

USING PDF DOCUMENTS FOR RAPID AUTHORING OF REUSABLE ELEARNING CONTENT IN **LOXtractor**

Projektarbeit am Fachbereich Informatik der Technischen
Universität Kaiserslautern, in Kooperation mit dem Deutschen
Forschungszentrum Künstliche Intelligenz (DFKI), Kaiserslautern

Frederick Schulz

June 12, 2008

Erklärung der Selbständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Zitate sind deutlich kenntlich gemacht.

Kaiserslautern, June 12, 2008

Frederick Schulz

Contents

1	Overview	1
1.1	Introduction	1
1.2	Learning, Informal Learning and Elearning	1
1.3	Elearning in the Workflow: SLEAM	2
2	Initial State and Task Description	7
2.1	Initial State of LOXtractor	7
2.2	Assigned Tasks and Their Motivation	8
2.2.1	Extending the Choice of Input Formats	8
2.2.2	Additional Improvements	10
3	User Guide	11
3.1	Step 1: Creating a New Learning Object Project	13
3.2	Step 2: Importing a PDF File	13
3.3	Step 3: Importing Written or Copied Text	14
3.4	Step 4: Editing Content and Metadata	16
3.5	Step 5: Extracting Learning Objects	16
4	Technical Realization	19
4.1	Technical Background	19
4.1.1	Eclipse RCP	19
4.1.2	Plugin Architecture	19
4.1.3	The PDF File Format	21
4.2	The PDF Parser Plugin	22
4.2.1	Image Extraction	23
4.2.2	Metadata Extraction	23
4.2.3	Text Extraction	24
4.2.4	Solved and Unsolved Problems	26
4.3	The Plain Text Parser Plugin	26
5	Conclusions	27
5.1	Related Work	27
5.1.1	PDF content extraction	27

Contents

5.1.2 Optical Character Recognition	28
5.2 Possible Improvements and Extensions	28
5.3 Conclusion	28
Bibliography	31

Overview

1.1 Introduction

This thesis presents and details my work in expanding the rapid authoring tool prototype **LOXtractor**, which has been developed at DFKI by Markus Ludwar as a diploma thesis in 2006. The focus of my work was in adding additional input sources, namely PDF files.

In the remaining pages of chapter 1 I will present the historical, scientific and applicational contexts of the **LOXtractor** tool. Chapter 2 will give an overview of **LOXtractor** itself while chapter 3 will be focused on my improvement and expansion work.

1.2 Learning, Informal Learning and Elearning

Learning, according to ([Doh01] p.3), is the process of acquiring impressions, informations and requirements from the environment and constructing new knowledge by relating these to existing correlations of knowledge, imagination and explanation. This leads to new competencies of acting and understanding. Learning is categorized as *informal*¹ if it takes place outside of organisations or events especially dedicated to it (like schools or seminaries) and instead happens during work or leisure time and without an explicit agenda or a designated instructor.

LOXtractor is a tool for generating elearning content in informal learning environments embedded in the workflow. Elearning, short for *electronic learning*, is used as a generic term for all learning activities using *new media* (see [RR02] p.16). Informal elearning happens all the time in all kinds of work environments. A recent survey by TNS Infratest, conducted for the german federal ministry of education and research (BMBF) during the european adult education survey (AES) (see [RB08] p.31) states that 35% of all adults aged 19 to 64 used computers and the in-

¹ At least it is in the scope of this work; there is no need to use one of the more elaborate definitions commonly used in scientific literature. See e.g. [Doh01], p.18ff.

ternet to improve their knowledge in the last twelve months. All these people (and many more) benefited from informal elearning. The most obvious example for informal elearning in the workplace is surfing the web for information needed to solve the current task at hand. That may be finding the mail address or telephone number of a correspondent, reading the online documentation of a software program or programming language to be used, or browsing message boards in search for an explanation of a sudden error message. This type of directed information gathering is called *elearning by distributing* ([RR02] p.16), because the media is only used to distribute information. The requirements for the learner are high (see [RR02] p.16), since information found on the web is (most often) not prepared for effective learning and contains a great share of irrelevant details (see [RL07a] p.1). Depending on the skills and competencies of the learner, the cost for preparing these raw information to extract the learning-relevant parts can be quite high.

Elearning content can come in various forms like direct representations of formal learning sessions (lectures, seminars, etc.) in new media formats like video recordings or podcasts. It can also consist of self-learning courses and interactive learning tools. These content forms are rather monolithic and contain many single pieces, often connected rather loosely. Separating or extracting these pieces for reuse is time consuming (see [Lud06] p.9). For the efficient storage a more finely grained unit is desirable. The form which is most interesting in the scope of this paper is the *learning object*. A learning object is a small, self-contained unit of informational content that cannot be divided any further (see [Lud06] p.9). The format of these contents can vary greatly. Most times it will consist of a single text section, a single image, or a table, but videos, audio files and interactive content are also possible. In addition to its actual content, the learning object may and should contain metadata describing its content (keywords, tags and similar) and a mapping locating this learning object in an ontology. This provides a both machine and human accessible way to find learning objects. A set of selected learning objects can easily be assembled to a bigger course unit – which can itself be described by metadata – and from these the bundling of full elearning courses is possible. So learning objects with high quality metadata are a highly reusable and flexible form of elearning content. This makes the reusable learning object a natural candidate to be used as a storage unit in knowledge management systems (see [Lud06] p.9f).

1.3 Elearning in the Workflow: SLEAM

Elearning by distributing, as described above, happens very often², consumes much time and resources and is often redundant since coworkers have already collected and prepared this information for themselves in the past and learning directly from those coworkers is both neglectable in frequency (see [RB08] fig.16) and doubles the cost due to two people being deterred from

2 Information technology workers spend an estimated 7 hours per week with information gathering, according to a study performed by The Ridge Group [Rid03].

work (see [Rid03]). None the less, informal learning in the workplace is important, even a condition for survival. ([Tri02] p.3) So, from an economic perspective, the learning's main negative effect, the cost in time, must be reduced as much as possible. Especially the redundancy in information search, retrieval and selection can be reduced greatly by implementing a knowledge management system. A working knowledge management system can shift a substantial part of ineffective 'elearning by distribution' to *elearning by interacting* ([RR02] p.16), using the advantages of new media not only for faster access to information but for facilitating the learning process, too.³ Workers using learning objects as units of information that are didactically prepared to contain higher information density and are enriched with metadata to facilitate their location while solving their current learning goal spend less time locating and preparing the necessary information. This process of transforming knowledge from implicit personal knowledge to explicit, collective forms as externalization in the epistemological and ontological dimension is described as the knowledge spiral concept of knowledge management by Nonaka and Takeuchi (see [NT97]). But simply representating and collecting knowledge, called 'squirrel knowledge management' [Sch01] is not enough. There have to be adequate organisational, pedagogical and at last technical environments facilitating and cultivating the usage of stored knowledge ([RR02] p.13f).

However, even with the best environments, all elearning solutions need someone to maintain and fill the knowledge repository, and for small and middle-sized enterprises (SME) hiring a specialized employee for this task or outsourcing it to a service provider is economically not feasible. This might be the reason why less than 25% of enterprises with 500 to 1000 employees use elearning solutions, and even less smaller enterprises do so.⁴

The german research center for artificial intelligence (DFKI), during the project *Task Embedded Adaptive eLearning (TEAL)*, devised an optimized process for workflow embedded authoring of elearning objects called *SLEAM* (see [RL07a]).

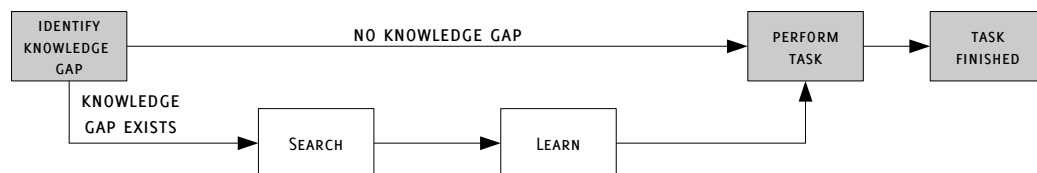


Figure 1.1: Normal task solving workflow

Figure 1.2 shows a schematic overview of the SLEAM process. SLEAM is an acronym for the sequence of activities – *Search Learn Extract Annotate Map* – that describes this method of

3 See *core statement 1* in [RR02]: 'A user friendly preparation of content can facilitate [...] learning processes.'

4 Estimated according to a survey of 40 'experts' [Ins06].

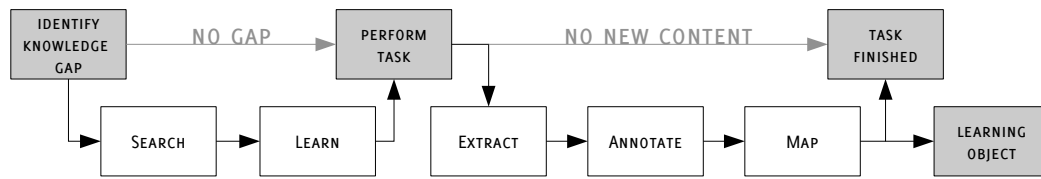


Figure 1.2: The SLEAM process

elearning object authoring. The normal task solving workflow is extended with additional steps to create a metadata-enriched learning object for use in a knowledge management system.

In the normal workflow (figure 1.1), for each task given the worker first determines if he has all knowledge necessary for the completion of the task. If one or more parts of knowledge are missing this is called a *knowledge gap*. The worker then searches for information and learns the missing knowledge, then proceeds with the task, now able to solve it since no more knowledge is missing. However, all effort done in the course of searching and learning is lost for everyone else but the worker himself.

