let LI be the set of $\mathcal{ALC}$ interpretations (Baader and Nutt, 2003). Then I is a function $S \to LI : I(s) = (\Delta^I, \cdot^{I(s)})$ associating each state $s \in S$ with a DL interpretation $I(s) = (\Delta^I, \cdot^{I(s)})$ where Δ^I is a (state independent) set of objects, the *interpretation domain*, and $\cdot^{I(s)}$ is a function assigning to each atomic concept $A \in \mathbf{C}$ a set $A^{I(s)} \subseteq \Delta^I$ and to each atomic role $R \in \mathbf{R}$ a set $R^{I(s)} \subseteq \Delta^I \times \Delta^I$.

Verification models of documents are temporal structures as illustrated by the subsequent example.

Example 10 (*Temporal Structure*)**.** The verification model shown in Fig. 8 is a temporal structure (S, R, I) where

$$\begin{aligned} S &= \{start.htm, safety\_notes.htm, toc.htm, camera\_overview.htm, \\ &\quad photography\_auto.htm, \ldots, glossary.htm\} \\ R &= \{(start.htm, safety\_notes.htm), (safety\_notes.htm, toc.htm), \\ &\quad (toc.htm, camera\_overview.htm), (toc.htm, photography\_auto.htm), \ldots\} \\ \Delta^I &= \{WarmAfterExtendedUse, NotDirectlyAtSun, DoNotFlashIntoEyes, \\ &\quad DoNotCoverFlash, NoRecoveryForDeletedPictures, \ldots\} \end{aligned}$$

and where $\cdot^I$ is a state-dependent interpretation of atomic terms such as *WarningTopic* with

$$\begin{aligned} WarningTopic^{I(safety\_notes.htm)} &= \{WarmAfterExtendedUse, NotDirectlyAtSun, \ldots\} \\ WarningTopic^{I(side\_note\_flash.htm)} &= \{DoNotFlashIntoEyes, DoNotCoverFlash\} \\ WarningTopic^{I(deleting\_photos.htm)} &= \{NoRecoveryForDeletedPictures\} \end{aligned}$$

S is the set of nodes and R is the set of edges of the verification model as depicted in Fig. 8. (*start.htm*, *safety_notes.htm*) $\in R$, for instance, represents that after having read *start.htm* it is recommended to proceed to *safety_notes.htm*.

The interpretation domain Δ^I represents the set of objects such as document parts and their topics, which appear in annotations of the verification model and which are constrained by $\mathcal{ALC}$CTL formulae. In the verification model of Fig. 8, these are the instances of concept *WarningTopic* in the states *safety_notes.htm*, *side_note_flash.htm*, and *deleting_photos.htm*.

The interpretation function $\cdot^I$ represents the annotation of the states of the verification model with local properties such as the topics of warnings by assigning values to atomic terms like *WarningTopic*. The value of interpretation $WarningTopic^{I(safety\_notes.htm)}$, for instance, corresponds to the annotation of concept *WarningTopic* in state *safety_notes.htm* of the verification model in Fig. 8.

Remark 10 (*Temporal Structure*)**.** The semantics of temporal logics requires R to be left-total, i.e., each state $s \in S$ has a successor $s' \in S$ such that $(s, s') \in R$. To ensure that the successor relation of verification models is left-total, reflexive edges may be added to the transition relation. For instance, for the finial state *glossary.htm* of the verification model depicted in Fig. 8, we add (*glossary.htm*, *glossary.htm*) to R.

$\mathcal{ALC}$CTL path concepts and formulae are evaluated on *fullpaths*:

Definition 3 (*Fullpath*)**.** Let (S, R, I) be a temporal structure and $s \in S$ a state. Then $FP_s := \{(s_0, s_1, \ldots) | s_0 = s \wedge \forall i \in \mathbb{N}_0 : (s_i, s_{i+1}) \in R\}$ is the set of *fullpaths* starting in state s.

Example 11 (*Fullpath*)**.** (*start.htm*, *safety_notes.htm*, *toc.htm*, *camera_overview.htm*, ...) $\in FP_{start.htm}$ is the prefix of a fullpath starting in state *start.htm* in the temporal structure depicted in Fig. 8.

