Reference Open Source Legal CMS

Version: FINAL 6.0
Due date of Deliverable:
Actual submission date:
Start date of Project: 1 January 2006
Duration: 30 months
Project Coordinator: Universiteit van Amsterdam (NL)

Lead contractor deliverable: Corvinus University Budapest
Participating contractors: Alma Mater Studiorum - Universita di Bologna (IT), University of Liverpool (UK), Fraunhofer Gesellschaft zur foerderung der angewandten forschung e.v. (DE), RuleWise b.v. (NL), RuleBurst (EUROPE) Ltd. (UK), knowledgeTools International GmbH (DE), Interaction Design Ltd. (UK), SOGEI - Societa Generale d'Informatica S.P.A. (IT), Ministro per le Riforme e le Innivazioni nella Publica Amministrazione (IT), Hungarian Tax and Financial Control Administration (HU), Budapesti Corvinus Egyetem (HU), Ministero dell'Economia e delle Finanze (IT), Consorzio Pisa Ricerche SCARL (IT)

Deliverable N°: 3.4  
Reference Open Source Legal CMS

Project funded by the European Community under the 6th Framework Programme

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
<th>Public</th>
<th>PP</th>
<th>Restricted to other programme participants (including the Commission Services)</th>
<th>RE</th>
<th>Restricted to a group specified by the consortium (including the Commission Services)</th>
<th>CO</th>
<th>Confidential, only for members of the consortium (including the Commission Services)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PU</td>
<td></td>
<td>PP</td>
<td></td>
<td>RE</td>
<td></td>
<td>CO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

Scope and objective of the deliverable

This document presents the functional specifications of a Content Management System designed inside of ESTRELLA especially for managing legal heterogeneous resources converted in CEN Metalex standard. The CMS called eXistrella aims to use the interchange standards (CEN Metalex, LKIF, LMIF) developed in ESTRELLA as milestone for designing an infrastructure that is able to manage a meta-layer of query related to the documents independent to the metadata model and national standard. This deliverable presents this innovative approach based on new open source technologies.

Organization of the deliverable

The deliverable is organized in two main parts: part I describes the analysis of the system requirements of a Legal CMS, part II describes the Architecture of the eXistrella (Estrella legal CMS).

Part I - System Requirements

Chapter 1 describes the benefits of a Legal Content Management System
Chapter 2 gives an overview of the expected functionality of the pilot application.
Chapter 3 presents different user scenarios of a Legal Content Management System.

Part I - System Architecture

Chapter 4 introduces the logical architecture of the system.
Chapter 5 introduces Daisy components
Chapter 6 describes the integration components between Daisy and eXist
Chapter 7 describes the eXist components
Chapter 8 describes the Interface module
Chapter 9 describes the LKIF queries
Chapter 10 describes the URI resolver module
Chapter 11 describes the Configurations module

Several annexes are enclosed for presenting some preliminary screenshot of the system.
Deliverable 3.4, CMS
Reference Open Source Legal CMS

Monica Palmirani, Luca Cervone
CIRSFID - Alma Mater Studiorum - Universita di Bologna

Caterina Lupo, Fabio Massimi, Pietro Lorenzini
Centro Nazionale per l'Informatica nella Pubblica Amministrazione

Paolo Mascellani
Consorzio Pisa Ricerche SCARL

Krisztian Macsar, Gabor Korosi, Gyorgy Strausz
Corvinus University Budapest
# Contents

INTRODUCTION................................................................................................................................. 6

PART I - SYSTEM REQUIREMENTS....................................................................................................... 7

1. BENEFITS OF A LEGAL CMS ........................................................................................................ 8
   1.1. BENEFITS OF LEGAL CMS ........................................................................................................ 9
       1.1.1. Organization Benefits of legal CMS ......................................................................................... 9
       1.1.2. Technical Benefits of legal CMS ............................................................................................. 10
   SYNTHESIS........................................................................................................................................ 10

2. USER SCENARIOS ........................................................................................................................... 11

3. FUNCTIONAL SPECIFICATIONS ..................................................................................................... 14
   3.1. DOCUMENT REPOSITORY ......................................................................................................... 14
   3.2. VERSIONING AND TEMPORAL MODEL MANAGEMENT ............................................................. 14
   3.3. PRESENTATION AND NAVIGATION OF THE CONTENT ............................................................. 14
   3.4. SEARCH FUNCTIONS ................................................................................................................. 14
   3.5. SEMANTIC WEB QUERY ........................................................................................................... 15
   3.6. URI RESOLVATION ..................................................................................................................... 15
   3.7. ADMINISTRATION FUNCTIONALITIES ....................................................................................... 15

PART II - SYSTEM ARCHITECTURE .................................................................................................... 16

4. LOGICAL ARCHITECTURE ................................................................................................................. 17

5. DAISY COMPONENTS ....................................................................................................................... 18
   5.1. DOCUMENT PACKAGE HANDLING ............................................................................................. 18
   5.2. TRANSACTION MANAGEMENT FOR REPOSITORIES ............................................................... 23
   5.3. QUERY MANAGER ...................................................................................................................... 27

6. EXIST REPOSITORY AND DAISY INTEGRATION ......................................................................... 29
   6.1. REQUESTING AND STORING DOCUMENTS WITH DAISY ......................................................... 29
   6.2. REQUESTING AND STORING DOCUMENTS WITH EXIST ........................................................ 29
   6.3. STORING DOCUMENTS INTO EXIST ....................................................................................... 29
   6.4. REQUESTING DOCUMENTS STORED INTO EXIST ................................................................. 30
   6.5. SUMMARY DIAGRAM OF THE LOGIC COMPONENTS ............................................................ 31

7. EXIST COMPONENTS ....................................................................................................................... 32
   7.1. UPLOADING DOCUMENTS ........................................................................................................ 32
       7.1.1 Packages upload .................................................................................................................... 32
       7.1.2 Performing an upload ......................................................................................................... 32
       7.1.3 Upload summary tables ..................................................................................................... 33
   7.2. SEARCH FUNCTIONS ................................................................................................................. 34
       7.3.1 Queries on documents ......................................................................................................... 34
           7.3.1.1 META queries ............................................................................................................. 34
           7.3.1.2 META queries summary tables ..................................................................................... 34
           7.3.1.3 Textual queries ......................................................................................................... 35
           7.3.1.4 Temporal queries ....................................................................................................... 35