The SLEAM process (figure 1.2) strives to conserve this effort. After *searching*, *learning* and completing the task, the worker *extracts* all relevant information from his sources, *annotates* them with metadata describing content and context and *maps* them into an ontology for better retrieval, thus creating a metadata enriched learning object ready to be deployed to the company's knowledge management system. Thus, the output of the workflow is not only the solution to the task at hand, but also a learning object. This learning object conserves part of the effort invested in information retrieval. A coworker assigned to a similar task can now locate this learning object in the company's knowledge database with the help of the metadata, keywords, and the ontology the learning object is mapped to. Thus, a lengthy and expensive search or a disturbance of the worker already possessing the competence or knowledge can be avoided – at least partially.

To create substantial advantages that justify the necessary investments, the cost of learning object creation must be significantly lower than what saving it creates. This is only realisable with an appropriate toolchain.

Rostanin and Ludwar suggest some requirements for toolchains supporting SME's elearning strategies in [RL07a] (figure 1.3). They conduct evaluations of existing learning content authoring systems with regard to these requirements in [RL07a] and [Lud06]. These evaluations show a great deficiency of elearning content authoring tools suited for SMEs. Specialized toolchains like RELOAD (see [Uni]) or EXPLAIN (see [imc]) require trained operators and content collected with general purpose software – like Microsoft Office, weblogs, or wikis – requires extensive postprocessing to maintain a well-ordered state. This lack of appropriate tools led to the development of the **LOXtractor** software described in the following chapter.

- The learning object authoring process...
 - must be performed by company insiders without deterring them from work for too long and
 - should focus on import and (partial) reuse of existing documents instead of from scratch creation.
- The resulting learning objects...
 - are fine-grained learning objects instead of full executable learning courses and
 - are annotated with metadata according to a standard format to allow import in and retrieval from learning content management systems.

Source: [RL07a]

Figure 1.3: Requirements for elearning strategies in small and middle sized enterprises

Initial State and Task Description

2.1 Initial State of LOXtractor

As stated above, there are – as of yet – no tools suited for supporting the SLEAM process of workflow embedded learning object authoring. Authoring tools for elearning most times focus on content creation from scratch, need specially trained authors or produce large monolithic training courses instead of fine grained learning objects desired for informal learning on the workflow (see [RL07a]). To proof the advantages of SLEAM and to conduct case studies, an experimental authoring tool tailored for SLEAM – **LOXtractor** – was implemented by Markus Ludwar as a diploma's thesis in 2006 (see [Lud06]).

This first **LOXtractor** version allows acquiring content from html documents. These can be wikis relying on the mediawiki software (see [Wika]) – e. g. wikipedia (see [Wikb]) – or arbitrary websites. The *parser* retrieves the documents from the internet and cleans up malformatted ones. The well-formed HTML documents are then analysed and the markup elements are used to transform the linear, textual format of HTML to tree structures. The page's `<title>` is transformed to the root node. Headlines – `<h1>` to `<h6>` – are represented by inner nodes. Higher numbers are treated as children of lower numbers. The leaves are `<p>` and `<div>` elements and text found directly in the `<body>` element. In most cases this should reflect the hierarchy of the information contained in the document very well. Images are represented by leaves, like text sections. At first only the image URL is recorded. The image itself is downloaded only when it is displayed or exported. Tables are treated separately to conserve their structure.

That tree representation of the document is then displayed in the *tree view* of **LOXtractor**. In the side panel on the left (figure 2.1, 1) the *tree view* displays the tree nodes. For the selected tree node, the content can be viewed in the content panel (figure 2.1, 2). The lower panel shows and modifies the metadata (figure 2.1, 3) and learning concept ontology (figure 2.1, 4) properties.

The *tree view* provides tools to delete, concatenate and reseparate nodes in order to trim and restructure information. After this, selected parts can be exported to learning objects. The *ex-*

2 Initial State and Task Description

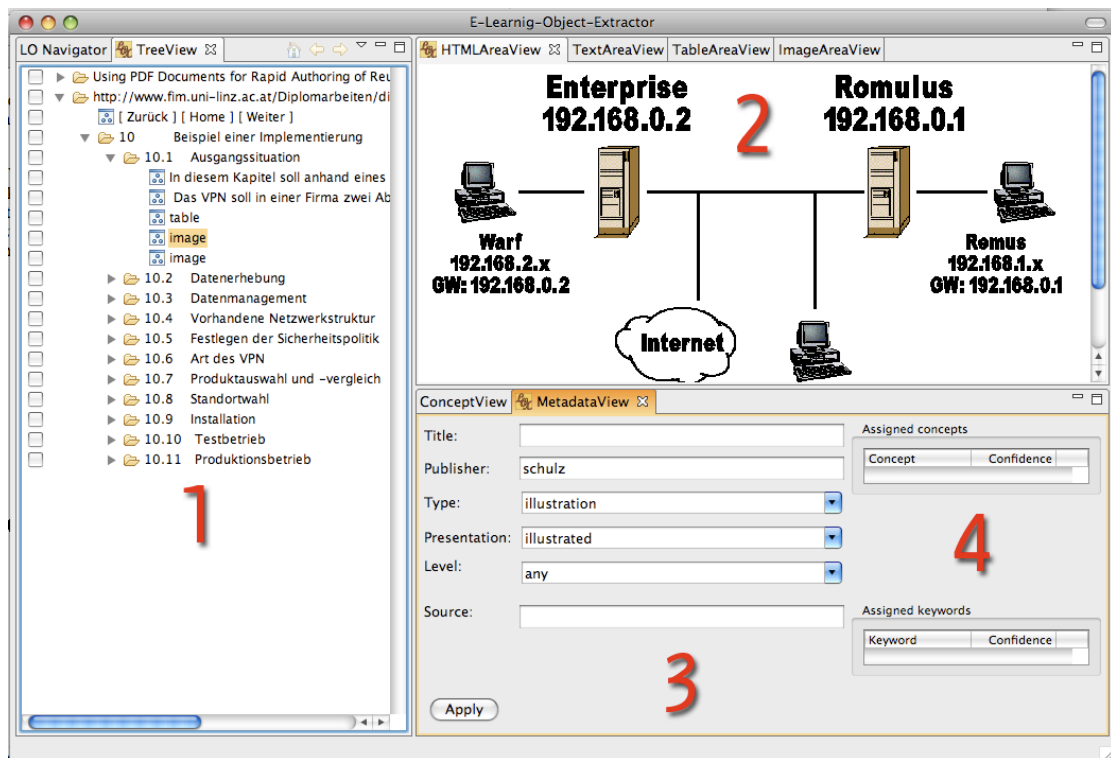


Figure 2.1: **LOXtractor** window: 1. side panel with *tree view* of document structure, 2. content panel with image, 3. metadata panel, 4. concept and keyword panel

tractor creates learning objects in SCORM and DAMIT (see [DFK]) format. SCORM has been chosen as the main output format for its widespread use, its role as a standard for learning content exchange, and its well-documented, XML-based format definition.

Since these features are not important for my work on **LOXtractor**, I refer to [Lud06] for a more detailed description.

2.2 Assigned Tasks and Their Motivation

While a first, working version of the **LOXtractor** tool exists, there is still a potential for improvements and extension. Ludwar presents some ideas in [Lud06], pages 81ff. One of them is the extension of the parser section with additional file types. The tasks assigned to me for my projects thesis will be described in the following sections.

2.2.1 Extending the Choice of Input Formats

One of the disadvantages of **LOXtractor** is its limitation to receive the input only from web sites (see [Lud06] p.89). My task was to extend the spectrum of file types usable as input for **LOX-**

SCORM

‘The Sharable Content Object Reference Model (SCORM) defines a Web-based learning “Content Aggregation Model (CAM)” and “Run-Time Environment” (RTE) for learning objects.[...] This reference model aims to coordinate emerging technologies and commercial and public implementations. The SCORM applies current technology developments to a specific content model by producing recommendations for consistent implementations to the vendor community.’

Source: [Adv]

Figure 2.2: Self-description of SCORM

tractor. I decided to implement an import tool – called *parser* in the **LOXtractor** context – for *Adobe PDF* files.

The *Portable Document Format* was developed in 1993 as an extension to the *PostScript* page description language, extending it with document structure and interactive navigation features (see [pdf06] p.23). Over the course of 15 years and 8 versions many features for collaborative editing, digital signing and archiving, security and digital rights management, and accessibility have been added. PDF has become a *de facto* standard for prepress systems and electronic document exchange (see [pdf06] p.24). In many places, a flow of PDF files replaces the previous paper workflow. A statistical survey from 2002 reported that an estimated 10% of all files indexed by Google are in PDF format (see [May02]). This is not surprising, since PDF has many properties qualifying it for electronic document exchange:

- Identical rendering on a great variety of platforms, ranging from all kinds of screen devices (cellphones to PCs) to home or office printers and industry scale printing machines, made possible by features like font embedding (see [pdf06] p.39f).
- Direct usability on different platforms without need for character set or end-of-line character conversion (see [pdf06] p.38).
- A well documented and publicly available format definition¹, leading to a great variety of software for creating, displaying and modifying PDF files on many platforms.
- Up- and partial downward compatibility, allowing the continued use of documents created with outdated versions of PDF (see [pdf06] p.42).
- The ability to protect files from manipulation (see [pdf06] p.41f).

¹ Several subset definitions like PDF/A, PDF/X and PDF/E are published ISO standards since 2001. PDF 1.7 was declared *ISO standard 32000* in december 2007.