Definition 4 (*Semantics of* $\mathcal{ALC}$CTL)**.** Given is a temporal structure $M = (S, R, I)$, $I : S \to LI : I(s) = (\Delta^I, \cdot^{I(s)})$ as defined above, a set of atomic concepts $\mathbf{C}$ and roles $\mathbf{R}$, a state $s \in S$, and a fullpath $x = (s_0, s_1, s_2, \ldots)$. The *interpretation* of $\mathcal{ALC}$CTL concepts w.r.t. M, s and w.r.t. M, x, in symbols $(M, s)(C)$, $(M, x)(C)$ is as follows:

$$\begin{aligned} &\text{(SC0)} && (M,s)(\top) = \top^{I(s)} =_{def} \Delta^I \\ &&& (M,s)(\bot) = \bot^{I(s)} =_{def} \emptyset; \\ &\text{(SC1)} && \text{for } A \in \mathbf{C}\text{: } (M,s)(A) =_{def} A^{I(s)} \\ &&& \text{for } R \in \mathbf{R}\text{: } (M,s)(R) =_{def} R^{I(s)} \\ &\text{(SC2)} && (M,s)(C \sqcap D) = (C \sqcap D)^{I(s)} =_{def} C^{I(s)} \cap D^{I(s)} \\ &&& (M,s)(C \sqcup D) = (C \sqcup D)^{I(s)} =_{def} C^{I(s)} \cup D^{I(s)} \\ &&& (M,s)(\neg C) = (\neg C)^{I(s)} =_{def} \Delta^I \backslash C^{I(s)} \\ &\text{(SC3)} && (M,s)(\forall R.C) = (\forall R.C)^{I(s)} =_{def} \{a \in \Delta^I | \forall b \in \Delta^I : (a,b) \in R^{I(s)} \to b \in C^{I(s)}\} \\ &&& (M,s)(\exists R.C) = (\exists R.C)^{I(s)} =_{def} \{a \in \Delta^I | \exists b \in \Delta^I : (a,b) \in R^{I(s)} \wedge b \in C^{I(s)})\} \\ &\text{(PC1)} && (M,x)(\mathsf{X}C) =_{def} C^{I(s_1)} \\ &&& (M,x)(C \,\mathsf{U}\, D) =_{def} \{a \in \Delta^I | \exists i \in \mathbb{N}_0 \, [a \in D^{I(s_i)} \wedge \forall j \in \mathbb{N}_0 (j < i \to a \in C^{I(s_j)})]\} \end{aligned}$$

(SC4) $(M,s)(\mathsf{EC}) = (\mathsf{EC})^{I(s)} =_{def} \{a \in \Delta^I \mid \exists x \in FP_s\colon a \in (M,x)(C)\} = \bigcup_{x\in FP_s}(M,x)(C)$,
$(M,s)(\mathsf{AC}) = (\mathsf{AC})^{I(s)} =_{def} \{a \in \Delta^I \mid \forall x \in FP_s\colon a \in (M,x)(C)\} = \bigcap_{x\in FP_s}(M,x)(C)$

The following rules determine when an $\mathcal{ALC}$CTL formula p is true in a state $s \in S$, or in fullpath $x = (s_0, s_1, \ldots)$, in symbols $M, s \models p$ and $M, x \models p$, respectively:

(SF0) $M, s \models \mathit{true}$
$M, s \not\models \mathit{false}$

(SF1) $M, s \models C \sqsubseteq D$ iff $C^{I(s)} \subseteq D^{I(s)}$
$M, s \models C \doteq D$ iff $C^{I(s)} = D^{I(s)}$

(SF2) $M, s \models p \wedge q$ iff $M, s \models p$ and $M, s \models q$
$M, s \models p \vee q$ iff $M, s \models p$ or $M, s \models q$
$M, s \models \neg p$ iff $M, s \not\models p$

(PF1) $M, x \models \mathsf{X}p$ iff $M, s_1 \models p$
$M, x \models p \,\mathsf{U}\, q$ iff $\exists i \in \mathbb{N}_0\ [M, s_i \models q \wedge \forall j \in \mathbb{N}_0 (j < i \rightarrow M, s_j \models p)]$

(SF3) $M, s \models \mathsf{E}p$ iff $\exists x \in FP_s\colon M, x \models p$,
$M, s \models \mathsf{A}p$ iff $\forall x \in FP_s\colon M, x \models p$

Example 12 (*Semantics of $\mathcal{ALC}$CTL*)**.** Let $M = (S, R, I)$ be a temporal structure as of Example 10 and as depicted in Fig. 8. Then

$$(M, \mathit{start.htm})(\mathsf{EX}\ \mathit{WarningTopic}) = (\mathrm{SC4}) \text{ in Definition 4}$$
$$\bigcup_{x\in FP_{\mathit{start.htm}}} (M,x)(\mathsf{X}\ \mathit{WarningTopic}) = \text{explanation below...}$$
$$(M, (\mathit{start.htm}, \mathit{safety\_notes.htm}, \ldots))(\mathsf{X}\ \mathit{WarningTopic}) = (\mathrm{PC1}) \text{ in Definition 4}$$
$$(M, \mathit{safety\_notes.htm})(\mathit{WarningTopic}) = (\mathrm{SC1}) \text{ in Definition 4}$$
$$\mathit{WarningTopic}^{I(\mathit{safety\_notes.htm})} = \cdot^I \text{ in Example 10}$$
$$\{\mathit{WarmAfterExtendedUse}, \mathit{NotDirectlyAtSun}, \ldots\}$$

