One approach to metadata inclusion in electronic documents

Vyacheslav Bessonov
Computer Science Department
Perm State University
Perm, Russia
v.bessonov@hotmail.com

Viacheslav Lanin
Department of Business Information Technologies
National Research University Higher School of Economics
Perm, Russia
lanin@perm.ru

The article describes an approach to the metadata inclusion into Open XML and ODF documents. This metadata allows implement semantic indexing. The described solution is realized as a software library SemanticLib that provides a uniform access to documents in these formats.

Open XML; OpenDocument Format; metadata; RDF

INTRODUCTION

Semantic indexing of electronic documents is intended to include special structure associated with the content of documents in its metadata. Most of the currently used electronic document formats do not permit the inclusion of additional information. Electronic documents open formats Office Open XML and OpenDocument Format become increasingly popular nowadays. By author's opinion these formats are the most promising.

I. OFFICE OPEN XML FORMAT

Office Open XML (OOXML) is a set of open formats based on ZIP and XML technologies intended for representation of electronic documents package of office applications such as spreadsheets, presentations, text documents.

In 2006 the Office Open XML was recognized as the standard ECMA-376 and 2008 as the international standard ISO/IEC 29500:2008.

Since 2007 version of Microsoft Office OOXML is the default format for all applications included in the package of Microsoft Office.

For each document type its own markup language is used:

• WordprocessingML for text documents;

• SpreadsheetML for spreadsheets;

• PresentationML for presentations.

OOXML also includes a set of specialized markup languages that can be used in documents of various types:

• Office Math Markup Language is used to represent mathematical formulas;

• DrawingML is used to represent vector graphics and diagrams.

Office Open XML uses Open Packaging Convention (OPC), created by Microsoft and intended for storing a combination of XML and binary files (eg, BMP, PNG, AVI and etc.) in a single container file.

II. OPENDOCUMENT FORMAT

OpenDocument Format (ODF) is an open document file format intended for storing and exchanging editable office documents such as spreadsheets, text documents and presentations.

ODF standard is created and supported by Committee ODF Technical Committee organization OASIS (Organization for the Advancement of Structured Information Standards). OASIS published ODF 1.0 in May 2005, Commission International Organization for Standardization / International Electrotechnical Commission ratified it in May 2006 as ISO/IEC 26300:2006, so ODF become the first international standard for office documents.

ODF was accepted as the national standard in the Russian Federation, Brazil, Croatia, Italy, Korea, South Africa, Sweden and Venezuela.

III. APPROACH DIFFERENCES

Although both formats are based on open technologies, and are actually ZIP-archives that contain a set of XML-files defining the contents of the documents, they use very different approaches to solve the same problems and have radically different internal representation.

Format ODF reuses existing open XML standards, and introduces new ones only if it is really necessary. For example, ODF uses a subset of Dublin Core to represent document metadata, MathML to present mathematical expressions, SMIL to present multimedia content of the document, XLink to provide hyperlinks, etc. It means primarily it is easy to use this format by people already familiar with the existing methods to process XML.

The Office Open XML Format uses solutions developed by Microsoft to solve these problems, such as, Office Math Markup Language, DrawingML, etc.

IV. Office Open XML and OpenDocument Format APIs

As mentioned above, despite the same set of used technologies – XML and ZIP, Office Open XML Format and the OpenDocument Format have very different internal representation. Besides over the formats are under permanent development, there are currently several revisions of each format with very different possibilities.

For the Office Open XML they are:

• ECMA-376;

• ISO / IEC 29500:2008 Transitional;

• ISO / EC 29500:2008 Strict.

For the OpenDocument Format they are:

• ISO / IEC 26300;

• OASIS ODF 1.1;

• OASIS ODF 1.2.

Existing software solutions designed to work with this formats are quite different. We will consider some of them.

A. Office Open XML APIs

All libraries and other software tools for working with documents in the Office Open XML Formats can be divided into two broad categories. We will reference these technologies next way:

• OPC API – low-level API, allowing working with OPC-structure of OOXML documents, but not providing opportunities to work with markup languages Office Open XML. Examples of those APIs are shown in Table I.

• OOXML API – high-level API, designed to work with specific markup languages (WordprocessingML, SpreadsheetML, PresentationML). Libraries and tools of this category typically are based on OPC API. Examples of OOXML APIs are shown in Table II.