So one can be sure to encounter the portable document format while searching for information through the internet. Unfortunately, the portable document format is less optimal for information extraction. Compared to markup languages like (X)HTML, it is not focused on transporting document structure and content but on transporting a pure visual information (see [LB95]). While recent developments, especially *tagged PDF* make up for that deficiency, their use is very limited, due in part to the inability of most PDF creation software to use these features. (see [pdf06] p.883ff). The facts stated above show clearly that PDF import for **LOXtractor** was both a necessity and a challenging task.

The requirements for the PDF parser were the following:

- Extraction of all text content
- Preserving the structure of the text
- Preserving as much text style information (text size, font information) as possible
- Extraction of images
- Extraction of metadata (title, author, subject and keyword information)
- Compatibility with PDF files created with different software and with different layouts

I also considered a parser for *office documents* – documents created with Microsoft Office, Open Office, or similar programs. While not as abundant as PDF files on the internet these files are none the less spread widely, especially in office environments. Ultimately, I decided against implementing such a parser for the following reasons. First, there are many different formats in circulation: doc, ppt, odf, odp, docx, just to name a few. Either, many different parsers would have to be implemented, or a suitable intermediate format – and converters from all other formats to this format running on all target platforms – would have to be found. Second, the access of office documents in a Java program is difficult since not many libraries exist. Third, the conversion from these file formats to PDF is trivially easy and either build-in or easily available – even for free – with the different software packages. Hence a parser for PDF documents would also allow the import of these documents with minimal additional effort on the user's side.

2.2.2 Additional Improvements

While **LOXtractor** now supports conversions from web pages and PDF documents to learning objects it was still not possible to enhance the learning object with self-written annotations. To add explanations and examples in one's own words, in the current state the user would have to publish it to a web site which is then parsed by **LOXtractor**. To avoid this unnecessary complication I added a pure text parser. This enables the user to copy and paste text from various applications or write annotations directly into **LOXtractor**. For branding and recognition purposes, a logo used as application icon and splash image was designed.

3

User Guide

When first starting **LOXtractor**, an empty workplace is shown. It should seem familiar to users experienced with Eclipse or other Eclipse RCP based applications. The left part is the *navigator view*, where all imported documents and exported learning objects for each learning object project are presented. This left pane can be toggled to a *tree view* of all currently opened imported documents. Then the area to the right displays contents and metadata for the selected node in the left panel (see figure 3.2).

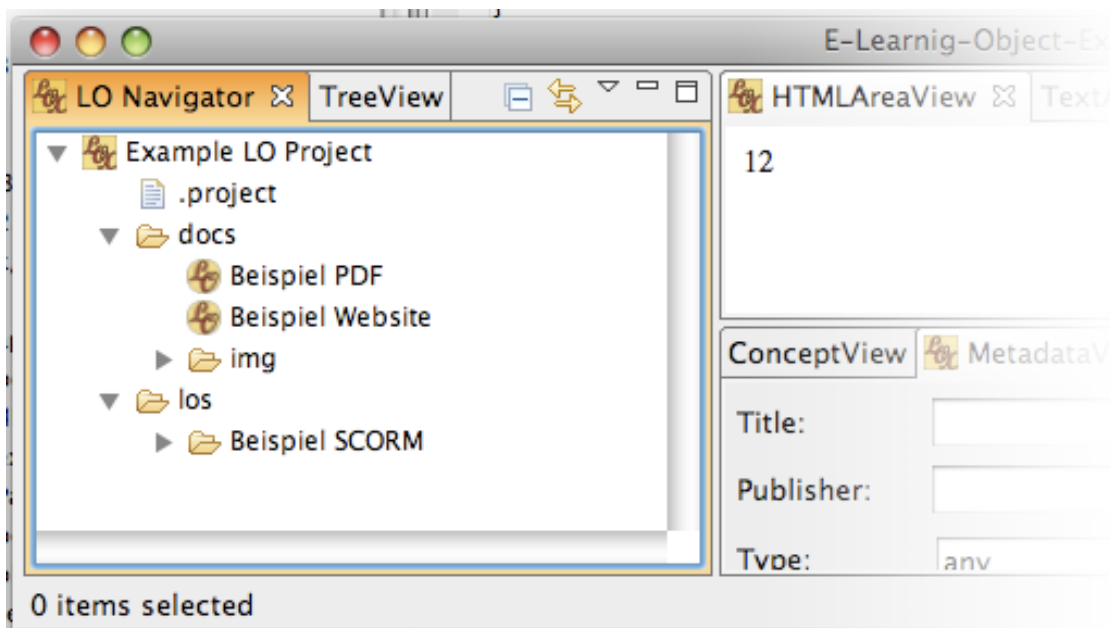


Figure 3.1: *Navigator view* in the left panel

Figure 3.1 shows the *navigator view* with one learning object project containing one imported document. Each learning object project provides two folders: `doc` for the storage of imported

documents with a subfolder `img` for images extracted from PDF files and `los` for the storage extracted learning objects.

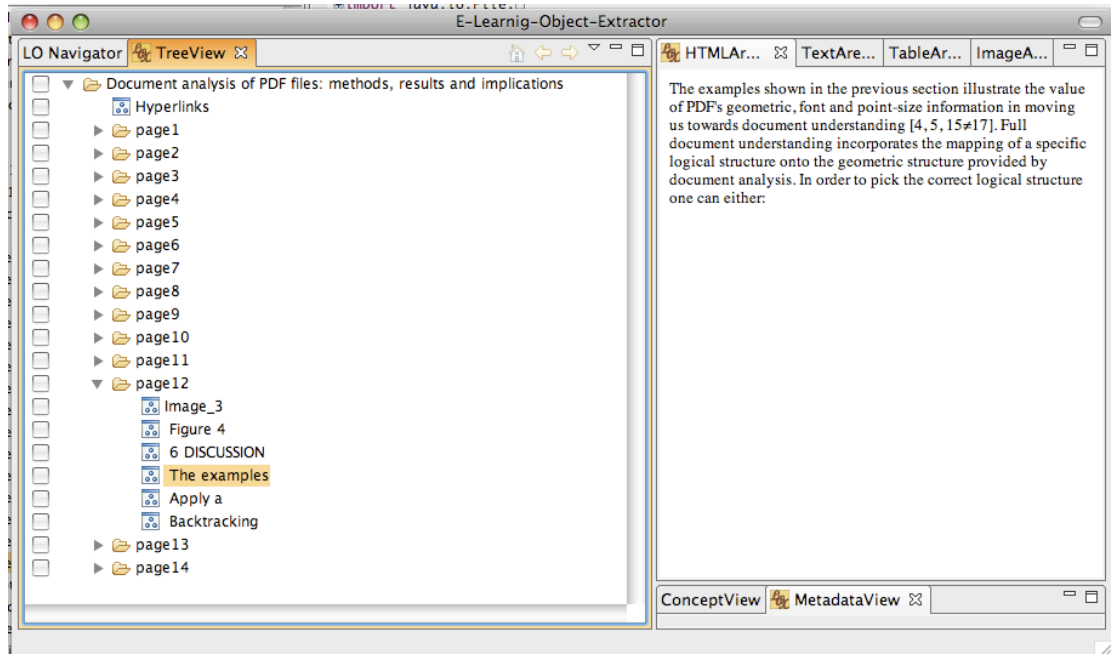


Figure 3.2: *Tree view* in the left panel, contents and metadata of selected node in the right panels

Figure 3.2 shows the *tree view* with the document seen in the *navigator view* now opened. One of the paragraphs of the first page is selected, and its contents are displayed in the upper half of the right window and its metadata in the lower half.

The **LOXtractor** workflow can be divided in the following steps:

1. Creating a New Learning Object Project
2. Importing a PDF File (repeated as necessary)
3. Importing Written or Copied Text (repeated as necessary)
4. Editing Content/Metadata
5. Extracting Learning Objects

In the following paragraphs, the steps of the workflow will be described from a user's perspective:

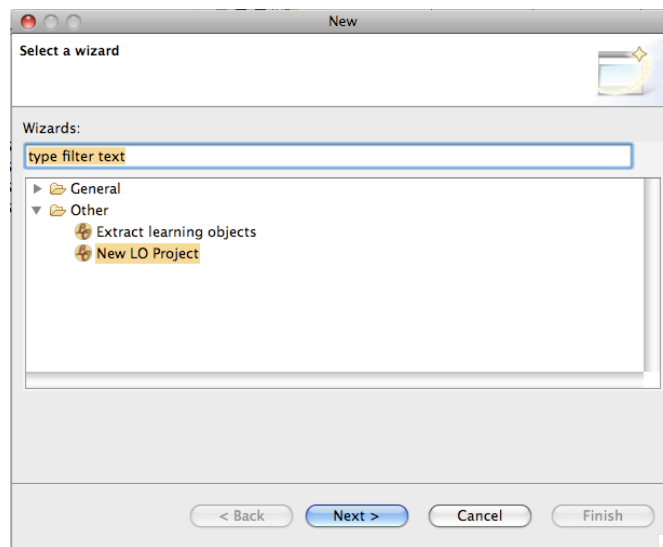


Figure 3.3: New project window

3.1 Step 1: Creating a New Learning Object Project

The first step in the creation of learning objects for a certain topic is to establish a container for the imported documents and exported learning objects. In eclipse terminology this is called a *project*. Projects are created by the *project creation wizard* accessible from the *File* menu. In this case we want to create a *New LO Project*. The next step inquires a name for the project. It might be a good idea to name the project after the concept to be explained. Finishing the wizard presents us with the *navigator view* again, now containing a node for the project labelled with the chosen name and containing the folders `docs` and `los`, as well as an eclipse-specific `.project` file which is of no interest to us. We can now start to add content.

