Semantic Web 0 (2022) 1
IOS Press

FitLayout: An RDF-Based Framework and Toolkit for Web Page Content Analysis

Radek Burget [a,*] and Hamza Salem
[a] *Faculty of Information Technology, Brno University of Technology, Czech Republic*
E-mail: burgetr@fit.vut.cz

Abstract. Despite the ongoing development of technologies that allow the publication of structured information within web pages, there still exist many web sources that publish useful data solely in the form of plain HTML documents. The data published in this way is difficult to extract and integrate with other data sets. One of the promising options is to analyze the visual presentation of data within the web page, but this is quite a complex task from an implementation point of view. In this paper, we present FitLayout, a framework and open-source toolkit that implements a web page processing workflow consisting of an arbitrary number of steps during which all page information is represented using a unified RDF model. This model contains detailed information about the visual appearance of the page and each of its content elements, as well as the results of analytical steps such as page segmentation. This allows easy archiving of all the details of rendered pages, generation of annotated data sets from web pages for different purposes, and their integration with other linked data. FitLayout also provides a platform for easy implementation of vision-based page analysis algorithms and includes ready-to-use implementations of algorithms for page segmentation, identification of important content elements, and others.

Keywords: Web page rendering, Web page analysis, Page segmentation, Web information extraction, Software framework, Document modelling, RDF

1. Introduction

Web pages present a potentially extremely rich source of information in a variety of areas, including news, product information, places, events, and a wide range of others. Recently, a great effort has been dedicated to the development of shared vocabularies such as Schema.org and the related metadata annotation formats including Microdata, RDFa or embedded JSON-LD, which allow information presented in HTML documents to be available in a machine-understandable form and integrated with other structured data sets. Although the share of structured information in documents continues to grow (the Web Data Commons statistics for October 2021 [1] report 47.4 % out of the 3.2 billion pages in the Common Crawl data set containing some kind of structured metadata), there still remains a vast number of web pages that contain no or only partial structured data annotations.

Automatic extraction of structured information from documents with no semantic annotations is a complex process that involves multiple problems that are often researched separately, such as page segmentation (the discovery of important content blocks) [2], logical structure detection (identification of logical relationships among the content parts) [3], recognition of important content elements [4] or structured record extraction [5]. Existing approaches to tackle these problems can be based on the analysis of different aspects of the source documents, which include the source code (DOM-based methods [5]) or even the visual presentation of the content (vision-based methods [2, 6])

*Corresponding author. E-mail: burgetr@fit.vut.cz.

and the analysis itself can be based on handcrafted rules [6] as well as the application of machine learning methods [2].

Regardless of the chosen approach, as part of the research, the methods need to be implemented and experimentally verified. Web pages, as source data, are complex entities consisting of HTML documents, CSS styles, images and other additional parts that need to be properly interpreted by a web browser. From this perspective, the researchers often face two basic problems that are closely related to each other:

- *Implementation of web page analysis methods.* When designing and implementing especially the vision-based methods, the researchers typically rely on employing some of the existing web browsers [2, 6, 7], whose application interface is not entirely designed for obtaining all the necessary information about the rendered page.
- *Web page data set creation.* For a reproducible evaluation of the developed methods, it is necessary to create stable sets of source web pages, that do not consist of the source URLs only (since the content behind a URL may change or disappear at any time) but they include all the analyzed aspects of the pages. At the same time, the data set must include the expected (for training or evaluation) and/or obtained results (for comparison) of the analysis that include mainly the detected visual areas (for page segmentation) or the boundaries and meaning of the detected important content elements. In addition, both the detected areas and content elements may be linked to other existing data sets, such as the JSON-LD structured data annotations.

As we discuss below in section 2, none of these problems is sufficiently covered by the existing tools, libraries and data formats and the researchers are forced to use their own proprietary solutions in both areas. This makes both the implementation of the web page analysis methods and reproducing their results very demanding. Therefore, we created our FitLayout framework, which uses semantic web technology, specifically RDF along with a set of vocabularies, and the related implemented software infrastructure to provide the following contributions:

- An explicit, extensible, and platform-independent description of all aspects of the web pages during all stages of their processing from their initial acquisition and rendering up to the results of the analytical steps (page segmentation, named entity recognition, etc.)
- Means for archiving an annotating the web pages that allow creating evaluation or training data sets.
- Software platform for implementing and evaluating the web page analysis methods.
- Tools for executing the analytical tasks and browsing the results via both the command-line interface (CLI) and a web-based graphical user interface (GUI).