TABLE I. OPC APIs comparison

	ECMA-376	ISO/IEC 29500:2008	ISO/EC 29500:2008 Strict
Packaging API	+		
System.IO.Packaging	+		
OpenXML4j	+		
libOPC		+	

TABLE II. OOXML APIs comparison

	ECMA-376	ISO/IEC 29500:2008	ISO/EC 29500:2008 Strict
Microsoft Office 2007 Automation		+	
Microsoft Office 2010 Automation			
Open XML SDK 2.0	+		
Apache POI		+	

B. ODF APIs

Libraries for operating with electronic documents in the ODF format can be divided into two broad categories too:

• Libraries in the ODF Toolkit. ODF Toolkit Union is the community of open source software developers. Its goal is simplifying document and document content software management.

• Third-party organizations libraries.

TABLE III. ODF APIs comparison

	ISO/IEC 26300	OASIS ODF 1.2
AODL		
odf4j	+	
ODFDOM	+	+
Simple Java for ODF		
lpOD	+	

V. SemanticLib

It is obvious that there should be a universal approach, allowed to work with electronic documents in various formats in a standardized way. SemanticLib was developed to solve this problem.

SemanticLib - is a program complex designed for semantic indexing of electronic documents. SemanticLib main functions are:

• create new and edit existing Office Open XML and OpenDocument Format documents;

• work with the document metadata, linking the metadata with the content of the document;

• providing an interface for SPARQL queries to the metadata document.

Here are the basic components of SemanticLib:

• *SemanticLib DOM* is an abstract model of an electronic document and its metadata, which may be applicable for the description of electronic documents in various formats (Office Open XML, OpenDocument Format).

• *SemanticLib Plugins* are specific SemanticLib DOM implementations using specialized API. For example, the OpenXmlSdkPlugin uses Open XML SDK 2.0, a plugin OdfDomPlugin uses ODFDOM [6].

• *SemanticLib Interpreter* is a module that allows to work with SemanticLib online.

• *SemanticLib Document Browser* is a GUI application that allows you to analyze the structure of electronic documents, view its metadata and run SPARQL queries.

• *SemanticLib Shell Extension* is Microsoft Windows Explorer extension, which adds to its context menu extra

points, allowing to run SemanticLib Document Browser for certain types of documents (.docx, .odt, etc).

VI. SemanticLib DOM

SemanticLib DOM is a set of interfaces and abstract classes that describe the model of an electronic document and its metadata. This model was designed in accordance with the ISO/IEC 29500 standard, which described in [1], [2], and the OASIS ODF 1.2 specification, which described in [3], [4]. Software implementation of this model is based on the implementations used in Open XML SDL 2.0 and ODFDOM libraries.

SemanticLib DOM as well as the Open XML SDK 2.0 and ODFDOM has layered architecture:

• the first layer contains functions for working with document package (e.g., OPC package as described in [2], or ODF package as described in [4]).

• the second layer contains features designed specifically to work with the structure of the document: add/delete paragraphs, change document content, etc.

A. *Document structure*

The document model has a hierarchical structure and schematically depicted in Fig. 1.

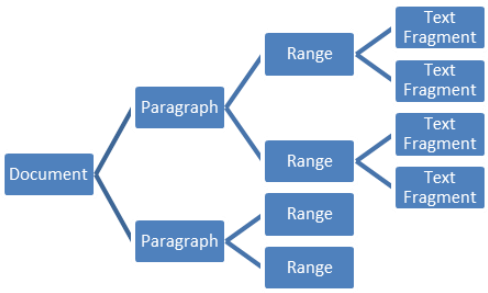

Figure 1. The model of the document used in SemanticLib

Fig. 2 shows a software implementation of DOM SemanticLib.

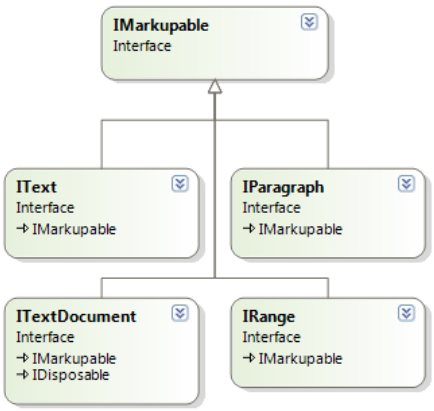

Figure 2. SemanticLib.Core.dll interfaces to work with OOXML and ODF documents

IMarkupable interface contains properties and methods that are used for semantic markup.

ITextDocument interface contains methods and properties for working with text documents, presented in a format like OOXML, and in the format ODF.

IParagraph interface contains properties and methods for working with particular paragraphs of the document.

IRange interface is used for working areas with continuous text contained in paragraphs.

IText interface is designed to work with particular text fragments contained in the text fields. The reason for the separation is the necessary to provide an opportunity for semantic markup of particular words in a text document.

It is worth to note that all mentioned interfaces inherit from interface IMarkupable, so the semantic markup can be used as well as at the level of the document and to its particular elements such as paragraphs, text fields and text fragments.