The second equation holds because (1) in the temporal structure of Fig. 8, all paths starting from state *start.htm* have the form (*start.htm*, *safety_notes.htm*, ...) and (2) for the semantics of the "next" operator X, just the state subsequent to the initial state *start.htm* is relevant (cf. (PC1) in Definition 4).

Note that the user does not interact with the system on the formal level described here. Instead, the user browses an example base of specifications that is structured according to frequently occurring types of specifications, the so-called specification patterns (Jakšić and Freitag, 2008; Weitl et al., 2009). The user selects a suitable example-based specification and then incrementally refines and adapts it to the specification problem at hand. In this process, the meaning of the current specification is illustrated by showing conforming and violating paths in small sample document graphs which can be adjusted to the current use case (Weiß, 2009).

6.2. Model checking $\mathcal{ALC}$CTL

Specifications represented in $\mathcal{ALC}$CTL are verified by model checking (Weitl, 2008; Weitl et al., 2009). In the case of specification violations, counterexamples are generated that precisely pinpoint the error locations within the document.

Definition 5 (*Model Checking Problem of $\mathcal{ALC}$CTL*)**.** Let M be a structure (S, R, I) as defined in Definition 2 such that S and Δ^I are finite. Let f be an $\mathcal{ALC}$CTL formula. The model checking problem of $\mathcal{ALC}$CTL is to decide for all $s \in S$ if $M, s \models f$.

Example 13 (*Model Checking Problem of $\mathcal{ALC}$CTL*)**.** Consider the $\mathcal{ALC}$CTL formula

$$\mathsf{EG}\ (\mathit{WarningTopic} \sqsubseteq \bot)$$

expressing that there should be some path on which it holds in every state (EG) that the set of "warning topics" is empty ($\mathit{WarningTopic} \sqsubseteq \bot$), in short, that there is some path without any warnings.

Table 1
Error report of verifying formula *AbbreviatedTerm* $\sqsubseteq$ EF *GlossaryTerm*.

Error location	Violating terms
photography_auto.htm	"Gigabyte", "Auto Focus"
connecting_pc.htm	"Universal Serial Bus", "Motion Picture Export Group"

Let $M = (S, R, I)$ be the temporal structure as of Example 10 and as depicted in Fig. 8. Then it holds:

$$M, start.htm \not\models \mathsf{EG}\,(WarningTopic \sqsubseteq \bot)$$
$$M, safety\_notes.htm \not\models \mathsf{EG}\,(WarningTopic \sqsubseteq \bot)$$

This is because the only successor of state *start.htm* is *safety_notes.htm* in which the interpretation of *WarningTopic* is not empty (cf. Fig. 8). Thus, in states *start.html* and *safety_notes.htm*, there is no such path such that *WarningTopic* $\sqsubseteq \bot$ holds in every state. However,

$$M, toc.htm \models \mathsf{EG}\,(WarningTopic \sqsubseteq \bot)$$

because there is a path (*toc.htm*, *glossary.htm*, *glossary.htm*, *glossary.htm*, ...) starting at state *toc.htm* on which it holds for every state that the interpretation of *WarningTopic* is empty. This path exists since we added a reflexive edge to the final state *glossary.htm* (cf. Remark 10).

Theorem 6 (*Polynomial Runtime Complexity*)**.** *The model checking problem of* $\mathcal{ALC}$CTL *is in polynomial time.*

For the proof we refer the reader to Weitl (2008). The polynomial runtime complexity is an important prerequisite for $\mathcal{ALC}$CTL model checking scaling up to relevant problem sizes (see also evaluation results in Section 8 and in Weitl (2008), Weitl et al. (2009) and Schönberg et al. (2009)). A polynomial model checking algorithm has been proposed in Weitl (2008) and implemented in Java. It is integrated into the demonstration software available on the Verdikt homepage.[3]

7. Error reporting

Definition 7 (*Error State*)**.** Let M be a temporal structure (S, R, I) as defined in Definition 2 such that S and Δ^I are finite. Let f be an $\mathcal{ALC}$CTL formula. Then

$$ES_{M,f} := \{s \in S \mid M, s \not\models f\}$$

denotes the set of *error states* in M w.r.t. f.