3.2 Step 2: Importing a PDF File

To import a PDF file, we first switch the left panel to *tree view*. The context menu item *Select Parser* brings up the *parser wizard* (figure 3.4). By default, the *Wikipedia Parser* is selected in the *Select Parser* radio group. We have to change that to *pdf Parser*. You will notice that the layout will change to reflect the different input needed by the PDF parser compared to the wikipedia parser. Now we need to supply some information: *LO Group Name* is the label of the node representing our documents in the *navigator view*. In the big edit box in the middle we insert the full paths of the PDF files we want to extract, separated by semicolons. Clicking the *Choose File...* button opens a file dialog box which allows browsing for a file and automatically adds it to the box. Finally, we need to specify a *Source Solder*. That should be the `docs` folder of the learning object project our document is belonging to. Finally, press the *Finish* button and wait

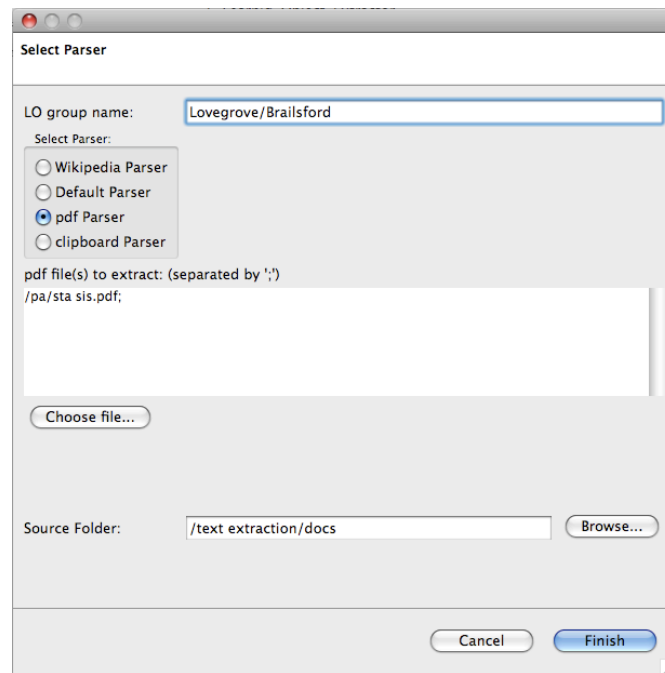


Figure 3.4: PDF parser dialog window

for the process to complete. If all went well, the *tree view* should come back to focus and display the extracted document (figure 3.2). The document is divided into pages and each page into paragraphs which are labeled with the first words they contain. Images are enumerated and included with the page they appear on and hyperlinks from the whole document are collected in a special node. While a separation into pages is given by the PDF file structure, paragraph detection is done with heuristics based on locality and common properties like font size and font face. While not always perfect, shortcomings can easily be corrected by manually joining paragraphs that were unnecessarily separated. A detailed description of the paragraph joining algorithm can be found in the developer's section (section 4.2).

3.3 Step 3: Importing Written or Copied Text

To import plain text written or copied from other applications we select the *clipboard Parser* in the *Select Parser* radio group of the *parser wizard* window. *LO Group Name* is filled with the title we like to give our piece of text and *Source Folder* is set to the `docs` folder of the learning object project our text is belonging to. After clicking *Finish*, the *tree view* should be back to front and we can begin selecting and editing content for the upcoming learning object creation.

3.3 Step 3: Importing Written or Copied Text

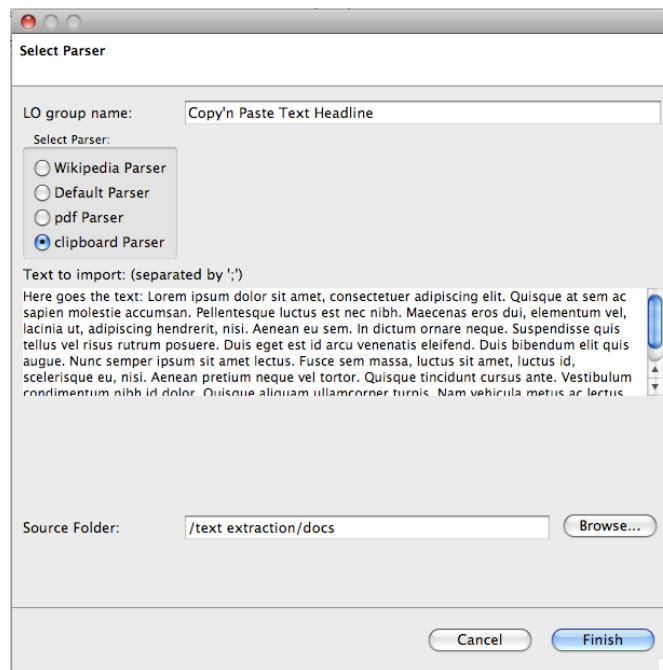


Figure 3.5: Text parser dialog window

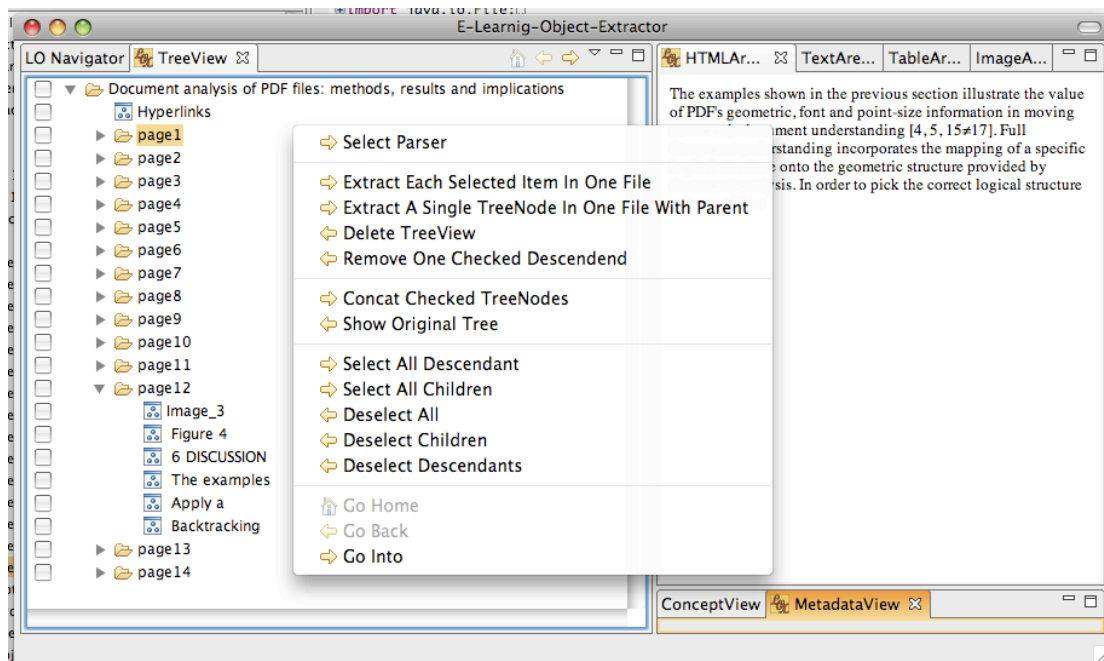


Figure 3.6: Context menu with node manipulation options

3.4 Step 4: Editing Content and Metadata

To edit the contents of imported documents, they have to be opened in the *tree view*. New documents are opened automatically after completing the *parser wizard*, older documents can be opened manually from their context menu in the *navigator view*. In the *tree view*, opened documents are displayed in a tree-like manner, with the leaves containing the text or images. By selecting leaves and using the context menu (figure 3.6), they can be concatenated (*Concat Checked Tree Nodes*) and deleted (*Remove One Checked Descendend*). Nodes that were created by concatenating two or more nodes can be separated again (*Show Original Tree*). The *tree view* of a document can be closed with *Delete Tree View*.