It was mentioned that a text document and its fragments are containers, i.e. they contain other elements:

• a text document contains a collection of paragraphs;

• each section contains a collection of text fields;

• each text area contains a collection of text fragments.

Fig. 3 shows the hierarchy of abstract classes that represent collections of text documents.

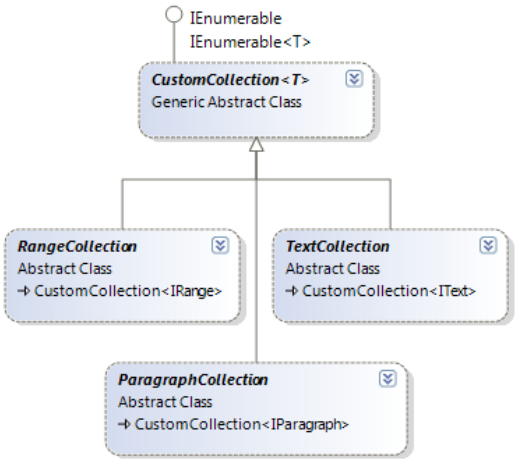

Figure 3. Collections of DOM SemanticLib

CustomCollection is the base class for all collections SemanticLib. It contains the common set of properties and methods, such as, for example, adding a new item in a collection, inserting a new item in a collection, removal element of the collection, etc.

ParagraphCollection represents a collection of paragraphs.

RangeCollection represents a collection of text fields.

TextCollection represents a collection of text fragments.

B. *Metadata model*

Different types of metadata SemanticLib can be divided into two groups:

• non-RDF metadata

• RDF metadata.

1) *Non-RDF metadata*

Metadata model of the group was developed based on analysis:

• core properties, extended properties, custom properties, described in [1], [2];

• predefined non-RDF metadata elements, described in [3].

Table IV shows a list of supported in the current version SemanticLib DOM properties that allow to describe the metadata of the document.

TABLE IV. NON-RDF DOCUMENT METADATA

SemanticLib Property	**OOXML Property**	**ODF Property**
Created	created	creation-date
Creator	creator	initial-creator
Description	description	description
Keywords	keywords	keyword
Language	language	language
LastModifiedBy	lastModifiedBy	creator
LastPrinted	lastPrinted	print-date
Modified	modified	date
Revision	revision	editing-cycles
Subject	subject	subject
Title	title	title
Application	Application	meta:generator
Characters	Characters	meta:character-count
CharactersWithSpaces	CharactersWithSpaces	meta:non-whitespace-character-count
Lines	Lines	meta:row-count
Pages	Pages	meta:page-count
Paragraphs	Paragraphs	meta:paragraph-count
Template	Template	meta:template
TotalEditingTime	TotalTime	meta:editing-duration
Words	Words	meta:word-count

2) *RDF metadata*

SemanticLib RDF metadata model is based on the ODF 1.2 metadata model, as described in [3].

Access to all RDF metadata of the document is performed by the manifest metadata, which in turn is also RDF document. At the program level for this interface IMetadataManfiest is used for this purpose. With IMetadataManifest you can access directly to the manifest's RDF graph, to gain access to existing or to add new RDF metadata files.

SemanticLib uses for work with RDF an Open Source Library dotNetRDF. It provides an opportunity to work with RDF graphs in memory, to serialize/deserialize graphs and to run SPARQL queries.

VII. SEMANTICLIB PLUGINS

The SemanticLib core library contains only a description of the document model (DOM). Implementation of the methods for processing documents of any format is contained in the plug-ins. Typically each plug-in is an implementation of SemanticLib DOM with some libraries described in paragraph V. For example, a plug-in SemanticLib.OpenXmlSdkPlugin.dll uses API Open XML SDK 2.0, a plug-in SemanticLib.LibOpcPlugin.dll contains API libOPC.

Using plug-ins using makes possible a high degree of flexibility and extensibility. If a library expire or a new one appears, developer can just replace or add a plug-in without changing the basic functions of libraries and existing code.

However, plug-in development becomes significant difficult because of the existing the variety and diversity libraries. For example, the library Office Open XML SDK 2.0 is created on the platform .NET, while the library ODFDOM is created in Java, which means a significant difficulty trying to promote interoperability between these libraries. It is also difficult to ensure interoperability between C/C++ and .NET libraries. Let's consider how these issues are resolved in SemanticLib.

A. *C/C++ plug-ins*

Let's see the interoperability between C/C++ and .NET code by the example LibOpcPlugin, which is the implementation of DOM SemanticLib with libraries libOPC, written in ANSI C.