The framework is based on a semantic RDF-based model that describes the web pages as they have been rendered by a web browser from their overall appearance (a screen shot) up to the detailed description of the visual appearance of the smallest content elements (an exact position and visual properties of every text or other elements). To our best knowledge, this feature is unique for the FitLayout framework, and it allows to store the rendered web pages, and use the obtained information for implementing and evaluating different page analysis methods (such as page segmentation methods) without dealing with browser-specific implementation problems. In addition, the results of page analysis (as for example the discovered content blocks) may be added to the existing RDF graph providing a complete picture of the page after performing different analytical steps. Finally, the chosen RDF description allows to easily link the content elements of a rendered page with other data sets including the associated Microdata annotations or embedded JSON-LD as well as any other linked data sets (as for example DBPedia).

2. Related work

The related research areas correspond to the two basic related problems addressed by FitLayout as mentioned in the introduction: the implementation of vision-based web page analysis algorithms and an explicit representation of both the input documents and the analysis (such as segmentation) results that can be stored, shared, or linked.

To our best knowledge, there is no platform or library for implementing the algorithms that require a rendered page on their input that would provide a complete and unified representation of the page. Authors typically use embedding an existing rendering engine [6], use a standard web browser controlled remotely via a specific tool such

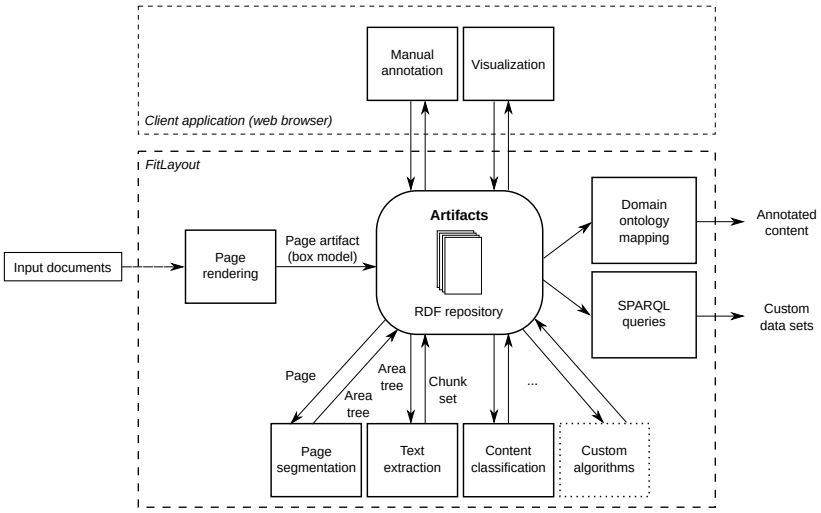

Fig. 1. Web page processing workflow: Page rendering as the initial step (left), different kinds of analytical steps (bottom) and interactive steps (top) and the analysis outcomes (right).

as Selenium web driver[1] [2] or implement their algorithms in JavaScript to run directly in the browser [7]. However, the browsers in this respect offer only a limited application interface, which is oriented on DOM[2] elements and obtaining the details about the actually rendered boxes is a demanding task that requires a deep understanding of the browser architecture. In addition, because no explicit description of the rendered page is used, it is difficult to reproduce and compare the results of the methods implemented in this way [8].

We are also not aware about any documented data format that would storing rendered web pages and annotating them manually or with the results of analysis methods. Most of the existing data sets such as SWDE [9] or WEIR [10] consist of saved HTML code only, which makes them unusable for evaluating the vision-based methods. For archiving complete web pages, there exists a standardized WARC[3] format used by [11] for creating web page data sets. However, it only stores page source data files. For reconstructing the rendered page, the source must be re-rendered by a web browser. Due to the continuous evolution of browsers and a number of external conditions, the result of a repeated rendering may not entirely correspond to the original page [11].

Many researchers have reported creating their own tools for interactive annotation of the expected results of the analysis (the boundaries the detected visual areas or content elements and their mining) with the purpose of manually creating a "gold standard" data set for the method evaluation [12, 13] or training the machine learning algorithms [2]. However, since no standard format for the exchange of such information is available, most of the published data sets provide this information in a proprietary way, typically as separate text files [9, 10, 13].

3. Basic concepts

FitLayout assumes processing an input web page in multiple steps whose selection and order depends on the target application and required outcomes. The expected workflow of processing a page is illustrated in Figure 1. The steps may include page rendering, segmentation and further analysis as discussed in detail below.