Example 14 (*Error State*)**.** Consider the $\mathcal{ALC}$CTL formula $f = \mathsf{EG}\,(WarningTopic \sqsubseteq \bot)$ and the temporal structure $M = (S, R, I)$ as of Example 13. Then $start.htm \in ES_{M,f}$ and $safety\_notes.htm \in ES_{M,f}$ but $toc.htm \notin ES_{M,f}$ because f is satisfied in state *toc.htm* but not in states *start.htm* and *safety_notes.htm* of M (cf. Example 13).

The set of error states $ES_{M,f}$ of a verification model M and an $\mathcal{ALC}$CTL formula f represent *error locations* which correspond to fragments of the document that do not conform to the criterion represented by formula f. By applying appropriate naming conventions, these states can be re-mapped onto the respective parts of the document. For instance, state *photography_auto.htm* represents the part of the document that is contained in *photography_auto.htm* (cf. Table 1, first data row).

Based on the model checking results, an error report as sketched in Tables 1 and 2 is generated. The first data row of Table 1 expresses that the terms "Gigabyte" and "Auto Focus" used in abbreviated form on page *photography_auto.htm* are not satisfying the formula *AbbreviatedTerm* $\sqsubseteq$ EF *GlossaryTerm*.

In addition to the error report, a CSS file is generated that highlights the violating terms within an error location of the document. For locating violating terms, a mapping of terms onto their locations

[3] http://www.verdikt.uni-passau.de.

Table 2
Error report of verifying formula $\exists \mathit{topicOf}.\mathit{Overview} \sqsubseteq \mathsf{AF}\, \exists \mathit{topicOf}.\neg(\mathit{Overview} \sqcup \mathit{Glossary})$.

Error location	Violating terms
camera_overview.htm	"Arrow Buttons", "Battery Cover", "Lens", …

in the document is built during the generation of the verification model and represented as a table. This is possible since the metadata document model contains the information needed to locate terms precisely in the document (cf. Section 4). The table consists of quadruples of *state*, *concept*/*role*, *instance value*, and *file location*. This set of tuples is indexed first by *state*, then by *concept*/*role*.

Example 15 (*Mapping Table*)**.** The tuple table of the use case contains entries like

(*photography_auto.htm*, *AbbreviatedTerm*, "Gigabyte",
`photography_auto.htm#html/body/div[1]/div[2]/ol/li[3]/span[2]`)
and
(*connecting_pc.htm*, *AbbreviatedTerm*, "Universal Serial Bus",
`connecting_pc.htm#html/body/div[2]/span[3]`).

When tracing the errors from the report in Table 1 back to their source, a table lookup reveals the locations of the violating terms. The state and term (instance value) are given in the error report, and the relevant concept is obtained from the formula $\mathit{AbbreviatedTerm} \sqsubseteq \mathsf{EF}\ \mathit{GlossaryTerm}$ because violating terms are instances of concept *AbbreviatedTerm* but not instances of concept $\mathsf{EF}\ \mathit{GlossaryTerm}$.

For further details on error analysis based on $\mathcal{ALC}$CTL model checking, we refer the reader to Weitl et al. (2010), Weitl and Nakajima (2010).

8. Experimental results

In previous work, we established the applicability of our approach to large and complex documents from the e-learning domain, which are in productive use in a university and a corporate setting (Weitl, 2008; Schönberg, 2006). In this section, we focus on the applicability of the approach to the domain of technical documentation by examining the presented use case. The use case consists of a document modeled after the existing documentations.

We checked the use case document against a total of eight criteria (**C1**–**C8**), each of which was specified as a single formula. Two of these formulae were satisfied by the document model, but we found violations for the other six. In particular, the formula representing **C1** (cf. Section 2.2) was satisfied, owing to the background knowledge inserted into the document model. In total, we found 31 violations in all pages of the document for all formulae. Fig. 9 shows the time required to check each formula (in milliseconds). As can be seen, the largest portion of time is required for loading the verification model from an XML file. The total of this time can be greatly reduced by checking all formulae at once, instead of checking each separately. The total time of 3.5 s is then reduced to just under 1 s.

The runtime results shown in Fig. 9 have been obtained on a notebook computer with Intel Pentium M processor at 1.6 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 16 s. The major portion of runtime was consumed by the information extraction and model generation process that analyze HTML markup, establish semantic relations between content fragments, enrich these correlations by using background knowledge, generate a graph model of the document, and finally generate a verification model for model checking using a set of parameters. Extracting the graph model from the document took about 5.8 s, while selecting and filtering this graph took under 10 s.