It was decided to use C++/CLI to enable interoperability between managed and unmanaged code. The main advantage of this solution is the ability to use object-oriented programming style even interacting with procedural code of libOPC. In this case plug-in consists of a set of classes that implement the interfaces of DOM SemanticLib.

B. *Java plug-ins*

Interoperability between Java and .NET code will be considered on the example plug-SemanticLib.OdfDomPlugin.

There are some solutions to ensure interaction between Java and .NET applications. For example, there are products of JNBridge company, which provide both in-process and inter-process (network cloud) communication.

However, in SemanticLib Open Source project jni4net was selected. Its aim is providing an in-process communication.

Deal with jni4net has several stages:

• Creating a proxy for a Java library with a special utility proxygen, which is part of jni4net.

• Creating a .NET stub, which provides the work with Java-proxy. This step is also performed using proxygen.

• Implementation of a plugin functional using the resulting stub.

This process is quite complex and requires specific skills, so it is necessary to create automation tools in future versions of SemanticLib.

C. *Working with plug-ins*

One of the significant advantages offered by SemanticLib, is the ability to work with a dynamic plug-ins. This feature is important if you work with a large number of different plug-ins. Plug-in manager, which is part of SemanticLib, helps user to manage the plug-ins loading process. Plugin Manager provides the following features:

• Find the required plug-ins in accordance with certain criteria, such as the name of the plug-in or the format of the document.

• Loading and unloading plug-ins.

• Viewing the meta-information about the loaded plug-ins (name, manufacturer, document format, etc.).

VIII. SEMANTIC LIB INTERPRETER

The main usage scenario SemanticLib involves writing code in one of the support. NET languages and the subsequent compilation of the code. However, this approach is not always convenient. Especially it is not convenient for users who are not programmers and have no special skills to work with IDE, compilers, etc.

SemanticLib Interpreter has been developed to solve these problems. He adds two more SemanticLib use cases:

• interactive work in interpreter mode;

• writing scripts and their subsequent dynamic compilation without the need for third-party tools (IDE, compiler, etc.).

However, it should bear in mind that SemanticLib Interpreter is a DLL and it need a host CUI or GUI application (for example SemanticLib Document Browser).

A. *Interpreter mode*

In this mode user interacts with SemanticLib in progressive interpretation kind. In this case the interpreter correctly handles all the variables, i.e. following line:

```
“var plugin = PluginManager.FindPlugin("SemanticLib.OpenXmlSdkPlugin.dll")"
```

is absolutely correct and the variable “plugin” can be used in subsequent commands.

B. *Scenario mode*

In this mode, the user instead of the progressive interpretation creates a so-called "scenario" (set of SemanticLib commands) and then compiles them into a CUI application.

IX. SEMANTICLIB DOCUMENT BROWSER

This application is still under development. It should be a GUI application written using Microsoft Windows Presentation Foundation framework. The main functions of this applications are:

• view the document structure and it’s metadata;

• SPARQL queries execution and presentation.

X. RESULTS

• SemanticLib DOM library that allows you to perform basic operations with the structure of Office Open XML and OpenDocument Format documents.

• Metadata model and that allows to manipulate document-level metadata.

• SemanticLib.OpenXmlSdkPlugin and SemanticLib.OdfDomPlugin plugins.

• SemanticLib Interpretet that allows you to use SemanticLib functions in interactive mode.

XI. GOALS

• To refine SemanticLib DOM, so that it covered more fully the ISO/IEC 29500 standard and OASIS ODF 1.2 specification.

• To refine the metadata model of the document, in particular the binding metadata to the document content.

• To develop a mechanism for context queries, i.e. queries that would take into account the document contents.

• To develop SemanticLib Document Browser and SemanticLib Shell Extension.

CONCLUSION

Semantic indexing of documents in Open XML Formats and Open Document Format can be implemented on the basis of the described solutions. The developed library is a part of the intelligent document processing project, but also can be used to solve other problems that require metadata inclusion.

REFERENCES

[1] ISO/IEC 29500-1 Second edition, 2011-08-15. Information technology – Document description and processing languages – Office Open XML File Formats. Part 1: Fundamentals and Markup Language Reference.

[2] ISO/IEC 29500-2 Second edition, 2011-08-15. Information technology – Document description and processing languages – Office Open XML File Formats. Part 2: Open Packaging Conventions. 138 c.

[3] Open Document Format for Office Applications (OpenDocument) Version 1.2 Part 1: OpenDocument Schema 29 September 2011.

[4] Open Document Format for Office Applications (OpenDocument) Version 1.2 Part 3: Packages 29 September 2011.

[5] OASIS OpenDocument 1.2 Metadata Examples, Oct 2,2009.

[6] http://incubator.apache.org/odftoolkit/odfdom/index.html

[7] http://dotnetrdf.org/