The results of the individual steps are stored in a central RDF repository. The repository holds a single RDF graph, which is further divided to named sub-graphs that we call *artifacts*. Each artifact represents a particular web page at some level of abstraction. A processing step may (optionally) consume an existing artifact, i.e. read and analyze the corresponding RDF sub-graph, and produce another artifact or add new information (new RDF statements) to

[1] https://www.selenium.dev/documentation/webdriver/
[2] Document Object Model
[3] http://bibnum.bnf.fr/WARC/

Table 1
Supported types of artifacts and the elements they consist of.

Artifact type		Content element	
Physical (rendering) level			
Page	A rendered page represented as a tree of *boxes*.	**Box**	A rectangular box generated by the rendering engine.
Logical (interpretation) level			
AreaTree	A tree of *visual areas* detected for example by page segmentation.	**Area**	A rectangular visual area detected in the page.
LogicalArea Tree	A tree of *logical areas* obtained by domain-specific interpretation of the visual areas.	**LogicalArea**	A logical area created by concatenating one or more visual areas that have some logical interpretation in the target domain.
ChunkSet	A set of extracted *text chunks*.	**TextChunk**	A continuous part of the document text extracted using different methods.

existing artifacts. When a new artifact *B* is produced based on the analysis of an existing artifact *A*, we call *A* a *parent artifact* and *B* a *derived artifact*.

We define several artifact types that are listed in Table 1. Depending on its type, each artifact consists of *content elements* that represent the contents of the artifact. All the artifact types and their content element types are formally defined as classes in the FitLayout ontologies that are discussed further in section 4.

Typically, the initial step consists of rendering the input page, i.e. obtaining a model of the rendered page from the input documents. This step produces a *Page* artifact that represents the page exactly as it has been rendered by the web browser rendering engine. Thus, it represents a *physical (rendering) level of abstraction*. It consists of a tree of *boxes*, that represent rectangular areas generated by the source DOM elements as defined by the CSS Visual Formatting Model [14]. For each box, detailed information about its contents and its visual appearance is stored such as its exact position in the page, used font, color, contained text and other properties defined by the FitLayout box model ontology. Similarly, the detailed visual properties are stored for the entire page including a bitmap snapshot of the rendered page for further reference.

Subsequent processing steps may include layout analysis (such as the application of page segmentation algorithms on the given page), identification of specific text elements, content classification or other kinds of automated document content analysis as well as interactive steps such as manual annotation of specific parts or browsing the obtained results. These steps provide additional views of the processed page that are related to some logical interpretation of some of the content parts. Therefore, the following artifact types form a *logical (interpretation) level* of the page description:

- The *AreaTree* models the important visual areas detected in the page and their nesting. Each area is represented by an *Area* element that has similar properties as a box. However, in contrast to the boxes that represent the physical output of the rendering engine, the meaning of the visual area is more general; the areas may be identified in a Page by simply filtering visually distinguished boxes [15] or by applying a page segmentation algorithm such as VIPS [6] or BCS [12]. In that case, the boundaries of the resulting areas and the organization of the entire area tree completely depend on the chosen segmentation method and its settings, and each visual area may consist of any number of boxes. The resulting AreaTree artifacts are normally derived from a Page artifact; however, FitLayout also provides different post-processing methods that allow to derive a new AreaTree form an existing one, for example by sorting the areas or joining them. Alternatively, the provided Web GUI may be used for creating the areas interactively, which is suitable for creating reference data sets as we discuss later in section 6.
- The *ChunkSet* on the other hand provides a "flat" view of the page consisting of a set of *text chunks*. With a text chunk, we understand a continuous part of the document text which is specific in any way. Typical means of extracting the text chunk include recognizing data type values (dates, numbers), matching regular expressions, named entity recognition, etc. The boundaries of the text chunks are not limited by the boundaries of boxes or visual areas, i.e. a text chunk may represent a substring of a box text or, on the other hand, cover the contents

of multiple boxes. However, each text chunk may have their source boxes or visual areas assigned in order to maintain the connections among the artifacts.
– The *LogicalAreaTree* provides the most abstract view of the page. It consists of *logical areas* where each logical area consists of any number of visual areas or text chunks that together form a logical entity that can be mapped to a target domain (e.g. a body of a published article consisting of multiple paragraphs – visual areas). The edges of the tree represent relationships among the logical areas that have some semantic interpretation (depending on the purpose for which the tree was created), as for example the relationship between a heading and contents of the article [16].

Finally, the considered outcomes of the entire process include the mapping of specific content elements of the analyzed page to a domain ontology (such as mapping a product title and description to the corresponding properties of the *Schema.org* ontology) or exporting the results in a structured form (e.g. a CSV file as we discuss later in section 6).