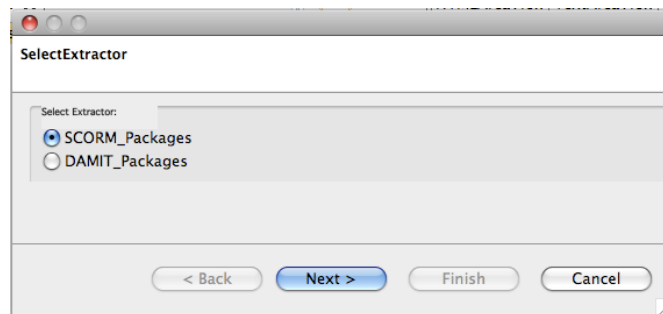
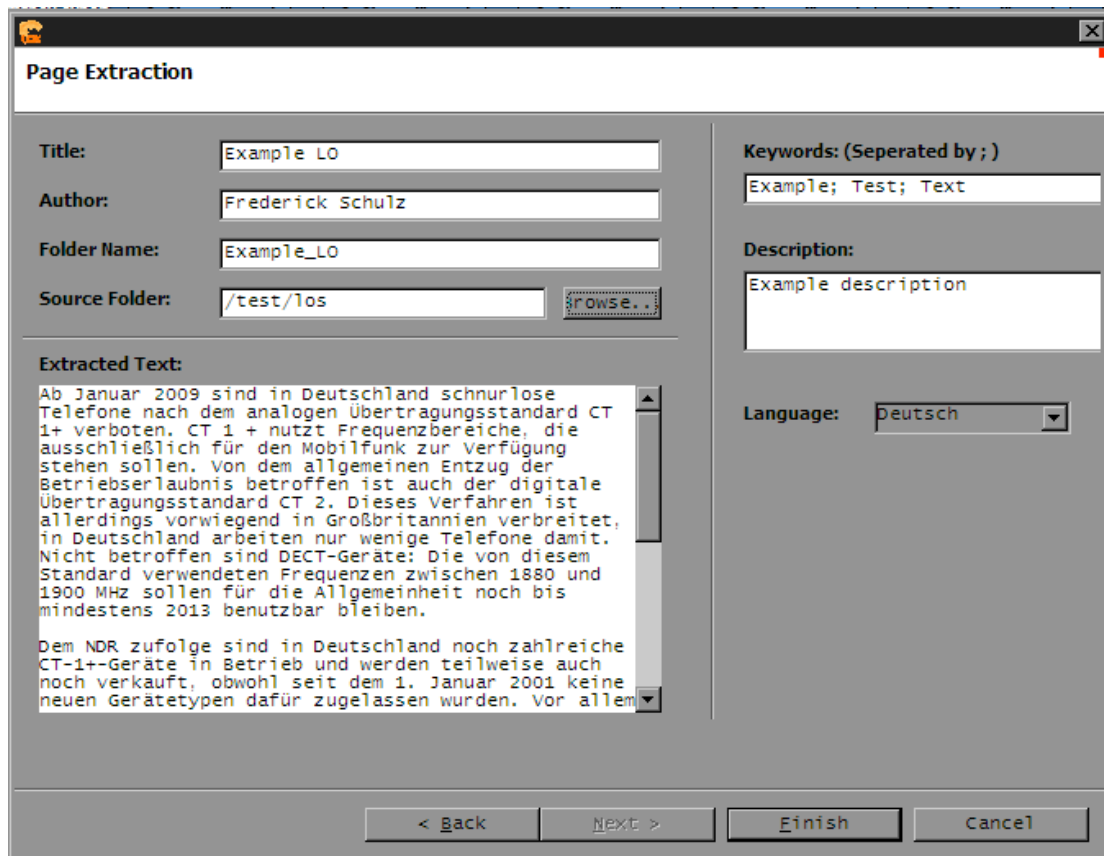


Figure 3.7: Extractor wizard: format selection dialog window

3.5 Step 5: Extracting Learning Objects

When finished with removing useless text and joining paragraphs to well-sized bundles we can start extracting the learning object. A single node can be extracted with the *Extract A Single TreeNode in One File With Parent* command from the context menu. Several nodes can be extracted at once with the *Extract Each Item in One File* command. Next, we see the extractor wizard. In the format selection dialog window (figure 3.7) we can select the desired output format. Let us select SCORM. In the following main window (figure 3.8) some metadata and the location to store the learning object have to be completed. The `los` folder of the current learning object project is the recommended location.

After clicking finish, the learning objects can be found in the chosen subfolder of the folder `los` in the workspace. From there they can be added to any knowledge database accepting the SCORM format.



The screenshot shows a 'Page Extraction' dialog window with the following fields and controls:

- Title:** Example LO
- Author:** Frederick Schulz
- Folder Name:** Example_LO
- Source Folder:** /test/los, with a 'Browse...' button.
- Keywords: (Seperated by ;)**: Example; Test; Text
- Description:** Example description
- Language:** Deutsch (selected in a dropdown menu)
- Extracted Text:** A text area containing two paragraphs of German text about mobile phone standards (CT 1+ and CT 2) and their usage in Germany.
- Navigation buttons:** '< Back', 'NEXT >', 'Finish', and 'Cancel'.

Extracted Text:

Ab Januar 2009 sind in Deutschland schnurlose Telefone nach dem analogen Übertragungsstandard CT 1+ verboten. CT 1+ nutzt Frequenzbereiche, die ausschließlich für den Mobilfunk zur Verfügung stehen sollen. Von dem allgemeinen Entzug der Betriebserlaubnis betroffen ist auch der digitale Übertragungsstandard CT 2. Dieses Verfahren ist allerdings vorwiegend in Großbritannien verbreitet, in Deutschland arbeiten nur wenige Telefone damit. Nicht betroffen sind DECT-Geräte: Die von diesem Standard verwendeten Frequenzen zwischen 1880 und 1900 MHz sollen für die Allgemeinheit noch bis mindestens 2013 benutzbar bleiben.

Dem NDR zufolge sind in Deutschland noch zahlreiche CT-1+-Geräte in Betrieb und werden teilweise auch noch verkauft, obwohl seit dem 1. Januar 2001 keine neuen Gerätetypen dafür zugelassen wurden. Vor allem

Figure 3.8: Extractor wizard: main dialog window

4

Technical Realization

4.1 Technical Background

4.1.1 Eclipse RCP

LOXtractor was developed on top of the Eclipse rich client platform (Eclipse RCP). *Rich clients* are, in contrast to *thin clients* or web applications, mainly located on the user's computer and not on a central server. This allows the native user interface of the platform to be used, using operating system features like drag and drop, system clipboards and UI customization and, important for mobile devices, offline usage (see [ML05]).¹ In contrast to *stand alone* applications, *rich client platforms* provide extensive frameworks and development tools, '[...] eliminat[ing] many of the menial programming tasks required to create UIs and access databases [...] and providing...' frameworks and infrastructure so developers could spend more time programming domain logic rather than reinventing the wheel.²

The Eclipse RCP derived from the Eclipse Java IDE. Being based entirely on Java, it is supported on a wide range of platforms, from all kinds of Unix flavors to Windows and even mobile devices, providing each user with the look and feel he is accustomed to without any adjustments on the developer's side.³

4.1.2 Plugin Architecture

Modularity in the Eclipse RCP is achieved via plugins. Plugins depend on each other, explicitly stating their dependencies in the *plugin manifest*. The Eclipse RCP itself is a set of plug-

1 AJAX and technologies like Google Gears try to deliver these features with web applications, too, but by reimplementing them instead of using the native operating system features.

2 'Foreword by John Weigand' in [ML05]

3 For a more in-depth analysis of potential platforms for **LOXtractor** see [Lud06], p.42ff

ins on top of a Java Runtime Environment⁴, as well as all eclipse base applications, like the Java IDE or **LOXtractor**. Plugins are managed, loaded and executed by the OSGi⁵ framework. The Eclipse Runtime plugin manages applications, special plugins containing the application logic. Together with plugins encapsulating user interface elements from the underlying operating system (Eclipse UI, JFace and SWT) these plugins form the Eclipse RCP.

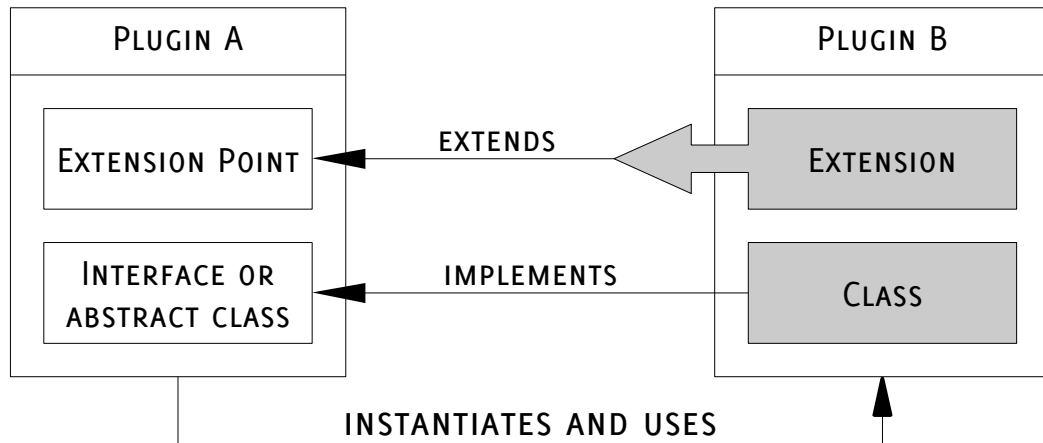


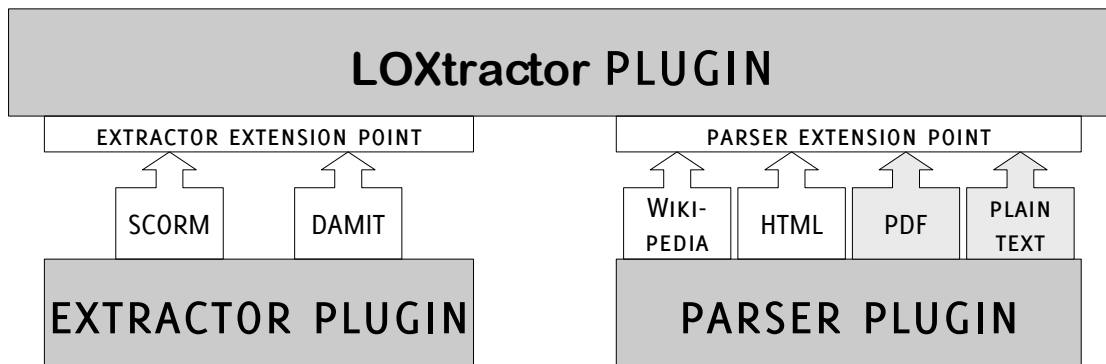
Figure 4.1: Extension and extension point relation

Plugins not only depend on each other by using classes or methods defined in other plugins, they can also provide functionality for each other via extensions and extension points (figure 4.1). Via extensions, for example, an image file viewer could be extended with the logic to display files in a new format. A plugin providing an extension point states this in its manifest file. It declares a unique ID for the expansion point, defines an interface or an abstract class any extension must implement and provides an XML schema, declaring which information any extension must provide. Likewise, the extending plugin has an entry in its plugin manifest that extends that extension point and an XML file corresponding to the provided schema, containing among others the name of the class implementing the extension point interface. A single plugin can have many extension points and can extend several extension points in other modules, it can even extend a single extension point more than once.