8. INTERFACE MODULE AND EXIST COMMUNICATION .................................................................. 38
   8.1. DISPLAYING A SINGLE DOCUMENT ......................................................................................... 39
   8.2. DISPLAYING THE RELATED SOURCES ..................................................................................... 39
   8.3. DOCUMENTS QUERIES TO EXIST: SUMMARY TABLES ........................................................... 39
9. LKIF RESOURCE MANAGEMENT ......................................................................................... 41
   9.1. STORING LKIF FILES IN THE eXIST DATABASE ......................................................... 41
   9.2. SHOWING THE SEMANTIC RELATIONS OF A DOCUMENT ......................................... 41
   9.3. QUERYING THE LKIF RESOURCES ........................................................................... 41
   9.4. SPECIFICATION OF THE LKIF QUERY ...................................................................... 42
   9.5. LKIF QUERY SUMMARY TABLES ............................................................................... 42
10. RESOLVER AND FRBR URI ................................................................................................. 43
    10.1. THE ARCHITECTURE OF THE RESOLVER ................................................................... 43
    10.2. THE eXISTRELLA URI RESOLVER, WHERE, WHEN AND WHY ............................. 43
    10.3. URI BASED QUERIES SUMMARY TABLES ................................................................. 44
11. TECHNICAL SPECIFICATIONS .......................................................................................... 46
    11.1. ESTRELLA CONVENTION (LMIF, CEN METALEX) VALIDATOR ............................. 46
    11.2. CONFIGURATION MODULE ..................................................................................... 46
    11.3. INTERACTIVITY WITH THE REPOSITORY VIA WEB SERVICES ......................... 48
12. ANNEX A – USE CASES ..................................................................................................... 50
    12.1. ACTORS ..................................................................................................................... 50
    12.1.1. repository manager ................................................................................................. 50
    12.1.2. system administrator .............................................................................................. 50
    12.1.3. end-user ................................................................................................................ 51
    12.2. USE CASES ............................................................................................................. 51
    12.2.1. login ....................................................................................................................... 51
    12.2.2. logout .................................................................................................................. 51
    12.2.3. set preferences ...................................................................................................... 52
    12.2.4. browse repository ................................................................................................. 52
    12.2.5. search repository ................................................................................................. 53
    12.2.6. show legal resource ............................................................................................... 54
    12.2.7. show affected legal resources ............................................................................... 56
    12.2.8. show affecting legal resources ............................................................................ 56
    12.2.9. show version chain ............................................................................................... 56
    12.2.10. show consolidation .............................................................................................. 57
    12.2.11. show filtered versions ......................................................................................... 57
    12.2.12. show repealed legal resources ............................................................................ 58
    12.2.13. show annexes ...................................................................................................... 58
    12.2.14. show meta-data ................................................................................................... 58
    12.2.15. download legal resource ..................................................................................... 59
    12.2.16. upload legal resource ........................................................................................ 59
    12.2.17. remove legal resource ........................................................................................ 60
    12.2.18. install .................................................................................................................. 60
    12.2.19. uninstall .............................................................................................................. 61
    12.2.20. create new user .................................................................................................. 61
    12.2.21. delete user ........................................................................................................... 61
13. APPENDIX B - PRESENTATION OF THE CONTENT ............................................................ 62
    13.1. LOGIN ....................................................................................................................... 62
    13.2. SEARCH ................................................................................................................... 62
    13.3. PREFERENCES ......................................................................................................... 63
Introduction

In this document we specify and present a prototype of a Legal Content Management System, eXistrella, using Open Source components where possible. Main aim of it is showing the feasibility of using different document management regimes in a unique Content Management System using the XML interchange standard of ESTRELLA and with a single interface for manipulating and retrieving heterogeneous documents. This is why we specify the expected functionality in detail to demonstrate its importance and the benefits, however only a part of it will be fully implemented in the prototype.

A central requirement of automated systems based on representations of legal knowledge is an explicit link to the original legal sources the knowledge is based on. Preferably this link should be isomorphic; each argument should be backed with plausible references to legal sources proving the validity of the argument. This is not a trivial task.

Legal sources come in many forms and formats, from different jurisdictions using different legal systems, in different languages, with different internal structures, etc. Legal sources are not self-contained entities but are meaningfully related to each other in one jurisdiction by for instance temporal relations, and relations of priority and relative legislative competence. Each jurisdiction has separate rules for assignment of identity, for versioning, for establishment of validity and applicability etc.

The legal sources themselves are stored in a single Content Management System architecture that is general enough to respect the legal content management requirements of multiple jurisdictions, while allowing manipulation of all legal sources in XML language designed for that purpose. So the architecture of eXistrella is composed by two parts: a CMS based on the most robust principles (Daisy) and from native XML repository (eXist). This architecture guarantees to have robustness and to exploit XML features.

An open-source content management system is chosen to serve as a basis of CMS development. However none of the existing CMS-es are adequate to fulfil all the requirements. Therefore the selected CMS should be extended in various ways to incorporate the semantic XML features, which are needed and described in the functional specifications.
Part I - System Requirements
1. Benefits of a Legal CMS

This section provides a description of the benefits of a CMS for the legal domain enhanced using semantic web technologies. The system will allow the users to manage legal sources enabling the extraction of information and legal knowledge.

Semantic web technologies adopt formal languages to represent document content and metadata thus providing facilities for aggregating information embedded within the text. Therefore these technologies can improve the functionalities provided nowadays by Content Management Systems by providing extensions that make it able to incorporate the semantic features, which are needed for an effective handling of legal knowledge.

One of the tasks within the ESTRELLA PROJECT is to develop a next generation content management which can be defined as a semantic content management. In this system the meaning is expressed by metadata according to a predefined structure and is used in the automated management of the content.

The eXstrella will use the ESTRELLA framework components for legal knowledge management:

− CEN METALEX : XML based representation language for legal sources;
− LKIF-core "Legal Knowledge Interchange Format": format building upon semantic web standards (OWL, RDF,) able to express concepts and rules;
− LMIF "Legal Metadata Interchange Format": format able to map the national standard metadata and LKIF-core concepts, useful for comparing metadata and semantic classification coming from different and heterogeneous XML standards;
− LKIF (s-expression, argument graphs, etc.) able to represent the legal reasoning statements of the law;
− URI: a naming and addressing convention.

The main tasks of the eXistrella are to store, manage and present content using all the power of the XML. Another essential requirement is to expose an interface to other subsystems. Therefore in an integrated system scenario the CMS should be an underlying infrastructure to other system components.

ESTRELLA CMS (eXistrella) will be a content management system enhanced with semantic functionalities compliant with the standard developed within ESTRELLA PROJECT. The system will assist the semantic layer providing capabilities that make system usage easier and much more powerful, adding new and advanced means for creating, sharing and accessing legal knowledge.
1.1. Benefits of legal CMS

The ESTRELLA CMS allows to achieve several relevant benefits for all the actors involved both in handling and retrieving data and knowledge from legal information sources. We can divide the benefit in two categories: organisation/processes benefits and technical benefits.

1.1.1. Organization Benefits of legal CMS

Benefits for all users:

a) Accessibility to the legal resource collection updated (point-in-time) and to the legal knowledge jointly;
b) Implementation of new services based on legal resources and legal knowledge;
c) New market of the legal information.