4. Ontological model for page description

For representing the artifacts, their properties and contents in RDF, we have designed a set of three ontologies that together provide a vocabulary for describing all aspects of the artifacts and their creation. A complete overview is shown in Figure 2. Formal definition of the ontologies using OWL and detailed documentation in HTML are available on the project web pages[4].

The organization of the ontologies corresponds to the levels of abstraction of the document representation as introduced in the previous section:

– Level 0: *FitLayout Core Ontology*. It defines the basic concepts, mainly the Artifact itself and its properties that provide details of how (using which method) and when the artifact has been created. For derived artifacts, the *hasParentArtifact* property points to their parent artifact they have been derived from.
– Level 1: *Box Model Ontology*. It represents the physical level of the page representation: the *Page* artifact, and the *Box* as its content element.
– Level 2: *Visual Area Ontology*. It provides means for describing the page at the logical level. It defines the *AreaTree*, *ChunkSet* and *LogicalAreaTree* artifacts and their corresponding content elements.

Most of the content elements (boxes, areas and text chunks) share the common *RectArea* superclass that generally represents a rectangular area with a given position on the rendered page (*Bounds*) and a set of font and color properties.

4.1. Rendered page representation

The representation of the rendered page is based on the Visual formatting model defined by the Cascading Style Sheets specification (CSS) [14] that defines a *box* as a basic unit of document content presentation in the page and the way the boxes are generated from the input HTML (DOM) elements. The corresponding *Box* class in the ontology has exactly the same meaning. The boxes are organized in a *box tree* using the *isChildOf* property: There is always a single *root box* that corresponds to the entire page, the leaf boxes correspond to the smallest atomic pieces of page contents (such as text or images) and the remaining boxes in the tree represent the boxes generated by HTML elements.

Note that although the structure of the box tree may look similar to the DOM structure, they are generally different. According to the CSS formatting model, a single HTML element may generate multiple boxes (as for example a paragraph containing multiple lines of text) or even no boxes (an invisible element) and the parent-child relationships among the boxes do not necessarily correspond to the parent-child relationships of the DOM elements because of different positioning schemes defined in the CSS specification.