The **LOXtractor** application plugin provides one extension point for exporting learning objects with the extractor plugin realising extractors for SCORM and DAMIT by defining two extensions to this extension point. It also defines an extraction point for parsers. Parsers are

⁴ actually only a subset called ‘Foundation Classes’ to reduce the memory footprint on mobile devices, see [ML05]

⁵ *The Open Service Gateway Initiative specifications*, developed and published by the OSGi Alliance specify a framework for defining, composing, and executing components or bundles, called *plugins* in the Eclipse world.

Figure 4.2: Extension points and extensions in **LOXtractor**

classes which import documents into the internal data structure. The parser plugin originally defined two parsers (for wikipedia and for general html pages) and was extended by me with two additional parsers for PDF files and plain text, now providing four extensions to the parser extension point in the **LOXtractor** application class.

4.1.3 The PDF File Format

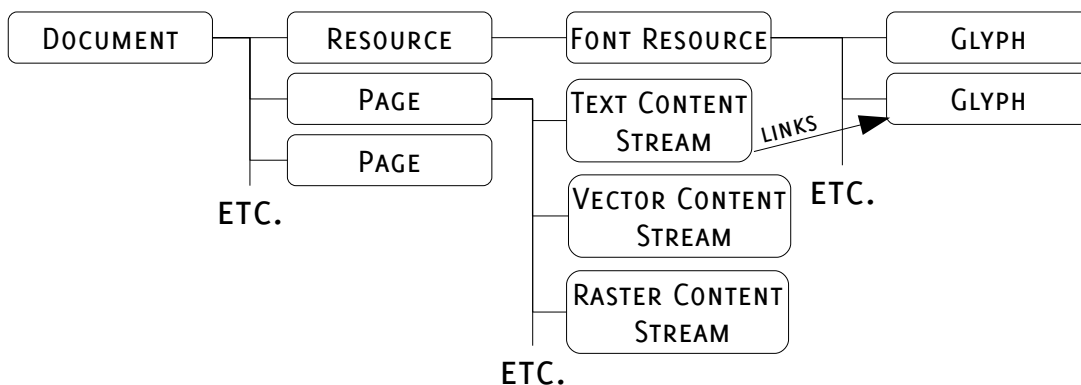


Figure 4.3: PDF file structure

PDF is a page description language. This means the basic building block in every PDF document is the *page* (see [MD02] p.539ff). There's a complex object hierarchy around the page objects. A full description would be impossible and unnecessary in this context, so only the most important aspects will be picked out.

The pages are organised in a b-tree. Each page object contains format information, references to *ressources* and *content streams* (see [MD02] p.544). *Ressources* are fonts, color spaces, patterns, or raster images that are stored in a central section and can be used in several pages.

Content streams define *vector drawings*, *text blocks* or *raster images* and their position on the page (see [pdf06] p.33). They are normally encoded and compressed with various algorithms, unlike the other PDF objects, which are mostly human readable plain texts.

Vector drawings use a *PostScript* compatible description language to place drawing objects (lines, boxes, complex pathes) and fill them with colors, patterns (e. g. from a pattern *ressource*), or gradients. Text blocks do not contain plain text but (compressed) references to parts of a font *ressource*. These parts, called *glyphs* represent single letters or ligatures (see [MD02] p.548f). There are various possibilities how mappings from these glyphs to the letters they represent are stored. This makes text extraction hard or even impossible in some cases⁶ (see [MD02] p.550ff). Tagged PDF files would include a machine readable representation of the text associated with each group of glyphs, but those files are very rare to come across (see [pdf06] p.883ff).

Raster images can be *ressources* or *content streams*. In both cases, a great variety of formats and compression algorithms is supported, including uncompressed, ASCII-encoded bitmaps, JPEG and JPEG2000 compressions, the PNG format and a great variety of TIFF flavors.

Metadata support in PDF files is very elaborate, thanks to the *eXtensible Metadata Platform* XMP. The XMP is ‘a standard format for the creation, processing, and interchange of meta-data, for a wide variety of applications’ [xmp05, p.7]. It provides the ability to store all kind of document metadata in XML format directly as part of the document and supports metadata standards like Dublin Core⁷, Resource Description Framework⁸, Exchangeable Image File Format⁹ or arbitrary user-defined XML namespaces for metadata, all with internationalization – allowing multiple values for different languages – and full unicode support.

4.2 The PDF Parser Plugin

The PDF Parser Plugin is my main contribution to the **LOXtractor** prototype. Functionally it can be divided in three main parts (image extraction, metadata extraction and text extraction) and some auxiliary work. The latter involved some modifications in the extension point schema and the parser selection wizard. The main parts will be described in the following paragraphs.

But first some explanations concerning the way documents are saved internally have to be made. The whole tree representing a document is saved in `TreeObject` objects. Simple `TreeObject` objects form the leaves and contain the text data as plain text and HTML fragments. Images are also represented by `TreeObject` leaves containing a reference to the image location in the workspace or the internet. Pages are represented by `TreeParent` objects, a direct subclass of `TreeObject` with a `Collection` of `TreeNode` references added. These link to the children,

6 Mappings can be missing or purposely wrong to prevent access to the text to anything but a human reader or OCR

7 see <http://dublincore.org>

8 see <http://www.w3.org/RDF>

9 see <http://exif.org>, developed by JEITA, the *Japan Electronics and Information Technology Industries Association*

i.e. all text sections and images on the page. The document as a whole is represented by a `TopParent` (subclass of `TreeParent`) containing the document metadata, a `TreeObject` for all hyperlinks and a `Collection` of the `TreeParent` objects for the pages. For persistence between runs of **LOXtractor**, a `TopParent` object provides methods to store the tree on disk and reconstruct its datastructure into memory.

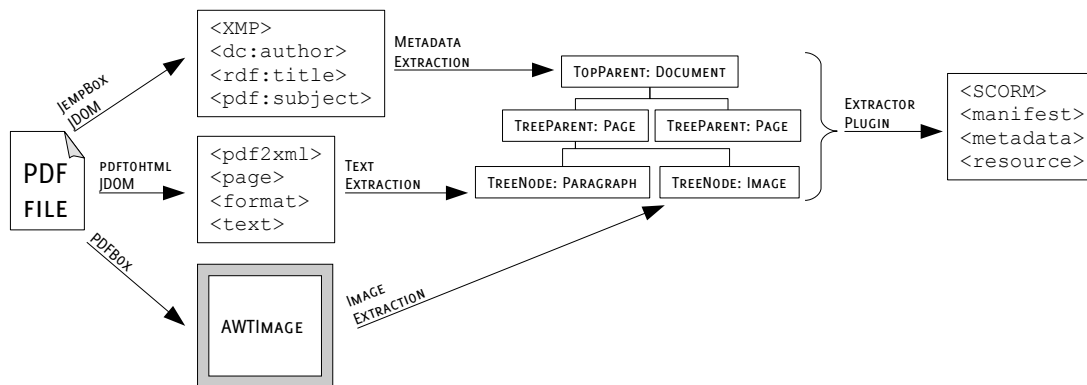


Figure 4.4: From PDF file to SCORM learning object

The PDF parsing process can be divided into 3 parts: image extraction, metadata extraction and text extraction. Each part contributes to the tree representation over a different intermediate state (figure 4.4).

4.2.1 Image Extraction

Image extraction was realized via the PDFBox library. ‘PDFBox is an open source Java PDF library for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents.’ [Litb] The library is published under the BSD license. PDFBox allows the handling of PDF files from an object oriented perspective by imposing an object hierarchy called `pdmodel` on them (see [Lit05] p.1). After opening and decrypting a PDF file with the facilities provided by the PDFBox library a `PDDocument` object is obtained. Its methods allow access to the `PDPage` objects encapsulating pages and their content streams. From there, `Collections` of `AWTImages` can be referenced and written to disk. Then `TreeObjects` referencing them can be created.

4.2.2 Metadata Extraction

After extracting the images, the `PDDocument` object is reused to extract the XML stream containing the metadata. The metadata is stored in this XML document according to the XMP specification (see [xmp05]). To transform the character data into a object oriented representation of the XML document, the library `JDOM` (see [Hun]) is used. From that representation,

the metadata is extracted with the help of the JempBox library (see [Lita]). **LOXtractor** is looking for author, title, subject and keyword information in the default language of the document. Dublin Core, PDF and XMP Basic metadata schemata are requested and the acquired information is merged and written to the `TopParent` object of the document.

4.2.3 Text Extraction

Text extraction was the most difficult feature to implement. While many commercial, free and open source libraries claim to support text extraction, the results were not satisfying. Most libraries simply extract the plain text as a monolithic block, discard all style information or separation into structural elements (e.g. paragraphs) and mix captions, headlines and footnotes in the body text.