Benefits for Public administration:

a) Legal drafting improvement, legislation quality, better regulation;
b) eDemocracy - argumentation during the decision law-making;
c) eGovernment application using the legal knowledge and legal resources;

Benefits for vendors tools and editors

a) interoperability between different tools through CEN METALEX and LKIF;
b) enhancing of the logic engine using the expressiveness of LKIF;
c) new market.

Benefits for legal knowledge workers

a) access to material already modelled;
b) instruments for checking and validating the modelling;
c) engine for applying the modelling.

1.1.2. Technical Benefits of legal CMS

Benefits for all users:

a. Storing, keeping and retrieving legal information sources, through uploading, downloading and versioning CEN METALEX-XML documents through web interface;
b. Storing, keeping and retrieving legal knowledge, through uploading, downloading and versioning LKIF-OWL and LKIF-rules documents through web interface.
Benefits for Public administration:
   a. Presenting and navigating legal information sources (CEN METALEX document collection) through a web portal;
   b. Retrieving document information (type of modification, dates) from metadata (LMIF) layer;
   c. Seeing ontological categories (LKIF) connected with the structural parts of the text;
   d. Navigating through the updated text of the legal resources for seeing all the versioning chain of the legislative documents;
   e. Searching documents by: number, date of delivery, country, type of document, full-text;
   f. Semantic searching through the LKIF-core classes.
   g. Comparison of heterogeneous legal sources coming from different countries, languages, standard through the combination of LMIF and LKIF;
   h. Verify the LKIF-rules and argument graph results by navigation of the legal sources.

Benefits for vendors tools and editors
   a. offering an interface to the other subsystems;
   b. providing an underlying infrastructure to the other components in an integrated system;
   c. handling CEN METALEX documents and their versioning through web services;
   d. handling the LKIF (OWL and rules) and LMIF;
   e. interconnecting through hyperlink each external document to CMS internal document.

Benefits for legal knowledge workers
   a. Navigating through the document collection starting from the LKIF-rules;
   b. Navigating the document collection for implementing LKIF-rules;
   c. Retrieving LKIF-rules;

Synthesis

ESTRELLA CMS allows verifying and stressing the links and relations between the ESTRELLA framework components for legal knowledge management: text document (CEN METALEX), ontologies (LKIF relationship, rules, logic and conditions), metadata (LMIF) and identifiers (URI).

The CMS is the application environment implementing all the potentials both of the XML based representation language for legal source and of the legal knowledge interchange format (LKIF).
Within the ESTRELLA project the CMS is the environment where verifying the consistency between a law textual representation in XML and its LKIF model.
ESTRELLA legal CMS provides a prototype useful for the development of more complex legal knowledge system in the future.
2. User Scenarios

As explained in the beginning of this deliverable the end user are mainly four:

A. citizens
B. public administration
C. vendors tools and editors
D. legal knowledge engineers

The CMS component inside of the ESTRELLA project aims to cope with the following application scenario:

A. The public officers will use the CMS for navigating through a CEN METALEX document collection by a web portal. The portal shall present the documents in CEN METALEX enriched with the information coming form the LMIF layer such as the type of modifications, the dates, etc. He/she should pass through a special view to see the LKIF-OWL categories connected with the structural parts of the text. The user shall also to navigate through the updated text of the legal resources with a special “navigation bar” for seeing all the versioning chain of the legislative documents. Moreover the end user shall search the documents by: number, date of delivery, country, type of document, full-text. Semantic searching should be possible: selecting a OWL-category into LKIF-core shall be possible to extract all the document linked to that. (see the http://normaxml.cirsifd.unibo.it).

Fig. 1 Navigation of a CEN METALEX using LMIF information, LKIF-core
Fig. 2 List of modifications.

Fig. 3 – Versioning navigation bar

Fig. 4 – Searching form
Fig. 5 – Searching form through the LKIF-core classes (example of type of modifications), in principle
this tool of search should be used by a OWL graphic representation of the ontology.

B. Interoperability with other applications. First of all editors can upload and
download documents CEN METALEX by web services and the versioning of
them (updated version). Therefore a mechanism of versioning is requested. A
web interface can be able also to upload the LKIF-OWL and the LKIF-rules
and to store them. During the uploading the system should validate XML
document, as well as the LKIF-OWL ontology and also the LMIF.

C. Interconnection through hyperlink/CGI script. Each external document to the
CMS can call a document of the CMS by a Web interface

2004-02-13&number=2&language=en&versionidentifier=2004-02-13 In this case
the vendors can implement the call of the hyperlink in this way for navigating to
the CMS document repository beside the Web-Service interface.

D. The legal knowledge engineer will use the CMS for navigating through the
document collection. In case the CMS can provide also a searching tool interface for
retrieving the LKIF-rules for favouring his/her job.

Fig. 6 List of LKIF-rules connected to the text.
3. Functional specifications

This part presents the functional specifications of the Legal CMS of Estrella: what the system should provide in term of functionalities in order to cope with the goals of the above scenario.

3.1. Document repository

All the documents in the life-cycle are stored in a Document Management System which uses the file system for persistence, consistency and robustness.

3.2. Versioning and temporal model management

The CMS uses the metadata concerning the temporal model based on the interval of enter in force, interval of efficacy, interval of application of the legal document. This aims to produce a temporal model of the Legal System. This characteristic is very peculiar to the legal domain and general purpose Versioning Tools (e.g. software engineering versioning tools) are not adequately equipped to handle this kind of versioning because based on transaction time. The versioning model manages all the chains of versioning, maintains coherence under a legal point of view, checks the semantic legal consistency of each operations done in regard to the whole legal system (e.g. conditional provisions of modification) and the time line.

3.3. Presentation and navigation of the content

The server is in charge of presenting the legal resources showing at least the following view of the documents:

- the normative chain of a cert legal resource (all the text that had modified a cert document),
- the version chain of a cert legal resource (all the version of a cert document),
- the whole legal system fixed in a due time (e.g. I want to know the legal system fixed in the time t),
- the legal system fixed in a cert time y with the perspective of the observer fixed in a cert time x (e.g. annulment event happened in time x),
- the list of all repealed documents or excluded from the Legal System;
- relationship between main document and annexes;
- present document with all the information between CEN METALEX and LMIF;
- present the information connection between CEN METALEX and LKIF,

3.4. Search functions

The server manages the indexing of the textual content, the querying on structured data jointly with full-text and temporal model and the extraction of the relevant fragments from the XML document collection.
3.5. Semantic web query

The semantic qualification made on the legal document is managed by the server side in order to show a specialized view with respect to the semantic relationships defined (e.g. give me all the modifications done by Act(x) in the time y).