[4] http://fitlayout.github.io/

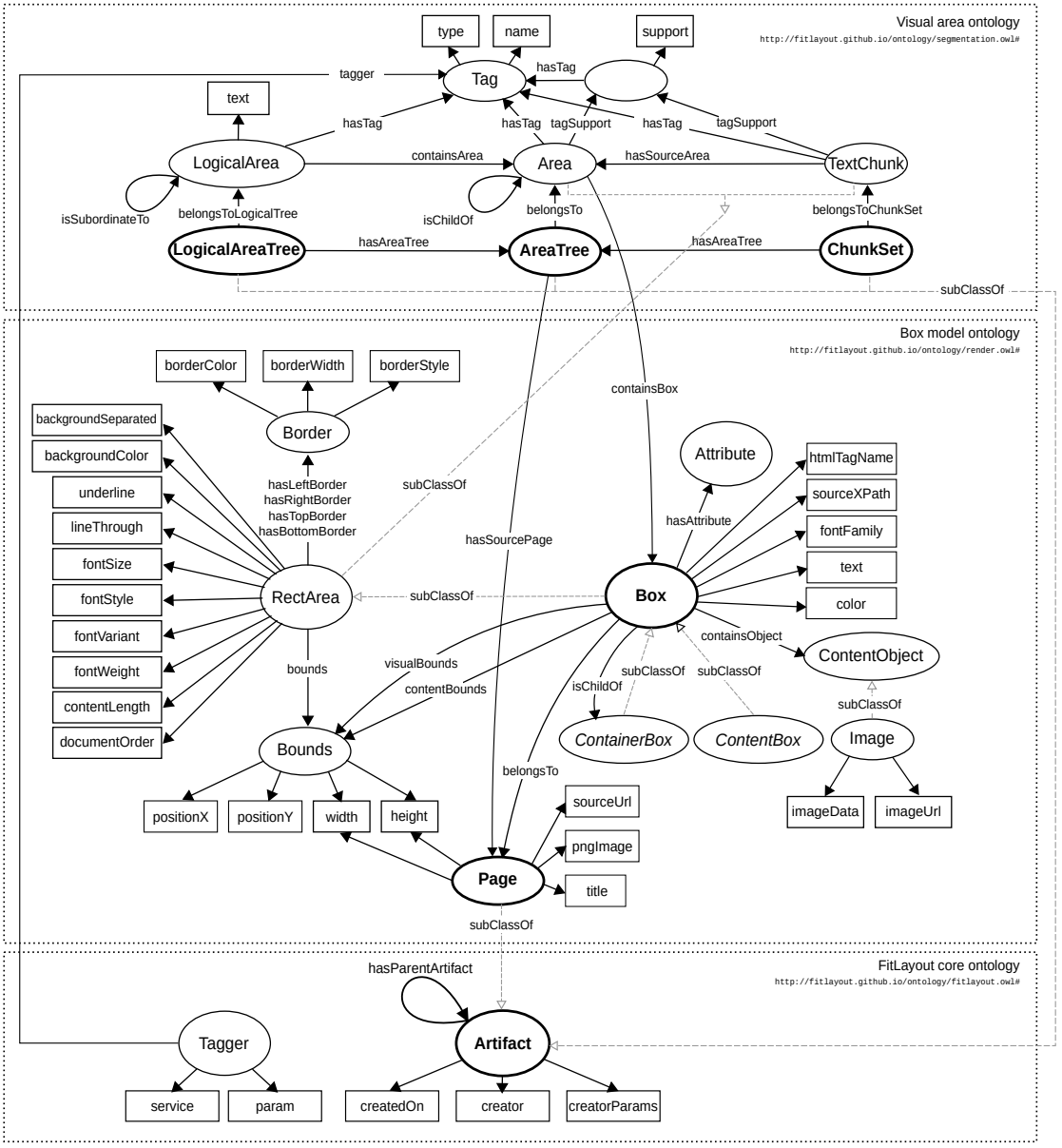

Fig. 2. FitLayout ontology levels: Basic concepts (FitLayout core ontology), detailed rendered page description (Box model ontology) and its logical interpretation (Visual area ontology).

4.2. Logical level representation

The logical views of the page are represented by the *AreaTree*, *ChunkSet* and *LogicalAreaTree* artifacts and the corresponding *Area*, *TextChunk* and *LogicalArea* elements. The areas and logical areas are organized in a tree; the text chunks that belong to a single ChunkSet create a simple collection only.

In addition, each logical content element may have a number of *Tags* assigned that are used for categorizing the content elements (e.g. the visual areas). The tags may be assigned either manually (for example using a graphical user interface) or automatically using so-called *Taggers*, which are built-in procedures that can be implemented using the FitLayout library.

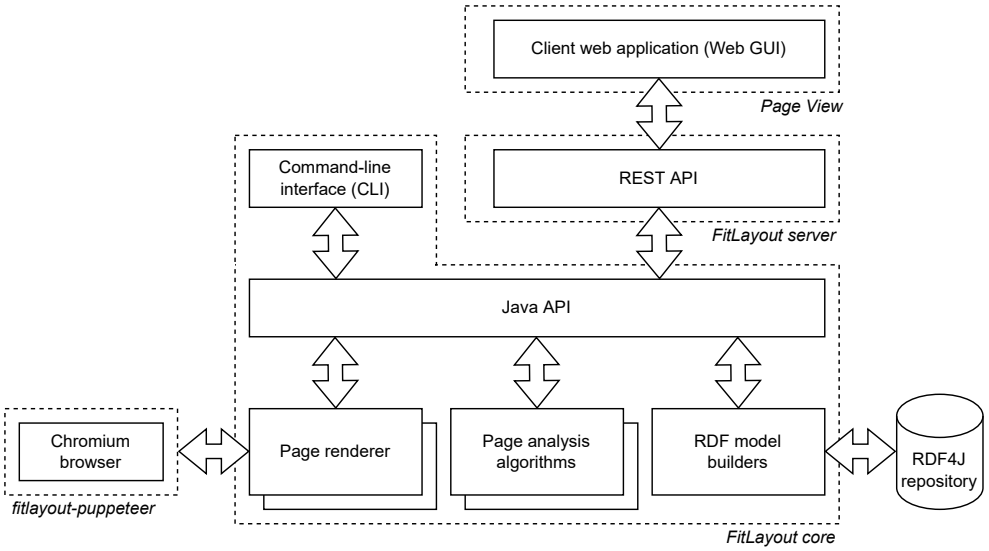

Fig. 3. FitLayout components and architecture.

5. FitLayout architecture overview

The entire FitLayout framework includes the ontologies introduced in the previous section and a set of tools that implement the page rendering, analytical algorithms, RDF storage and other components. The software architecture is shown in Figure 3. For maintainability reasons, the project is split to several software packages:

- *FitLayout core*[5] – a Java library that provides the core functionality of FitLayout including the data models, RDF serialization, artifact storage, algorithm implementations and a command-line interface.
- *FitLayout server*[6] – a server application implemented in Java (Jakarta EE platform) that provides a web application interface (REST API) for accessing the stored artifacts and invoking the implemented algorithms.
- *Page View*[7] – client web application that provides a graphical user interface for invoking the tasks and browsing the obtained results via a web browser. It is implemented in JavaScript using the Vue.js application framework and it communicates with the server application via the REST API.
- *fitlayout-puppeteer*[8] – a separate JavaScript application (on the node.js platform) that includes an installation of a Chromium browser and supporting JavaScript routines for rendering the target pages using the browser and obtaining the rendered page data.