Remark 11 (*Caching*)**.** The document model can be generated beforehand in a preprocessing step, cached, and reused as long as the document itself remains unmodified. Only changing the original document must trigger a new information extraction and model generation cycle. The JBoss Drools extraction rules can also be pre-compiled, reducing the information extraction time to slightly under 1 s.

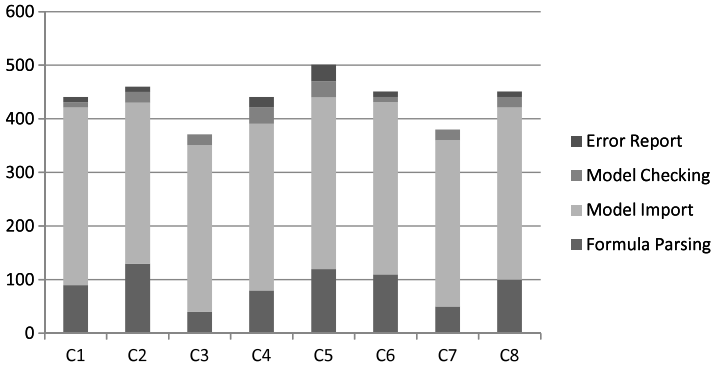

Fig. 9. Evaluation Results (times in ms).

Model 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 against a verification model that has already been loaded.

Background knowledge on syntactical indicators and on the content domain helped to increase the number of recognized terms in the document (*recall*) from 77 to 104 (an increase of 35%). Since no terms or concepts were falsely extracted without using background knowledge, no errors had to be corrected, so the *precision* remain unchanged.

The *application costs* for our verification system arise from

Preparing the document Sometimes, the existing documents do not contain all the markup required to correctly identify the meaning of their content. The extent of available semantic markup often varies depending on the application domain: e-learning documents usually contain a high amount of such metadata, while pure web documents contain little or none. For our use case, we first created the HTML sources and added the semantically relevant markup afterwards, which took about 1 h.

Preparing the background knowledge Available background knowledge can help balance missing markup in the source document. Especially ontologies of relevant terms, and knowledge about keywords that indicate some document function like "Definition" are important in this regard. While this kind of background knowledge is often available for different content domains (e.g., Rogers et al., 2001, ChefMoz,[4] the UK Integrated Public Sector Vocabulary,[5] or the Getty Art and Architecture Thesaurus,[6]) it takes considerable time and effort to compile them if they are not available. There are several approaches to compile ontologies from public information sources like Wikipedia (Hepp et al., 2007; Wu and Weld, 2008; Nastase and Strube, 2008) that might help to reduce the cost of obtaining new ontologies. Other background knowledge like information about CSS classes can be created at small expense. Creating the knowledge bases for our limited use case required about half an hour.

Preparing the specification Eight criteria for the content of the document have been expressed in natural language and then formalized in terms of $\mathcal{ALC}$CTL. Manually specifying the formulae took about 2 h. Using the pattern-based specification approach to assist reduces this time to approximately 15 min.

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 2 h in total.

4 http://chefmoz.org/, visited 06/2010.

5 http://www.esd.org.uk/standards/ipsv_internalvocabulary/, visited 06/2010.

6 http://www.getty.edu/research/conducting_research/vocabularies/aat/, visited 06/2010.

Altogether, the setup cost of our verification system amounts to about 4 h of manual effort. This initial effort amortizes quickly when a document is changed frequently or when parts of it are reused in different contexts, which is typically the case for technical documentations.

Employing background knowledge during the information extraction process has several implications for the verification process.

- Some criteria can now be checked that were not available before, because the necessary data could not be detected (e.g., abbreviations, or related topics).
- Some verification results are now more accurate than before, because of higher precision and recall for some data (e.g., terms, or function types).
- By making knowledge about how information is encoded in a document explicit and external, instead of mixing it with the extraction logic, the extraction rules can be specified more clearly. This results in reduced effort for creation and maintenance.
- The runtime results for the extraction process are slightly increased in direct proportion to the amount of background knowledge available. Efficient representations can help to reduce this effect.

Application scenarios and documents investigated so far have shown that the advantages to using background knowledge clearly outweigh the drawbacks in most cases, including the cost for obtaining the knowledge. Only for very simple documents and simple consistency criteria employing background knowledge can be regarded as superfluous.