The LMIF is the layer defined in the configuration module that permits to define the relationship between metadata of each national standard with the LKIF-core general classes. So it is possible to query all the document with a date of publication equal "19990517" because we have a general class in LKIF-core called

Example for the publication metadata

```
<mcontainer xmlns:rdfa="http://www.w3.org/TR/xhtml-rdfa-primer" name="pubblicazione"
    id="metalex_id178386" rdfa:about="" rdfa:rel="rdf:type" rdfa:href="[lkif-legal_action:Act_of_Publication]" tipo="GU" num="37" norm="19921219">
    <meta id="metalex_id178379" name="pubblicazione-tipo" rdfa:about="#pubblicazione"
    rdfa:property="[lkif-expression:Publication_Type]" rdfa:content="GU"/>
    <meta id="metalex_id178343" name="pubblicazione-num" rdfa:about="#pubblicazione"
    rdfa:property="[lkif-expression:Publication_Number]" rdfa:content="37"/>
    <meta id="metalex_id178333" name="pubblicazione-norm" rdfa:about="#pubblicazione"
    rdfa:property="[lkif-time_modification:Publication_Date]" rdfa:content="19921219"/>
</mcontainer>
```

or for the time of enter into force

```
<mcontainer xmlns:rdfa="http://www.w3.org/TR/xhtml-rdfa-primer" name="entratainvigore"
    id="metalex_id178260" rdfa:about="" rdfa:rel="rdf:type" rdfa:href="[lkif-legal_action:Act_of_Com mencement]" norm="19930101">
    <meta id="metalex_id178255" name="entratainvigore-norm" rdfa:about="#entratainvigore"
    rdfa:property="[lkif-time_modification:Enter_in_Force_Date]" rdfa:content="19930101"/>
</mcontainer>
```

3.6. URI Resolution

This module is able to resolve the URI and translate them in URL able for the navigation. In case of duplication of documents the resolver return the list of the document closer to the query or request. The URI Resolver is able also to tokenize an URI into basic tokens for passing them to a different mechanism of identification belong to external legacy systems.

3.7. Administration functionalities

Manage the server side via web interface. This function can manage all the catalogues, vocabularies, thesaurus and the XML-schema of the system. This part is dedicated to manage the repository of XML schemas (CEN Metalex, LKIF) for permitting the validation of the XML files and for storing the LMIF configurations for checking the semantic validation.
Part II - System architecture
4. Logical Architecture

Architecture of Estrella CMS, eXistrella, follows the Model-view-controller architectural design pattern as you can see on Figure 1. Thanks to this consideration the data model, business logic and user interface is completely isolated and that is the reason why appearance or business logic can be easily modified without affecting the other.

The Model represents the information (documents, knowledge models, meta information, etc), View corresponds to elements of the presentation layer (textboxes, buttons, tables, etc) and the Controller makes possible the manipulation of stored data and manages the communication between user actions and the Model.

The following architecture shows that the system is basically composed by two main parts: Daisy repository and eXist repository with several developing in java for developing the application layer.

Figure 1

In the following chapters you can find the specification of the different modules.
5. Daisy components

5.1. Document package handling

We would like to manage “law packages” beside single documents. These packages are zip files containing all related document of a regulation for instance source format (html, pdf), XML based representation (CEN METALEX, NIR), knowledge model, etc. Name of law packages, their contained folders and files fulfils the requirements of the naming convention introduced in D302 document of WP3 and refined in the CWA2008 of the CEN MetaLex workshop hold in Portrose, 4 June 2008¹.

There are two alternative solutions to handle these packages:

- Uploading and pack/unpack them every time a user would like to access their content. This method has great disadvantages for instance if several users ask different files of law packages we have to unpack them in the same time that requires a very fast server especially in case of big sized packages. The performance of the CMS is highly decreased.

- We are using the structure of Daisy documents.
  The pre-defined document type determines the number of parts inside a document, but the number of files in a law package is not known so we cannot use document parts in this way.

However there is no limit on how many links can be added to a document so if we define a document type (PackageFile) with a name and an attachment part we are able to create a document (PackageFile) for every single file found in the package then we can link these documents to our main document. The links can also be annotated according to the naming convention of the contained file.

¹ http://svn.metalex.eu/svn/MetaLexWS/documentation/2008proposal/
The idea of the second solution can be seen on Figure 2.

In this way the access of stored files needs lower costs increasing the efficiency of our CMS.

![Figure 2](image)

**Figure 2**

In this way the access of stored files needs lower costs increasing the efficiency of our CMS.

![Figure 3](image)

**Figure 3**

<table>
<thead>
<tr>
<th>Actors</th>
<th>User (application module)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The end-user is authenticated and has the appropriate rights</td>
</tr>
<tr>
<td>Post conditions</td>
<td>The end-user is uploaded a selected law package</td>
</tr>
<tr>
<td>Description</td>
<td>The end-user would like to upload a law package through the web based user interface. Packages are parts of the repository documents.</td>
</tr>
<tr>
<td>Actors</td>
<td>User (application module)</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The end-user is authenticated and has the appropriate rights</td>
</tr>
<tr>
<td>Post conditions</td>
<td>The end-user is modified an existing law package</td>
</tr>
<tr>
<td>Description</td>
<td>The end-user would like to modify a package of a repository document. In practice modification means the replacement of the package with a new version of it.</td>
</tr>
</tbody>
</table>

**Figure 4**

<table>
<thead>
<tr>
<th>Actors</th>
<th>User (application module)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The end-user is authenticated and has the appropriate rights</td>
</tr>
<tr>
<td>Post conditions</td>
<td>The end-user has received the selected law package</td>
</tr>
<tr>
<td>Description</td>
<td>The end-user asks for the law package of the selected repository document through the web-based user interface. (for instance to list its content)</td>
</tr>
</tbody>
</table>

**Figure 5**

<table>
<thead>
<tr>
<th>Actors</th>
<th>User (application module)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The end-user is authenticated and has the appropriate rights</td>
</tr>
<tr>
<td>Post conditions</td>
<td>The end-user has received the selected document</td>
</tr>
<tr>
<td>Description</td>
<td>The end-user asks for a document of the selected law package through the web-based user interface</td>
</tr>
</tbody>
</table>

**Figure 6**
You can see the flowchart of package creation and modification on Figure 7.

Document creation and manipulation methods on the repository are operating on a temporary workspace and documents will be stored in the repository after invoking their save method. This behaviour can be exploited during transaction management.

On Figure 8 and Figure 9 you can see the flowchart of getting a law package or its contained documents.
You can see the sequence diagram of a document (package) creation on Figure 10.

The Daisy document creation consists of several steps:

- Creating the document
- Setting the values of required and meta data fields
- Creating PackageFile documents to each file found in the law package
- Adding links to these PackageFiles
- Saving the document (commit)

You can see the sequence diagram of a document (package) modification on Figure 11.

If we would like to create a new version of an existing Daisy document, we can lock the affected variant guaranteeing consistent document versions to others. Different variants can be modified at the same time, because of the scope of lock.
The retrieval of a law package can be seen on Figure 12.

![Figure 12](image)

You can see the retrieval of a single document of a law package on Figure 13.