The FitLayout Core software package defines a unified Java API for representing the individual artifacts and the contained elements, their transformation from and to their RDF models and an infrastructure for implementing and executing the individual analytical steps. An important part is a built-in RDF-based artifact storage implemented using RDF4J[9].

Additionally, the core library contains an extensible set of implemented, ready-to-use algorithms for the following tasks:

- *Page rendering*. Primarily, a rendering backend based on the open-source Chromium browser is provided. The browser itself together with supporting JavaScript routines is provided in the separate *fitlayout-puppeteer* software package.

[5] https://github.com/FitLayout/FitLayout
[6] https://github.com/FitLayout/FitLayoutWeb
[7] https://github.com/FitLayout/PageView
[8] https://github.com/FitLayout/fitlayout-puppeteer
[9] https://rdf4j.org/

– *Page analysis*. The package implements some popular page segmentation methods including VIPS [6] and BCS [12] and various area tree post-processing methods for sorting and filtering the visual areas.
– *Content classification and extraction*. A set of taggers is available that can recognize entities in document text by different methods including regular expressions, using Stanford NER [17] or by spotting occurrences of values used by the JSON-LD metadata provided within the page. Subsequently, the corresponding tags can be added to the visual areas or new text chunks can be extracted.

The library can be directly used for implementing Java applications for rendered web page processing[10]. It also includes a command-line interface (CLI) that allows executing the individual tasks such as page rendering, analysis and storage from command line. This is particularly useful for batch execution of the tasks as for example rendering large sets of input pages and their subsequent segmentation.

The FitLayout server and PageView client application together implement a client-server web application that provides a graphical user interface for interactive invocation of the tasks and graphical browsing of the obtained results. It is primarily intended for performing interactive experiments with different configurations of the algorithms.

For a convenient local installation, we provide ready-to-use docker containers on Docker Hub[11]. Although local installation is the most flexible and preferred way of using FitLayout, a running demo of the Web GUI is also available online[12].

6. Applications and testing

The main motivation behind FitLayout was the development of vision-based web page processing (mainly page segmentation and data extraction) algorithms and the related need to implement and evaluate the algorithms. Initially, in 2009, we run experiments with web content classification based on visual features [18] for which, it was necessary to create training and testing data sets of web pages with annotated semantics of specific content parts (such as article titles and authors). Due to lack of appropriate tools, we started to develop an ad-hoc solution that later evolved to the FitLayout tool set.

In our later paper dedicated to the BCS page segmentation method [12], we were able to use FitLayout for creating a testing data set in a collaborative way: We rendered a set of testing web pages, stored their complete RDF descriptions and using the FitLayout GUI, we let a group of volunteers to manually annotate the expected content segments. The annotation results were added to the RDF descriptions as the new visual areas. Finally, we compared the results of the tested algorithm with the manually annotated areas.

In 2015, we successfully applied FitLayout in the SemPub 2015 Challenge that consisted of extracting scholarly information from CEUR proceedings web pages. FitLayout allowed us to apply a vision-based approach to this task [15] that was evaluated as both the best performing and the most innovative one within the challenge [19]. Later, we generalized this approach for multiple domains by implementing different visual presentation pattern matching algorithms in FitLayout [20, 21].

In order to demonstrate the capabilities of FitLayout for creating larger data sets, for the purpose of this paper, we have prepared a demonstration data set of news web pages. From various news feeds, we gathered URLs of web pages containing published news articles together with JSON-LD structured data that have been processed in the following steps:

1. Each page was rendered using the Chromium browser, and the corresponding Page artifact was stored. Simultaneously, the JSON-LD structured data annotations contained in the page were extracted and stored in the same RDF repository.
2. Using the built-in FitLayout algorithms, we extracted the basic visual areas from the rendered box model.
3. Using the available taggers, we matched the page contents with the provided structured data values and extracted and tagged the corresponding text chunks.