9. Related work

Schematron (Jelliffe, 2002) and xlinkit (Nentwich et al., 2002) are powerful tools for validating the consistency of XML documents. Schematron is an ISO/IEC standard, based on XSLT and XPath for the specification and validation of consistency requirements on XML documents. As for xlinkit, consistency rules are expressed in a language based on first order logic with embedded XPath predicates. Consistency rules may refer to external resources that can be mapped to XML representations, such as relational databases. If rules are violated, suggestions for repairing the detected errors are generated and presented to the user (Nentwich et al., 2003). The xlinkit engine has been integrated into the commercial software "messageAUTOMATION validator" that offers a graphical rule editor and means for rule testing and management (Message Automation Ltd , 2010).

Our approach is not targeted at the XML data model of ordered trees but at documents with a graph (but not necessarily tree) structure such as hypertext or DITA (Priestley and Hackos, 2005) documentations of interrelated topics. Properties of paths in such graph-structured documents are hard to express and inefficient to check using XPath and first order logic, which are fundamental to Schematron and xlinkit. This is because the number of paths to consider grows exponentially in the size of the document or is infinite. In our approach, we do not track paths but calculate the set of states satisfying an $\mathcal{ALC}$CTL formula by fixed point iteration over a graph-based representation of document models (Weitl, 2008); cf. also Huth and Ryan (2004). This way, the runtime complexity of the algorithm remains polynomial. XML processing, however, is not well suited for efficiently implementing such an approach (Weitl, 2008). Moreover, terminological background knowledge about the domain of discourse is hard to encode in XML-based validation methods. A detailed study of the expressiveness and performance of our approach as compared to methods based on XML processing is presented in Weitl (2008).

There are several approaches, e.g. Stotts et al. (1998), Sciascio et al. (2005), Flores et al. (2008), using some propositional temporal logics (CTL, LTL), which enable the specification of complex properties along browsing paths in hypermedia structures. In the MCWeb project (de Alfaro, 2001), a restricted version of the μ-calculus is applied to express and verify properties of web sites while (Fernandez et al., 1999) suggests Prolog extended with path expressions for modeling and verifying the structure and content of web sites. Although $\mathcal{ALC}$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 $\mathcal{ALC}$CTL results in richer and more precise error reports that clearly pinpoint problems within the document.

In the VERDI project (Alpuente et al., 2004), a rule-based, formal specification language, has been used to define syntactic/semantic properties of web sites. A verification facility based on term rewriting computes the specifications not fulfilled by the web site and helps in repairing errors by finding incomplete or missing web pages. While complex requirements for certain (sets of) web pages can be checked, path-related properties are out of the scope of the approach.

A formal consistency management component based on description logics is proposed in Egly et al. (2005) as an extension to the content management system for technical documentation Schema ST4 (Gruppe, 2006). 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. Weitl, 2008).

A powerful and flexible framework for checking the consistency of collections of interrelated documents has been proposed by Scheffczyk (2004). 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.

There are many information extraction and information retrieval tools available. Lapis relies on text constraints to define patterns that are used to select data from web pages (Miller and Myers, 1999). The commercial toolkit Lixto uses a rule language based on constraints and path expressions similar to XPath (Baumgartner et al., 2001, 2005). Both rely on graphical user interfaces to assist users in specifying rules and constraints. Road Runner employs regular expressions to generate wrappers for web pages (Crescenzi et al., 2002). It attempts to generate these wrappers semi-automatically by examining a small number of pages and building the wrappers to conform to these samples, before applying them to a larger set of similar pages. Our approach has several advantages for document verification. We can use background knowledge in more ways and more flexibly, thus making better use of its effects on extraction quality. It is also possible to employ a host of external tools and easily include new ones, without being limited by the capabilities of the graphical interface currently in use. In addition, our approach not only extracts the data but also records its source in the document, a feature that is necessary for error tracing.

10. Conclusion

We have presented an approach to document verification that spans the entire verification cycle: from the original document, a metadata model is extracted that represents the structure and the content of the document, with special account for semantic relationships among document parts and topics. With respect to that model, a set of consistency criteria is constructed in an incremental process based on specification patterns and examples. The example- and pattern-based specification is translated into formulae of the temporal description logic $\mathcal{ALC}$CTL. In addition, a verification model is generated that is a view on the document model optimized for the verification of the given set of $\mathcal{ALC}$CTL formulae. The $\mathcal{ALC}$CTL model checker produces a list of counterexamples which are processed to generate an error report pinpointing all violations of the given criteria in the original document. To this end, unique identifiers and mapping tables help to track objects in counterexamples back through the document model to their precise locations in the original document. After users correct the errors presented to them, the verification cycle begins anew.