![Figure 13](image)

### 5.2. Transaction management for repositories

Transaction management is a very important part of every data store system. A single transaction is composed of one or more independent units of work, each reading and/or writing information to the data store. When this happens it is often important to ensure that the data store is left in a consistent state.

In our case it is more important to provide the integrity of data because XML documents are stored both in a native XML database (eXist) and a repository system (Daisy repository). We have to make all document based operations atomic in order to maintain the consistency of stored information.

Both eXist repository and Daisy repository provides mechanisms to lock contained objects so during a transaction no other process can access the affected resources. Besides this kind of mutual exclusion we have to be able to commit or rollback (in case of an exception) all the changes made during the transaction.
The RepositoryAdaptor module has to implement the transaction management beside other functionalities such as distribution of queries between the repositories.

You can see the flowchart of document creation/modification on Figure 14.
We would like to manage “law packages” that are zip files containing all related document of a regulation for instance source format (html, pdf), XML based representation (CEN METALEX, NIR), knowledge model, etc… Name of law packages, their contained folders and files fulfils the requirements of the naming convention introduced in D302 document of WP3 (see also http://svn.metalex.eu/svn/MetaLexWS/documentation/2008proposal/). You can find more information on package handling in a previous chapter.

The transaction manager unpacks the law package then uploads the XML files through Exist Adaptor. I/O API validates them using validator API.

You can see the sequence diagram of a successful document manipulation on Figure 15.

After the transaction manager gets the needed schemas it uploads the XML files into the Exist DBMS. Exist API provides methods to validate contained XML files against schemas.

You can see the sequence diagram of uploading a not valid XML on Figure 16.
In case of an invalid XML file we have to rollback the transaction that means the removal of all uploaded files.

You can see the sequence diagram of a rollback operation because of an arisen exception during Daisy document creation on Figure 17.

In this case we have to rollback all modifications to Daisy repository but it can be solved easily by discarding the affected Daisy document. Beside this it is also required to remove the uploaded XML files from Exist DBMS and from Pellet.
5.3. **Query Manager**

We would like to retrieve the resource contained in this Content Management System by defining queries. However it is not as simple as it seems because we have two different backend systems to search and merge their result set according to the original query.

The whole procedure can be seen on Figure 18.

![Figure 18](Image)

The QueryManager module is responsible for querying Exist DBMS and Daisy Repository. After a successful search the presentation layer receives a collection of QueryResult objects. It is a standardized interface to present the data of different systems.

QueryResult contains all information stored in both back ends for a specific document:

- URI of the document
- Meta information stored in Exist
- Daisy document with all repository specific information

Meta information is represented as a HashMap that is the reason why it can be extended dynamically. Daisy document contains for instance the identifier of the document, modification information (last access, creator, last modifier, etc) and attached files, law packages.
Class definition of QueryResult can be seen on Figure 19.

![QueryResult class definition](image)

**Figure 19**

Presentation layer collects filtering information on a web form that can be:

- Exist specific (date of publication, date of enter in force)
- Daisy specific (full text search phrase, document name)

QueryManager asks Exist DBMS trough IO API and ExistAdaptor. IO API returns URI and metadata of the appropriate documents. ExistAdaptor has to transform this kind of result to QueryResult objects.

At this point QueryManager has a list of document that fulfills the Exist specific search phrases. However this list has to be filtered so Daisy is asked as well. The sequence diagram of a search can be seen on Figure 20.

![Sequence diagram of a search](image)

**Figure 20**
6. eXist repository and Daisy integration

6.1. Requesting and storing documents with DAISY

DAISY stores all documents into the file system. The files are stored in a flat manner without taking care of the hierarchical structure of the legal documents and without knowing the XML structure. In fact a legal document can have several attachments that are in a step down in the whole structure of the legal resource. Moreover a legal document can be written in many versions.

In order to point a document DAISY stores all the ID of the uploaded documents into a MYSQL database. For each document DAISY saves also URI meta information that during the uploading phase is provided by eXist layer after the parser of the XML files. Two types of URI are returned: WORK-URI for the entire document, EXPRESSION-URI for each version.

The cases are three:

1. the URI is included in the XML file and it is compliant to the related metadata in the document;
2. the internal URI doesn’t exist so the metadata of the XML provide the necessary information for building URI;
3. in case the internal URI is not compliant to the related metadata the package is rejected;

DAISY uses an API called ADAPTOR in order to deal with external application and, of course, with itself too.

6.2. Requesting and storing documents with eXist

In order to store documents and request documents from the eXist repository, DAISY must deal with an API called I/O API. This API refers to some other APIs to accomplish the tasks that the adaptor asked for.

6.3. Storing documents into eXist

Even if a valid document can be stored in eXist repository, it must successfully pass through another check. In fact the document must be written in CEN METALEX natively or it must be the result of a translation by another document format that is recognized by eXistrella. To check this pre-requirements, after the above validations the validator APIs deal with another API called “configuration API”. The configuration API deals with another little instance of the eXist database in which are stored the CEN METALEX schema, the LKIF-core files and an LMIF vocabulary file for each supported document formats. Note that the validator also communicates with the configuration API while it is processing the two validations described in the previous section. The validation is twofold: XML validation and semantic validation.
Once all these checks are successfully passed the validator returns an “ok” signal to the I/O api. If the I/O API receives an “ok” signal it can call the eXcommunicator-upload to request the storage of the document into the database. Otherwise, if the document is not valid for some reason, the I/O API will returns to the adaptor a set of information useful to correct the document.

6.4. Requesting documents stored into eXist

In order to perform a request (a query) on the set of documents contained in the eXist database, the IOAPI deals with the eXcommunicator–query API and receive from it a generic query. After this step the IO/API deals with the validator that, analysing the appropriate LMIF vocabularies creates a valid and working XQUERY.

The extracted document (or URIs list) is returned to the I/O API that returns it to the validator. If no documents are compliant to the request the whole process returns a set of information to the adaptor that can use this information in order to perform an improved request.
6.5. Summary diagram of the logic components

The diagram below represents the whole process.
7. eXist components

7.1. Uploading documents

Of course the eXistrella CMS must be populated with a large set of legal resources having their XML format and their attachments that can be either in XML format or in other multimedia format. The CMS will give the possibility to upload documents packages and to retrieve or read documents performing ad-hoc queries on the repository.

7.1.1 Packages upload

As we say in the other sections of this documents, for the eXistrella CMS the concept of “document” must switch to the concept of “document package”, this because the documents consists of several parts, some of them stored in the eXist repository and some other stored in the DAISY repository. This means that the upload of the package is not simply a “save to disk” operation but is a more complex task that needs to pass through a "Transaction manager" layer managed by Daisy business logic module.

7.1.2 Performing an upload

The uploading operation is performed as following.