Others extract text with the goal to *look like* the PDF page, but sever connections between structurally grouped elements, leading to a fine-grained mess of unrelated text fragments which the user would have to puzzle together. For **LOXtractor** the desired output is some intermediate, coarse-grained structure, conserving text style and structure of the document. After some experiments with different libraries the following workflow was established:

1. The PDF document is converted to XML data with the command line tool *pdftohtml*.
2. The XML data is converted to an XML document object in Java with the JDOM library.
3. The text fragments stored in the XML document are grouped by identical style.
4. The groups of text fragments with the same style are grouped by local proximity into paragraphs.
5. For each paragraph a `TreeObject` is created to store its text and style.
6. The `TreeObjects` are inserted into the document tree as children of the corresponding page `TreeParent`.

pdftohtml is a command line tool originally intended to create HTML documents from PDF files.¹⁰ Important for its use in **LOXtractor** is the feature to create an XML document instead. Figure 4.5 shows a sample from such a file. The program is executed from within the Java runtime and its output to `stdout` is captured and parsed to get a JDOM XML document. It is necessary to provide the environment variable `pdf2htmlcommand` containing the absolute path to the *pdftohtml* executable. (Step 1 & 2)

Now the XML data is in a Java object based form and can be manipulated by the Java program. As seen in the example (figure 4.5), the text is grouped by lines in `text` elements, each line carries location and size properties and each line references to a description of its style

¹⁰ It is available free of charge and supports a great variety of platforms. See [Lin]