[10] A set of sample application is provided in a separate repository at https://github.com/FitLayout/Demos
[11] https://hub.docker.com/u/fitlayout
[12] https://layout.fit.vutbr.cz

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX box: <http://fitlayout.github.io/ontology/render.owl#>
PREFIX segm: <http://fitlayout.github.io/ontology/segmentation.owl#>
PREFIX map: <http://fitlayout.github.io/ontology/mapping.owl#>
PREFIX schema: <http://schema.org/>

SELECT DISTINCT ?art ?chunk ?text ?fontSize ?fontWeight WHERE {
   ?chunk segm:hasTag ?tag .
   ?tag map:describesInstance ?art .
   ?tag map:isValueOf schema:headline .
   ?art rdf:type schema:NewsArticle .
   ?chunk segm:text ?text .
   ?chunk box:fontSize ?fontSize .
   ?chunk box:fontWeight ?fontWeight .
}
```

Fig. 4. A SPARQL code query that lists all text chunks that correspond to the article headlines from the data set together with their text, used font size and font weight.

As the result, we obtained three artifacts for each processed page: a Page artifact representing the rendered page (a box tree), an AreaTree artifact containing the distinguishable visual areas (an area tree) and a ChunkSet artifact containing text chunks that correspond to the occurrences of the structured data fields provided in JSON-LD metadata, such as occurrences of article titles, descriptions, authors and other provided properties.

The entire data set has the form of an RDF graph where the sub-graphs that correspond to the individual artifacts are distinguished by context URIs (therefore, the data set is published in the N-Quads serialization). It allows to combine multiple aspects of the page content in the queries from the low-level rendering details up to the mapping to the structured metadata. For example, the sample query in Figure 4 produces a CSV result that contains all text chunks from the entire data set that represent the article headlines (provided as the `schema:NewsArticle` property value in the JSON-LD data), and the font size (in pt units) and font weight used for rendering them. Among other applications, such data is potentially useful for training a classifier as we have shown in [18].

For convenience, we have created two versions of the data set: A large one (2554 pages, 7662 artifacts, ~118 million RDF triples) and a smaller one, which is a subset of the larger one (825 pages, 2475 artifacts, ~38 million RDF triples). The data sets together with the scripts used for their creation are available at Zenodo[13]. A read-only repository containing the smaller data set is available for browsing online via the public FitLayout GUI [14].

7. Conclusions

FitLayout provides a platform for implementation and evaluation of web page analysis methods that work with the visual presentation of the contents and, at the same time, a framework for creating flexible web page processing workflows. The FitLayout architecture with the central RDF repository allows to build and store a platform-independent model of a page during all stages of its processing from an initial rendering up to its segmentation or linking with structured data sets.

This architecture allows archiving the rendered pages and creating data sets containing the rendered page details supplemented by detected visual areas, metadata tags, manually created annotations and other additional information. The data sets can be stored as RDF graphs or exported in other structured formats, for example using SPARQL queries tailored for a particular application. At the same time, the persistently stored artifacts can be re-used at any time and processed in alternative ways. For example, when comparing different page segmentation methods, they may all use a single input Page artifact, which improves the efficiency by avoiding repetitive rendering (when compared for example with [8]) and assures that all the compared methods are provided with identical input pages.

FitLayout has been developed for more than a decade and practically applied in many different scenarios as discussed in Section 6, including an awarded application in the SemPub 2015 Challenge. It is published as open source and we believe that its use can also be beneficial for other research teams in the related areas.

[13] https://doi.org/10.5281/zenodo.6962687

[14] https://layout.fit.vutbr.cz/browser/#/r/8b287754-4264-4578-a37d-b2c5e1e6deb8

Acknowledgments

The work is supported by the Brno University of Technology project “Application of AI methods to cyber security and control systems”, no. FIT-S-20-6293.

References

[1] C. Bizer, R. Meusel, A. Primpeli and A. Brinkmann, Web Data Commons - Microdata, RDFa, JSON-LD, and Microformat Data Sets - Extraction Results from the October 2021 Common Crawl Corpus, University of Mannheim, 2022, Accessed on 2022-06-08. http://webdatacommons.org/structureddata/index.html#results-2021-1.