1. The end-user uploads the document package as zip file through interface connected with Daisy.
2. The Daisy layer stores in a temporary memory the document-package.
3. Daisy unzips the package and passes only the XML files, including the LKIF files, to eXist-API.
4. The eXist-API distinguishes the CEN MetaLex documents by the LKIF files and stores them in two different eXist repositories.
5. The eXist-API validates the two kinds of XML documents respect the relative XML-schema.
6. The only CEN MetaLex files are validated also respect the minimal metadata set required by the CMS and declared in LMIF.
7. In case of errors the eXist-API returns an appropriate error to the Daisy "transaction manager" and it deletes the documents from its repository (rollback).
8. In case of positive validation the eXsit extracts the URI from the main document of the package and passes it to Daisy "transaction manager". There are three cases respect the URI:
   a. The URI is included in the document as LKIF declaration.
   b. The URI is not included in the document but there are all the metadata requested for building it (country, type-doc, date-delivery, number, language, version). The URI is the URI of the expression because it represents the version in the Daisy document model.
   c. The URI is not included and there aren't metadata for inferring it. In this case the document and all the package is rejected by the eXist and consequently from CMS through the "transaction manager".

9. Daisy manages the transaction through the "transaction manager" module and stores in the Daisy repository the document, in MySQL the metadata. One of fundamental MySQL's metadata is URI.

### 7.1.3 Upload summary tables

#### PackageUpload

<table>
<thead>
<tr>
<th>Description</th>
<th>Upload in the eXistrella the package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>all XML files belong to the ZIP file uploaded by the end-user</td>
</tr>
<tr>
<td>Output</td>
<td>• Success validation for all the package - string or boolean</td>
</tr>
<tr>
<td></td>
<td>• URI metadata for all the XML files (MetaLex/CEN XML)</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requisites</td>
<td>The end-user upload a ZIP file with the URI included in the XML as statement using RDF/A syntax</td>
</tr>
</tbody>
</table>

#### ConfUpload

<table>
<thead>
<tr>
<th>Description</th>
<th>Configuration uploading. This method uploads the configuration files for the good management of the validation of the XML files.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• XML-schema or RDF or OWL files (MetaLex/CEN, LMIF, LKIF-core)</td>
</tr>
<tr>
<td>Output</td>
<td>Success validation and upload in eXist - string or boolean</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requisites</td>
<td>Inside of each configuration file there is the source or the name of the local standard which it belong to.</td>
</tr>
</tbody>
</table>
7.2 Search functions

7.3.1 Queries on documents

Users want to download or read the documents hosted by eXistrella or their attachments. In both cases documents are retrieved using very complex XQUERYs due to the fact that the queries must be performed on a set of documents that comes from different data formats.

7.3.1.1 META queries

In order to perform query on document coming from different standards, eXistrella implements a meta-query mechanism.

In this manner the queries are independent from the original metadata used in the national standard, but it is based on the patterns defined by CEN METALEX and also based on the RDFa assertions that the document has. This mechanism permits to use all the expressiveness of the original national standard but in the meantime to use the meta queries based on the LMIF layer that assignees for each national metadata a general class in LKIF ontology.

7.3.1.2 META queries summary tables

DocQuery

<table>
<thead>
<tr>
<th>Description</th>
<th>Any query respect the minimal data set defined:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- number</td>
</tr>
<tr>
<td></td>
<td>- country</td>
</tr>
<tr>
<td></td>
<td>- type of document</td>
</tr>
<tr>
<td></td>
<td>- title</td>
</tr>
<tr>
<td></td>
<td>- article/section n.</td>
</tr>
<tr>
<td></td>
<td>- date of delivery</td>
</tr>
<tr>
<td></td>
<td>- date of enter into force</td>
</tr>
<tr>
<td></td>
<td>- date of publication</td>
</tr>
<tr>
<td></td>
<td>- full-text</td>
</tr>
<tr>
<td></td>
<td>- n. of publication</td>
</tr>
<tr>
<td></td>
<td>- language</td>
</tr>
<tr>
<td></td>
<td>- URI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Method of I/O API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Array of URI - stringArray</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement</td>
</tr>
</tbody>
</table>
ActiveRefQuery

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the list of the URI (document) cited inside of the document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>URI of the document</td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement</td>
</tr>
</tbody>
</table>

PassiveRefQuery

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the list of the URI (document) that cites the current document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>URI of the document</td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement</td>
</tr>
</tbody>
</table>

7.3.1.3 Textual queries

The textual query is possible specifying not only the full text pattern to match, but also specifying the part of the legal document where to search. So it is possible to search "privacy" only in the title or in the preamble or in the body of the legal text exploiting simply the XML tagging.

TextQuery

<table>
<thead>
<tr>
<th>Input</th>
<th>Method of I/O API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Array of &lt;Fragment of XMLtext&gt; - Array of XML</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement</td>
</tr>
</tbody>
</table>

7.3.1.4 Temporal queries

The temporal queries are based on the fact that the legal resources follow a dynamic behaviour on the basis of modifications, changes, amendments. In other words events affect the legal resources over the time and consequently also its LKIF representation. For this reason to manage the versioning during the lifecycle of the document is very important feature for the legal domain. The temporal queries are independent to the physical organisation of the document (multiversioning, monoversioning, several files for each document, etc.), the important element are the RDFa assertion done inside of the document itself.
We can distinguish five main temporal queries:

(A) access to the static time like enter in force. This query asks all the documents enter in force in a due date. This query provides several outputs because in the same time multiple documents can enter in force for the first time. This time is a static time because it is like the "birth" date of the document respect the legal system;

(B) access to the dynamic time like version of the document in force in a due time. This query provides all the documents "valid", under the legal point of view, in a due time. This mechanism is called "point-in-time" query;

(C) access to the dynamic time of the fragment of the document. In this case the query aims to detect all the fragments that are in force in a due time, like a new article included in the last version. This query can involve the enter in force time or the effective time (when a article is effective and applicable by the judge or by the citizen);

(D) all the modifications done in a due date. This query permits to extract all the passive references or the active references on the base of the action of modification interested fixed in a due time. E.g. give me all the abrogation done on the document X in the time t. Or give me all the substitution provisions in force in a time k;

(E) access to different simultaneous versions and disambiguate them on the base of the view point of the observer. This is the case of two o many versions in force in the same time due to retroactive events that artificially modifies in the past the document content. Because these kind of modifications are called ex-tunc, we have the paradox of multiple versions in the same instant t both valid under the legal point of view (e.g. in the criminal law judge must decide on the base of the best choice). So only the point of view of the observer can disambiguate this query and a new parameter (date of the observation) is needed in input.