We have shown that background knowledge about the document format, about the application domain of the document, about the content domain in which the document is used, and about the general linguistic correlations can help us to enhance the quality of the results of information extraction and model generation as well as the precision and ease of application for users in the specification process. We have demonstrated the expressiveness of $\mathcal{ALC}$CTL for content- and path-related criteria and have evaluated its adequacy and performance in typical application scenarios. In combination with its low utilization costs, the presented framework constitutes a new step towards closing the gap between the power of formal methods and their practical applicability.

In future work, we will extend the amount of automation that can be applied to the information extraction and to locating the errors. Furthermore, there is a strong evidence that the runtime

efficiency of the model generation can be increased by caching and query optimization methods known from relational databases. Another part of further research is devoted to an optimized assistance for the specification of consistency criteria by supporting the extension of the pattern and example base with user-defined scenarios, thus increasing both the flexibility of the specification method and the user's familiarity with the specification process.

Acknowledgements

We thank Shin Nakajima for many valuable comments.

References

Alpuente, M., Ballis, D., Falaschi, M., 2004. Verdi: An automated tool for web sites verification. In: Proc. of JELIA 2004. In: LNAI, vol. 3299. Springer, pp. 726–729.
Baader, F., Nutt, W., 2003. Basic description logics. In: The Description Logic Handbook — Theory, Implementation and Applications. Cambridge Univ. Press, pp. 47–100 (Chapter 2).
Baumgartner, R., Flesca, S., Gottlob, G., 2001. Visual web information extraction with lixto. In: Proceedings of the VLDB Conference. pp. 119–128.
Baumgartner, R., Frölich, O., Gottlob, G., Harz, P., Herzog, M., Lehmann, P., 2005. Web data extraction for business intelligence: the lixto approach. In: Vossen, G., Leymann, F., Lockemann, P., Stucky, W. (Eds.), Datenbanksyst. in Business, Technologie und Web. LNI P-65. Bonner Köllen Verlag, pp. 30–47.
Bost, T., Bonnard, P., Proctor, M., 2007. Implementation of production rules for a rif dialect: A mismo proof-of-concept for loan rates. In: Paschke, A., Biletskiy, Y. (Eds.), RuleML. In: Lecture Notes in Computer Science, vol. 4824. Springer, pp. 160–165.
Crescenzi, V., Mecca, G., Merialdo, P., 2002. Wrapping-oriented classification of web pages. In: SAC '02: Proceedings of the 2002 ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1108–1112.
de Alfaro, L., 2001. Model checking the world wide web. In: CAV 01. In: LNCS, vol. 2102. Springer, pp. 337–349.
Doorenbos, R.B., 1995. Production matching for large learning systems. Ph.D. Thesis, Pittsburgh, PA, USA.
Egly, U., Schiemann, B., Schneeberger, J., 2005. Tech. documentation authoring based on semantic web methods. Künstliche Intelligenz 2, 56–59.
Emerson, E., 1990. Temporal and modal logic. In: van Leeuwen, J. (Ed.), Handbook of Theoretical Computer Science: Formal Models and Semantics. Elsevier, pp. 996–1072.
Fernandez, M., Florescu, D., Levy, A. Y., Suciu, D., 1999. Verifying integrity constraints on web sites. In: Proc. of the 16th Internat. Joint Conf. on Artificial Intelligence. IJCAI. Morgan Kaufmann Pub. Inc., pp. 614–619.
Flores, S., Lucas, S., Villanueva, A., 2008. Formal verification of websites. Electronic Notes in Theoretical Computer Science 200, 103–118.
Gruppe, S., 2006. Schema ST4 Leistungsbeschreibung. SCHEMA Electronic Documentation Solutions GmbH.
Hepp, M., Siorpaes, K., Bachlechner, D., 2007. Harvesting wiki consensus: Using wikipedia entries as vocabulary for knowledge management. IEEE Internet Computing 11 (5), 54–65.
Huth, M., Ryan, M., 2004. Verification by model checking. In: Logic in Computer Science, Modelling and Reasoning about Systems, 2nd edition. Cambridge University Press (Chapter 3).
Jakšić, M., Freitag, B., 2008. Temporal patterns for document verification. Tech. Rep. MIP-0805, University of Passau.
Jakšić, M., Freitag, B., 2009. Evaluation der Methode zur Nutzerunterstützung. Tech. Rep. MIP-0906, University of Passau, Germany.
Jakšić, M., Freitag, B., 2010. Temporal patterns for document verification. In: Proc. of the 6th International Workshop on Automated Specification and Verification of Web Systems, WWV'10, Vienna, Austria, pp. 17–31.
Jelliffe, R., 2002. The schematron assertion language 1.6. http://xml.ascc.net/resource/schematron/Schematron2000.html, visited 06/2010.
Klyne, G., Carroll, J.J., 2004. Resource Description Framework (RDF) Concepts and Abstract Syntax, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-concepts/, visited 06/2010.
Message Automation Ltd. 2010. messageAUTOMATION validator product site. http://www.messageautomation.com/products/validator.html, visited 06/2010.
Miller, G.A., 2006. WordNet — a lexical database for the English language. http://wordnet.princeton.edu/.
Miller, R., Myers, B., 1999. Lightweight structured text processing. In: USENIX Annual Technical Conference. Monterey, CA, pp. 131–144.
Nastase, V., Strube, M., 2008. Decoding wikipedia categories for knowledge acquisition. In: Fox, D., Gomes, C.P. (Eds.), AAAI. AAAI Press, pp. 1219–1224.
Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A., 2002. xlinkit: a consistency checking and smart link generation service. ACM Transactions on Internet Technology (TOIT) 2 (2), 151–185.
Nentwich, C., Emmerich, W., Finkelstein, A., 2003. Consistency management with repair actions. In: Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, pp. 455–464.
Priestley, M., Hackos, J., 2005. OASIS Darwin Information Typing Architecture (DITA) Language Specification v1.0, OASIS Standard.
Proctor, M., 2007. Relational declarative programming with jboss drools. In: Negru, V., Jebelean, T., Petcu, D., Zaharie, D. (Eds.), SYNASC. IEEE Computer Society, p. 5.
Prud'hommeaux, E., Seaborne, A., 2008. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, visited 06/2010.