[2] S.R. Jayashree, G. Dias, J.J. Andrew, S. Saha, F. Maurel and S. Ferrari, Multimodal Web Page Segmentation Using Self-Organized Multi-Objective Clustering, *ACM Trans. Inf. Syst.* **40**(3) (2022). doi:10.1145/3480966.

[3] A. Namboodiri and A. Jain, Document Structure and Layout Analysis, in: *Digital Document Processing*, B.B. Chaudhuri, ed., Advances in Pattern Recognition, Springer London, 2007, pp. 29–48. ISBN 978-1-84628-726-8.

[4] A. Kravchenko, Large-scale holistic approach to Web block classification: assembling the jigsaws of a Web page puzzle, *World Wide Web* **22**(5) (2019), 1999–2015. doi:10.1007/s11280-018-0634-6.

[5] S. Shi, C. Liu, Y. Shen, C. Yuan and Y. Huang, AutoRM: An effective approach for automatic Web data record mining, *Knowledge-Based Systems* **89** (2015), 314–331.

[6] D. Cai, S. Yu, J.-R. Wen and W.-Y. Ma, *VIPS: a Vision-based Page Segmentation Algorithm*, Microsoft Research, 2003.

[7] A. Sanoja and S. Gancarski, Block-o-Matic: A web page segmentation framework, in: *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, 2014, pp. 595–600. doi:10.1109/ICMCS.2014.6911249.

[8] J. Kiesel, L. Meyer, F. Kneist, B. Stein and M. Potthast, An Empirical Comparison of Web Page Segmentation Algorithms, in: *Advances in Information Retrieval*, D. Hiemstra, M.-F. Moens, J. Mothe, R. Perego, M. Potthast and F. Sebastiani, eds, Springer International Publishing, Cham, 2021, pp. 62–74. ISBN 978-3-030-72240-1.

[9] Q. Hao, R. Cai, Y. Pang and L. Zhang, From One Tree to a Forest: A Unified Solution for Structured Web Data Extraction, in *SIGIR '11*, Association for Computing Machinery, New York, NY, USA, 2011, pp. 775–784–. ISBN 9781450307574. doi:10.1145/2009916.2010020.

[10] M. Bronzi, V. Crescenzi, P. Merialdo and P. Papotti, Extraction and Integration of Partially Overlapping Web Sources, *Proc. VLDB Endow.* **6**(10) (2013), 805–816–. doi:10.14778/2536206.2536209.

[11] J. Kiesel, F. Kneist, M. Alshomary, B. Stein, M. Hagen and M. Potthast, Reproducible Web Corpora: Interactive Archiving with Automatic Quality Assessment, *J. Data and Information Quality* **10**(4) (2018). doi:10.1145/3239574.

[12] J. Zelený, R. Burget and J. Zendulka, Box clustering segmentation: A new method for vision-based web page preprocessing, *Information Processing and Management* **53**(3) (2017), 735–750. doi:10.1016/j.ipm.2017.02.002.

[13] J. Kiesel, F. Kneist, L. Meyer, K. Komlossy, B. Stein and M. Potthast, Web Page Segmentation Revisited: Evaluation Framework and Dataset, in: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 3047–3054–. ISBN 9781450368599. doi:10.1145/3340531.3412782.

[14] B. Bos, Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification, W3C Working Draft, W3C, 2016, http://www.w3.org/TR/2016/WD-CSS22-20160412/.

[15] M. Milička and R. Burget, Information Extraction from Web Sources based on Multi-aspect Content Analysis, in: *Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015*, Communications in Computer and Information Science, Vol. 2015, Springer International Publishing, 2015, pp. 81–92. ISSN 1865-0929. ISBN 978-3-319-25517-0.

[16] R. Burget and P. Smrz, Extracting Visually Presented Element Relationships from Web Documents, *International Journal of Cognitive Informatics and Natural Intelligence* **7**(2) (2013), 13–29.

[17] J.R. Finkel, T. Grenager and C. Manning, Incorporating non-local information into information extraction systems by Gibbs sampling, in: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, 2005, pp. 363–370.

[18] R. Burget and I. Rudolfová, Web Page Element Classification Based on Visual Features, in: *1st Asian Conference on Intelligent Information and Database Systems ACIIDS 2009*, IEEE Computer Society, 2009, pp. 67–72. ISBN 978-0-7695-3580-7. doi:10.1109/ACIIDS.2009.71.

[19] A. Di Iorio, C. Lange, A. Dimou and S. Vahdati, Semantic Publishing Challenge–Assessing the Quality of Scientific Output by Information Extraction and Interlinking, in: *Semantic Web Evaluation Challenges*, Springer, 2015, pp. 65–80.

[20] R. Burget, Information Extraction from the Web by Matching Visual Presentation Patterns, in: *Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia*, Lecture Notes in Computer Science vol. 10579, Springer International Publishing, 2017, pp. 10–26. ISBN 978-3-319-68722-3. doi:10.1007/978-3-319-68723-0_2.

[21] R. Burget, Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents, in: *15th International Conference on Web Information Systems and Technologies*, SciTePress - Science and Technology Publications, 2019, pp. 326–333. ISBN 978-989-758-386-5. doi:10.5220/0008350103260333.