**InForceDoc**

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the document entered in force in a due date. The enter in force date is the first event of the document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Date of query</td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement about the enter in force</td>
</tr>
</tbody>
</table>

**VersionActiveMod**

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the list of fragments (provision of modification) that acts a particular type of modification in a due time t.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Type of modification</td>
</tr>
<tr>
<td>Date of query</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF and RDFa statement</td>
</tr>
<tr>
<td><strong>VersionPassiveMod</strong></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Provide the list of fragments (provision of modification) affected by a kind of modification in a due time.</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Type of modification</td>
</tr>
<tr>
<td></td>
<td>Date of query</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Array of URI</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>List of errors</td>
</tr>
<tr>
<td><strong>Pre-requirements</strong></td>
<td>LMIF and RDFa statement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>VersionDoc</strong></th>
</tr>
</thead>
</table>
| **Description** | Provide the document version in force in a due date  
Note: the same query with the current date is used for obtaining the last updated version of the document |
| **Input**       | Date of query  
Date of observer (optional) |
| **Output**      | Array of URI  |
| **Errors**      | List of errors |
| **Pre-requirements** | LMIF and RDFa statement |

<table>
<thead>
<tr>
<th><strong>VersionFragment</strong></th>
</tr>
</thead>
</table>
| **Description**     | Provide the fragment of the document in force in a due date.  
Note: the same query with the current date is used for obtaining the last updated version of the document |
| **Input**           | Date of query  
Type of date (efficacy or forcing)  
Partition specification  
Number of the partition |
| **Output**          | Array of URI  |
| **Errors**          | List of errors  |
| **Pre-requirements** | LMIF and RDFa statement about the efficacy and the forcing for each fragment modified. |
8. Interface module and eXist communication

Using the eXistrella CMS the user can read not only the legal resource of a document but also several other information regarding the documents, like semantic relations with other documents, time and date information, lifecycle of the document and finally transaction information respect the database (author of the submitter, date of submission, date of the last modification).

In order to give a great experience to the users, the interface of the eXistrella CMS must be designed and developed using the modern web technologies.

AJAX must be massive used in order to permit the visualization of the related info of a document without accessing to another section of the CMS, but simply passing over (i.e. intercepting the on-mouse-over event) the document title (in a document list) to list the related documents, or passing over a particular section of the legal resource (when the user is reading a single legal resource) to list the documents that the section affects or the documents that are affected by the section.

EXistrella CMS is composed by an eXist database, a DAISY CMS, and an intermediate layer that permits to the DAISY CMS to communicate with the eXist database.

The interface of eXistrella must be plugged to the DAISY CMS and not to the eXist database directly. That because users do not upload only XML files of the legal resource, but it also need to manage files of other nature, such like PDF version of the document or audio related to sentences or to debate record.

All the files that are not XML files are uploaded into DAISY repository, so the interface must deal with DAISY, and DAISY must deal with the eXist database in order to return to the interface all the XML documents but also all the multimedia files related to a legal resource.

The eXistrella CMS must also provide a universal XSLT style sheet in order to properly display the legal sources in the METALEX format. This is not a trivial task due to the fact that the documents are translated by different data format so they can have structures that differ in several points.

Note that, for our research purposes, not all of the above features in term of query or interface will be developed.
8.1. Displaying a single document

As we say in the previous paragraph, the purpose to show METALEX legal sources translated by different legal format is not a trivial purpose. Nevertheless the CEN METALEX standard, enriched with RDFa assertions, helps to favor this task.

Each document stored in eXistrella will be linked to an XSLT that translates the structures of the documents in XHTML structures in order to make them displayable by a web browser independently to the XSLT engine.

The XSLT translates all the hierarchical sections of the documents into nested XHTML lists, all the containers into generic divs, block elements into paragraphs and inline elements into generic span. The XSLT also assigns an ID or a CLASS to each created elements and subsequently a CSS containing the style properties for each element is linked to the produced XHTML document.

8.2. Displaying the related sources

When a user is reading a list of legal resources (i.e returned after a search in the eXistrella repository) he/she must be able to instantly view the other resources related to them. To do this he can pass with the mouse on the title of an item of the document and the related sources will appear on the right side of the screen in a special part of the window. So he/she can choose directly to read the displayed sources clicking on them. In this information are included also Daisy information such as name of the author, date of creation, date of modification. For this purpose the query module is managed by the "Presentation layer" of Daisy in order to join together all the information (coming from Daisy repository and from eXist) in a unique presentation for the user.

8.3. Documents Queries to eXist: summary tables

DocInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>Return the minimal docinfo concerning a list of URI passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Array of URI [work-URI or expression-URI]</td>
</tr>
<tr>
<td>Output</td>
<td>Array of eXistDocInfo</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>correct and existing URIs</td>
</tr>
</tbody>
</table>
### DocValidation

<table>
<thead>
<tr>
<th>Description</th>
<th>Return the status of the XML document: valid or not valid respect the XML-schema declared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• XML file</td>
</tr>
<tr>
<td>Output</td>
<td>Success validation - string</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>XML declaration of the XML-schema inside of the file</td>
</tr>
</tbody>
</table>

### SemanticDocValidation

<table>
<thead>
<tr>
<th>Description</th>
<th>Return the status of the XML document: valid or not valid respect the minimal data set of metadata included in the XML file and requested as mandatory by the CMS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• XML file</td>
</tr>
<tr>
<td>Output</td>
<td>Success validation - string</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>LMIF declaration in the configuration file</td>
</tr>
</tbody>
</table>

### ConfigRequest

<table>
<thead>
<tr>
<th>Input</th>
<th>Method of I/O API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Configuration files - byteArray</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td></td>
</tr>
</tbody>
</table>
9. LKIF Resource management

9.1. Storing LKIF files in the eXist database

One of the main goals of the CMS is to show the semantic relationships connected with the collections of documents stored in the legal CMS.

This means that each document (or each document package) is potentially connected with a set of LKIF files that models the rules, the norms, the concepts and the arguments included in the document.

The LKIF files concerned to a document are potentially many and vice versa a document contains many references to several LKIF files. So the relationship between LKIF files and document is many to many (M:N). The LKIF file are written in an XML derived language. This means that the search of the relations concerning a document can be a very complex task that, usually, afflicts a very large set of XML documents.

The best way to query the LKIF resources related to a document is to use XPATH or XQUERY languages.

So, if we store the LKIF files (XML files) in a particular instance of the eXist database of the eXistrella CMS, we can use the above technology in order to perform the complex queries in a very customizable way and with a high performance results.

9.2. Showing the semantic relations of a document

The semantic relations of a document must be displayed when a user is reading a document. But also a user needs to choose a document related to other documents.

This means that the semantic information related a document (described in the LKIF resources) must be displayed on a side of the document (when the user is reading the document) or in a popup-like window (when the user pass the mouse over the title of a document while he is reading a list of document).

This can be done using the most recent web technologies (AJAX).

9.3. Querying the LKIF resources

The libraries of the eXistrella CMS must supply a web service whose task is to return all the relationships of a particular document querying the eXist database.