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pdf2xml SYSTEM "pdf2xml.dtd">
<pdf2xml>
<page number="1" position="absolute" top="0" left="0"
height="1188" width="918">
  <fontspec id="0" size="24" family="Helvetica" color="#000000"/>
  <fontspec id="1" size="12" family="Times" color="#000080"/>
  <fontspec id="2" size="52" family="Helvetica" color="#000000"/>
  <text top="942" left="257" width="256" height="14" font="0">
    ADOBE SYSTEMS INCORPORATED
  </text>
  <text top="1042" left="257" width="147" height="14" font="1">
    http://www.adobe.com
  </text>
  <text top="137" left="257" width="468" height="50" font="2">
    XMP Specification
  </text>
</page>
</pdf2xml>

```

Figure 4.5: Sample XML file created by *pdftohtml*

properties in a `fontspec` element. The first thing to do is to group the redundant style definitions. The `fontspec` elements are iterated and a `Collection` of `Fontspecgroup` elements is created. The class `Fontspecgroup` is a custom datatype, having a unique name and storing size, font and color information and a list of IDs referencing which `fontspec` elements share this text style. Each `fontspec` element is member of exactly one `Fontspecgroup`. When this assignment is done, the references to `fontspec` elements in the `font` attribute in each `text` element is replaced by a reference to the correct `Fontspecgroup`. Now it is obvious which lines of text share a common text style and might belong to the same paragraph. (Step 3)

To group lines together in paragraphs, some assumptions are made. First, two lines belonging to the same paragraph must have identical text style, indicated by referencing the same `Formatgroup`. Second, they must start at the same left coordinate. The first line of a paragraph may be indented, so a skew to the left is allowed after the first line. Third, they have a regular vertical spacing of no more than line height (with double line spacing, normally much lower). According to these criteria, lines are grouped to paragraphs, represented by `Paragraph` objects. `Paragraph` objects store the text content of the paragraph, and a reference to the `Formatgroup` defining the text style for this part of the text. (Step 4)

In the last step, the `Paragraph` objects are transformed to `TreeNode`s and appended to the `TreeParents` representing the page. From the first words of each paragraph a label for the

`TreeNode` is created to identify paragraphs in the *tree view* easily. With the information from the `Fontspecgroup` referenced, a HTML `<p>` element containing the text in the original style is created and appended to the `TreeNode`.

4.2.4 Solved and Unsolved Problems

While implementing the PDF parser, several problems arose and had to be solved or avoided. The first class of those problems were performance problems. Originally, the W3C DOM implementation included in the Java Runtime Environment was used for the manipulation of XML documents. This caused parsing times of several minutes for documents with single-digit page numbers; this was unacceptable. The use of the JDOM library caused an enormous speedup and made the handling of large documents (several hundreds of pages) possible. On the downside, the set of dependencies was enlarged further. It also led to a refactoring that moved all third-party libraries in a supportive plugin to avoid version clashes between different instances of the same library used in all **LOXtractor** plugins.

Many problems resulted from errors and shortcomings of the *pdfhtml* program. A long-known – but never fixed – bug in this program causes the generated XML code to be malformed. Several search-and-replace operations rectified this, fortunately with little impact on performance. One of these was a `<A>` tag being closed with ``. Fixing the errors in *pdfhtml* was not possible in the given time frame. Another error in this category prevented the preservation of bold and italic text styles. To obtain valid XML, all `` and `<i>` tags have to be removed from the XML text file before passing it to the parser.

An intrinsic problem with the XML approach is the use of reserved XML elements in the text that is to be extracted. The most obvious candidate for failure is the `]]>` character string – the `CDATA` end delimiter. The occurrence of this string in a text section creates invalid XML code. *pdfhtml* should have taken care to encode this sequence properly but fails in doing so.

A more general problem are the PDF-intrinsic rights management and content protection measures. Support for password-protected PDF documents and PDF documents with restricted text extraction is not implemented. Circumventing these protection measures would be critically close to ‘hacking’, so no progress is to be expected in this field. Fortunately, this feature is rarely used.

4.3 The Plain Text Parser Plugin

The plugin realising the import of plain text is called *clipboard Parser*. Its implementation is nearly trivial: A `TopParent` with a single `TreeNode` child is created. The input text is assigned to the `TreeNode` both in the text attribute and a HTML `<p>` element. Despite its simplicity, its use in quickly adding content is indisputable.

Conclusions

5.1 Related Work

5.1.1 PDF content extraction

The field of content extraction from PDF files with its problems outlined in section 4.1.3 received a great variety of both academic and commercial treatment. While most commercial solutions focus on visually exact reproduction of the PDF content and were of no use due to high license fees, some works in the academic sector propose and implement interesting approaches for extraction focussed more on semantics than on visual similarity: With my approach of joining lines that are located closely together and share a common text style, I follow the approach of Tamir Hassan in [Has02]. Hassan describes and implements a program that converts PDF files to HTML documents using a bottom-up grouping algorithm. Starting from single glyphs, words, lines and finally text columns are formed based on proximity and alignment measures. Unfortunately, the library he used (*JPedal*) is no longer available free of charge, so it was not possible to reuse his coding work. A similar approach was described by William S. Lovegrove and David F. Brailsford (see [LB95]) though no actual implementation is available. Hassan continued to work on this topic in the following years (see [HB05]) comparing his algorithm to top-down segmentation algorithms based on visual analysis of pages. Here, *rivers of whitespace* are identified, which are supposed to outline paragraphs. This algorithm has been explained and implemented earlier by Christian Liensberger in [Lie05]. An entirely different approach based on plain text (which is delivered by simple text extraction software) was used by Brent M. Dingle (see [Din04]). Here, based on a dictionary of names and some assumption on the structure of scientific publications, abstract, author name and title are extracted from a plain text representation of the document's first page.

5.1.2 Optical Character Recognition

Closely related is the wide and complex field of optical character recognition with its unmanageable amount of both scientific and commercial research and publication. Giving a comprehensive survey is not possible in this document, so only a few aspects will be considered.

Treating a bitmap representation of the PDF document's pages with layout recognition algorithms used in OCR applications might improve the paragraph clustering results. This is certainly worth looking into for future improvements, since it combines the performance of OCR layout detection with the text correctness, since text extracted directly from the document bears no risk of recognition errors. The open OCR tool OCRopus (see [OCR]) – a project led by DFKI's *Image Understanding and Pattern Recognition* group – naturally comes to mind as a starting point.

5.2 Possible Improvements and Extensions

Still missing for use in a production environment is a backend construction to automate the upload of exported learning objects, now stored on the workplace, to a central knowledge management repository. Currently the resulting SCORM learning objects have to be processed manually.

Contrary to the html parser, the PDF parser does not recognise tables. Table detection in HTML is nearly trivial¹ compared to tables in PDF, which are not marked and often composed of several PDF objects – e. g. separate objects for lines and content. Perhaps the work on table recognition done by Kieninger (see [Kie98]) – applying vertical neighborhood graphs and several proximity and alignment measures on text blocks to find tables and table-like structures – could be applied here to improve the PDF parser.

Another project, Aperture (see [Adu]) – led by Aduna and DFKI – might be promising to provide input to **LOXtractor** from a great variety of sources. Its PDF import however – failing to conserve document structure – was considered too simple for use in **LOXtractor**.

5.3 Conclusion

The prototype of a rapid authoring tool for reusable learning objects, **LOXtractor** was extended with the ability for importing PDF files and for direct input of plain text. The access to the PDF content was facilitated by several third party libraries. The ability to process PDF files was a major step forward to the goal of creating an application that integrates the creation of small-scale learning objects, their annotation with metadata and their mapping to an ontology for later retrieval into the task solving workflow, as intended by the SLEAM process. Especially

¹ Tables are designated by <table> tags

small and medium sized enterprises can profit from this easy and affordable way to conserve individual informal learning effort for the whole company.



Bibliography

- [Adu] ADUNA: *Aperture, a flexible content and metadata extraction framework*. <http://www.aduna-software.com/technologies/aperture/overview.view>
- [Adv] ADVANCED DISTRIBUTED LEARNING INITIATIVE: *Shareable Content Object Reference Model 2004*. <http://www.adlnet.gov/scorm/>
- [DFK] DFKI – DEUTSCHES FORSCHUNGSZENTRUM FÜR KÜNSTLICHE INTELLIGENZ: *Forschungsprojekt DaMiT - Data Mining Tutor*. http://www.dfki.de/web/kompetenz/elearning/projekte/base_view?pid=59
- [Din04] DINGLE, Brent M.: Abstract Extraction from PDF Files / Texas A&M University. 2004. – Forschungsbericht
- [Doh01] DOHMEN, Günther: *Das informelle Lernen*. Bonn : Bundesministerium für Bildung und Forschung, 2001 (BMBF Publik)
- [DTPS07] DÖSINGER, Gisela ; TOCHTERMANN, Klaus ; PUNTSCHART, Ines ; STOCKER, Alexander: Bedarforientierter technologiegestützter Wissenstransfer. In: BREITNER, Michael H. (Hrsg.) ; BRUNS, Beate (Hrsg.) ; LEHNER, Franz (Hrsg.): *Neue Trends im E-Learning*. Heidelberg : Physica-Verlag, 2007
- [GGM⁺05] GELDERMANN, Brigitte ; GÜNTHER, Dorothea ; MOHR, Barbara ; SACK, Claudia ; REGLIN, Thomas ; LOEBE, Herbert (Hrsg.) ; SEVERING, Eckart (Hrsg.): *Leitfaden für die Bildungspraxis*. Bd. 5: *Blended learning für die betriebliche Praxis*. Bielefeld : W. Bertelsmann Verlag, 2005
- [GGWV07] GABRIEL, Roland ; GERSCH, Martin ; WEBER, Peter ; VENGHAUS, Christian: Blended Learning Engineering: der Einfluss von Lernort und Lernmedium auf Lernerfolg und Lernzufriedenheit - Eine evaluationsgestützte Untersuchung. In: BREITNER, Michael H. (Hrsg.) ; BRUNS, Beate (Hrsg.) ; LEHNER, Franz (Hrsg.): *Neue Trends im E-Learning*. Heidelberg : Physica-Verlag, 2007

- [Has02] HASSAN, Tamir: PDF to HTML Conversion / University of Warwick. Coventry, West Midlands, UK, März 2002. – Project Report
- [HB05] HASSAN, Tamir ; BAUMGARTNER, Robert: Intelligent Wrapping from PDF Documents. In: SVÁTEK, Vojtěch (Hrsg.) ; SNÁŠEL, Václav (Hrsg.): *Proceedings of the RAWS 2005 International Workshop on Proceedings of the RAWS 2005 International Workshop on Representation and Analysis of Web Space*. Točná, September 2005, S. 33 – 40
- [Hun] HUNTER, Jason: *JDOM, a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code*. <http://www.jdom.org>
- [imc] IMC INFORMATION MULTIMEDIA COMMUNICATION AG: *Authoring Management Platform EXPLAIN*. <http://www.explain-project.de/>
- [Ins06] INSTITUT FÜR MEDIEN- UND KOMPETENZFÖRDERUNG (MMB): *Corporate Learning 2006*. http://www.sofind.de/vfs/pp/checkpoint_mmb_306.pdf. Version: 2006
- [Kie98] KIENINGER, Thomas G.: Table Structure Recognition Based On Robust Block Segmentation. In: *roceedings of the fifth SPIE Conference on Document Recognition*. San Jose, California, Januar 1998
- [LB95] LOVEGROVE, William S. ; BRAILSFORD, David F.: Document analysis of PDF files: methods, results and implications. In: *Electronic Publishing* 8 (1995), September, Nr. 2 & 3, S. 207 – 220
- [Lie05] LIENSBERGER, Christian: Ideas for Extracting Data from an Unstructured Document / Database and Artificial Intelligence Group, TU Wien. 2005. – Final Report
- [Lin] LINCOLN & COMPANY, A DIVISION OF BISCOM INC.: *PDFTOHTML Conversion Program*. <http://pdftohtml.sourceforge.net/>
- [Lita] LITCHFIELD, Ben: *JempBox, an open source Java library that implements Adobe's XMP specification*. <http://www.Jempbox.org>
- [Litb] LITCHFIELD, Ben: *PDFBox, an open source Java PDF library for working with PDF documents*. <http://www.pdfbox.org>
- [Lit05] LITCHFIELD, Ben: Making PDFs Portable: Integrating PDF and Java Technology. In: *Java Developers Journal* (2005), März. http://java.sys-con.com/read/48543_1.htm

- [Lud06] LUDWAR, Markus: *Methoden zum schnellen Erstellen von wieder benutzbaren Lerninhalten in kleinen und mittleren Unternehmen*, TU Kaiserslautern, Fachbereich Informatik, Diplomarbeit, Dezember 2006
- [May02] MAYR, Philipp: Das Dateiformat PDF im Web – eine statistische Erhebung. In: *nfd – Information, Wissenschaft und Praxis* 8 (2002), S. 475 – 481
- [MD02] *Kapitel 12 – Das Dateiformat PDF*. In: MERZ, Thomas ; DRÜMMER, Olaf: *Die PostScript- & PDF-Bibel*. Heidelberg, 2002, S. 523 – 557
- [ML05] MCAFFER, Jeff ; LEMIEUX, Jean-Michel: *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison Wesley Professional, 2005
- [NT97] NONAKA, Ikujiro ; TAKEUCHI, Hirotaka: *Die Organisation des Wissens: Wie japanische Unternehmen eine brachliegende Ressource nutzbar machen*. Frankfurt am Main : Campus, 1997
- [OCR] OCROPUS PROJECT, THE: *The OCROPUS(tm) open source document analysis and OCR system*. <http://code.google.com/p/ocropus/>
- [pdf06] *PDF Reference, sixth edition: Adobe Portable Document Format version 1.7*. San Jose, California : Adobe Systems Incorporated, 2006
- [RB08] ROSENBLADT, Bernhard von ; BILGER, Frauke: *Weiterbildungsbeteiligung in Deutschland und Europa – Eckdaten zum BSW-AES 2007 / TNS Infratest Sozialforschung*. München, Januar 2008. – Forschungsbericht
- [Rid03] RIDGE GROUP IN CONJUNCTION WITH SOLIS CONSULTING, THE: *Information Gathering in the Electronic Age*. electronic publication on <http://safaribooks.com>, Januar 2003
- [RL07a] ROSTANIN, Oleg ; LUDWAR, Markus: *From Informal Learner to Active Content Provider: SLEAM approach / DFKI, Kaiserslautern*. 2007. – Poster Session of the 2nd European Conference on Technology Enhanced Learning (ECTEL 07)
- [RL07b] ROSTANIN, Oleg ; LUDWAR, Markus: *LOExtractor - Rapid Authoring Tool to Support Workflow-Embedded Authoring*. In: *Proceedings of the 3rd International Workshop on Learner-Oriented Knowledge Management & KM-Oriented E-Learning (LOKMOL 2007)*, 2007
- [Ros08] ROSENBLADT, Bernhard von: *BSW-AES Arbeitspapier Nr. 3: Weiterbildungsbeteiligung in Deutschland und Europa – Konzeptionelle Fragen / TNS Infratest Sozialforschung*. München, Januar 2008. – Forschungsbericht

- [RR02] REINMANN-ROTHMEIER, Gabi: Der Wandel der Bedingungen des Lehrens und Lernens: Wissensmanagement. In: *Grundlagen der Weiterbildung - Praxishilfen* 49 (2002), Oktober, S. 5.380
- [RS06] ROSTANIN, Oleg ; SCHIRRU, Rafael: Identification of User's Learning Goals in Workflows. In: *Proceedings of the Joint International Workshop on Professional Learning, Competence Development and Knowledge Management - LOKMOL and L3NCD held in conjunction with the 1st European Conference on Technology Enhanced Learning*, 2006
- [Sch01] SCHNEIDER, Ursula: *Die sieben Todsünden im Wissensmanagement. Kardinaltugenden für die Wissensökonomie*. Frankfurt am Main : Frankfurter Allgemeine Buch, 2001
- [Tri02] TRIER, Matthias: Der Wandel der Bedingungen des Lehrens und Lernens: Zur Kritik des überkommenen Weiterbildungsprinzips. In: *Grundlagen der Weiterbildung - Praxishilfen* 49 (2002), Oktober, S. 5.370
- [Uni] UNIVERSITY OF BOLTON: *Reusable eLearning Object Authoring & Delivery*. <http://www.reload.ac.uk/>
- [Wika] WIKIMEDIA FOUNDATION, INC.: *Mediawiki, a free software wiki package originally written for Wikipedia*. <http://www.mediawiki.org/wiki/MediaWiki>
- [Wikb] WIKIMEDIA FOUNDATION, INC.: *Wikipedia – The Free Encyclopedia*. <http://www.wikipedia.org/>
- [xmp05] *XMP Specification: Adding Intelligence to Media*. San Jose, CA : Adobe Systems Incorporated, 2005
- [ZS] ZHANG, Wende ; SONG, Yanjuan: *Research on PDF documents information extraction system Based-on XML*. – Bibliothek der Fuzhou Universität, Fujian, Volksrepublik China

List of Figures

1.1	Normal task solving workflow	3
1.2	The SLEAM process	4
1.3	Requirements for elearning strategies in small and middle sized enterprises . . .	5
2.1	LOXtractor window	8
2.2	Self-description of SCORM	9
3.1	<i>Navigator view</i> in the left panel	11
3.2	<i>Tree view</i> in the left panel, contents and metadata of selected node in the right panels	12
3.3	New project window	13
3.4	PDF parser dialog window	14
3.5	Text parser dialog window	15
3.6	Context menu with node manipulation options	15
3.7	Extractor wizard: format selection dialog window	16
3.8	Extractor wizard: main dialog window	17
4.1	Extension and extension point relation	20
4.2	Extension points and extensions in LOXtractor	21
4.3	PDF file structure	21
4.4	From PDF file to SCORM learning object	23
4.5	Sample XML file created by <i>pdftohtml</i>	25