Rogers, J.E., Roberts, A., Solomon, W.D., van der Haring, E., Wroe, C.J., Zanstra, P.E., Rector, A.L., 2001. Galen ten years on: Tasks and supporting tools. In: Patel, V., et al. (Eds.), Proceedings of MEDINFO 2001. IOS Press, Amsterdam, pp. 256–260.
Scheffczyk, J., 2004. Consistent document engineering. Dissertation, Universität der Bundeswehr München.
Schönberg, C., 2006. Model checking temporal description logics. Diploma Thesis, University of Passau.
Schönberg, C., Freitag, B., 2009. Extracting and storing document metadata. Tech. Rep. MIP-0907, University of Passau.
Schönberg, C., Jakšić, M., Weitl, F., Freitag, B., 2009. Verification of web-content: A case study on technical documentation. In: Proc. of the 5th International Workshop on Automated Specification and Verification of Web Systems, WWV'09, Linz, Austria.
Schönberg, C., Weitl, F., Jakšić, M., Freitag, B., 2009. Logic-based verification of technical documentation. In: Proceedings of the 9th ACM Symposium on Document Engineering. DocEng 09. ACM, Munich, Germany, pp. 251–252.
Schönberg, C., Pree, H., Freitag, B., 2010. Rich ontology extraction and wikipedia expansion using language resources. In: Proc. of the 11th International Conference on Web-Age Information Management, WAIM'10. In: LNCS, vol. 6184, Jiuzhaigou, China, pp. 151–156.
Sciascio, E. D., Donini, F. M., Mongiello, M., Totaro, R., Castelluccia, D., 2005. Design verification of web applications using symbolic model checking. In: Proc. of ICWE 2005. In: LNCS, vol. 3579. Springer, pp. 69–74.
Stotts, P.D., Furuta, R., Cabarrus, C.R., 1998. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. Information Systems 16 (1), 1–30.
Walsh, N., Muellner, L., Stayton, B., Apr 2005. DocBook: The Definitive Guide, Version 2.0.12. O'Reilly, online Version.
Weiß, H., 2009. Specification-by-Example: Beispiels-basierte Spezifikationen von Dokumenteigenschaften. Diplomarbeit, Universität Passau.
Weitl, F., 2008. Document Verification with Temporal Description Logics. Ph.D. thesis, University of Passau.
Weitl, F., Nakajima, S., 2010. Incremental construction of counterexamples in model checking web documents. In: Proceedings of the 6th International Workshop on Automated Specification and Verification of Web Systems, WWV'10, Vienna, Austria, pp. 61–75.
Weitl, F., Jakšić, M., Freitag, B., 2009. Towards the automated verification of semi-structured documents. Journal of Data & Knowledge Engineering 68, 292–317.
Weitl, F., Nakajima, S., Freitag, B., 2010. Structured counterexamples for the temporal description logic $\mathcal{ALC}$CTL. In: Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy.
Wu, F., Weld, D. S., 2008. Automatically refining the wikipedia infobox ontology. In: WWW '08: Proceeding of the 17th international conference on World Wide Web. ACM, New York, NY, USA, pp. 635–644.