The relationships are many to many, so it is not so trivial job to extract from all the LKIF files which are involved in the query. Any LKIF file have a representation of the normative content of the rules and in the top section a part called <sources> where are listed all the URI of the CEN Metalex sources involved in.
9.4. Specification of the LKIF query

**Input parameter:** URI of the XML document  
**Output parameter:** component object containing a set of LKIF documents

**Method:**  
The query will be passed by the "presentation logic" module of Daisy to eXist with the URI of the document interested of the query.  
`eXistestrella-excommunicatorquery` selects all the LKIF documents where the `<source>` block contains the URI of the document interested in.  
**Presentation:** three modalities of presentations are possible:  
   a) on the list of the documents Ajax component shows the LKIF pertinent with the document `on_mouse_over`;  
   b) as a particular view of the document;  
   c) in a graphic way.

9.5. LKIF query summary tables

**LKIFResources**

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the list of the LKIF document concerned to the current document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>URI of the document</td>
</tr>
<tr>
<td>Output</td>
<td>Array of XML</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>none</td>
</tr>
</tbody>
</table>
10. Resolver and FRBR URI

FRBR is a standard that have two primary objectives. One of this is very important for our project: to provide a clearly defined, structured framework for relating the data that are recorded into a CMS or stored in some digital way.

So, for legal resources we can have Works URIs, Expression URIs and Manifestation URIs (see the ESTRELLA naming convention).

The resolver, developed by Fabio Vitali and Luca Cervone of the University of Bologna, is a configuration file for apache server. It takes in input a well formed FRBR URI and resolves it producing a URL that points to a digital document stored somewhere.

In the eXistrella CMS the produced URL will be handled both by the DAISY layer and by the eXist layer. This not means that the resolver dials directly with the eXist, but this means that the DAISY layer can simply pass the produced URL to the eXist database layer in order to retrieve or delete a Work or a Specific Expression of a legal resource.

10.1. The architecture of the resolver

As we previously said, the resolver is an APACHE configuration files. It is based on the APACHE mod_rewrite module.

After a request comes in (i.e. an user write an URL in the address field of a browser and clicks “GO” or a user clicks a link) and APACHE as determined the corresponding server, the mod_rewrite starts to process all the rules defined in the URI resolver. At the end of this process a new URL is produced, and it can be a URL that directly points to a resource in the server or an URL that contains a query string so that actually points to a script giving to it some parameters.

This means that each one of these operations is done BEFORE the content of a requested resource is actually retrieved and processed and BEFORE any software hosted by the APACHE server starts to run (i.e. TOMCAT or DAISY).

10.2. The eXistrella URI resolver, where, when and why

The eXistrella URIs are formed by several token, delimited by the “/” character, that actually represent to some metadata belonging to a Work or an Expression. They are formed by the country of the work/expression, by its type, by its number, by its language and so on.

The main task of the resolver is to detect the tokens contained in the URI and to add them to a parameterised query string that then is managed by the DAISY layer of the eXistrella CMS in order to retrieve all the resources related to the requested work or expression.
This is done every time a resource is requested, in order to translate the requested URI into a query string that DAISY can understand and process.

Subsequently DAISY sends the extracted metadata (again rewritten in the original URI format) to the eXist layer that with this information can retrieve (or can delete) the XML related sources stored in the database.

### 10.3. URI based queries summary tables

#### Work Request

<table>
<thead>
<tr>
<th>Description</th>
<th>Query of a documents by work-URI. The work-URI is the prefix of the extended URI. It is equal for each versioned package (e.g. /hu/act/2007-01-12/34/ that means the law n. 24 of 12 January 2007, in the Hungarian Law). See the D3.2 for more information and the new MetaLex/CEN documentation on the web site of CEN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• URI of Work</td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI of expressions - stringArray</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>correct and existing URI</td>
</tr>
</tbody>
</table>

#### Expression Request

<table>
<thead>
<tr>
<th>Description</th>
<th>Query of the documents by expression-URI. The expression-URI is the prefix of the extended URI. It is equal for one versioned package (e.g. /hu/act/2007-01-12/34/en@2008-04-20 that means the law n. 24 of 12 January 2007, in the Hungarian Law, in English, version updated at 2008-04-20). See the D3.2 for more information and the new MetaLex/CEN documentation on the web site of CEN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• URI of expression (URI of work is the URI of all the Daisy document, URI of expression is the URI of the Daisy-version)</td>
</tr>
<tr>
<td>Output</td>
<td>DocXML - XML</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>correct and existing URI</td>
</tr>
</tbody>
</table>
## Delete Work

<table>
<thead>
<tr>
<th>Description</th>
<th>Delete a package with a work-URI specified by the end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• URI of work</td>
</tr>
<tr>
<td>Output</td>
<td>Success of cancel in eXist - string or boolean</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>correct and existing URI</td>
</tr>
</tbody>
</table>

## Delete Expression

<table>
<thead>
<tr>
<th>Description</th>
<th>Delete a particular expression package with a expression-URI specified by the end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• URI expression</td>
</tr>
<tr>
<td>Output</td>
<td>Success of cancel in eXist - string or boolean</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td>correct and existing URI</td>
</tr>
</tbody>
</table>

## DocBrowse

<table>
<thead>
<tr>
<th>Description</th>
<th>Any query using also the URI work or expression. NOTES: The URI is resolved and DAISY can use the URI itself to ask to eXist the corresponding document and the information extracted from the URI (by the resolver) to query its repository and show the related resources.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Method of I/O API</td>
</tr>
<tr>
<td>Output</td>
<td>Array of URI - stringArray</td>
</tr>
<tr>
<td>Errors</td>
<td>List of errors</td>
</tr>
<tr>
<td>Pre-requirements</td>
<td></td>
</tr>
</tbody>
</table>
11. Technical specifications

11.1. Estrella convention (LMIF, CEN METALEX) validator

Documents are sent to the IO/API then it sends the documents to the validator and some other files to permit the validator to check if the document is valid. To be valid the document must:

d) Be compliant with URI naming convention
e) Be compliant to METALEX standard
f) Carries with it a LKIF-core.owl (in case) file
g) Have RDFa assertion regarding some elements

These three validations are made sequentially. A document can't be stored if all the validations are made successfully.

The validation API perform also another task, in fact, when a query is requested to the IO/API it is sent to the validation API that deals with the configuration API in order to replace the requested generic query with a valid XQUERY.

![Diagram of the configuration module]

Figure 23

11.2. Configuration module

The aim of the configuration module is to store and manage common files, values beside initialization tasks. These files can be XML schemas, XSL transformation files, OWL files or simple text values.
The basic implementation of this module is a serializable hash map following the singleton design pattern. It contains keys and values where key is an identifier of the resource and the value is a text or a file reference. This hash map is stored in an XML file and the application reads it at start up. There can be defined required values where users cannot change the key, and user defined keys. CMS modules ask for resources by their identifiers (keys).

This module can be configured on the administration web site.

There are two alternative ways to implement this:

c. The first method is to store these common files on the file system of the application server. This solution has some disadvantages, for instance in case of multiple application servers we have to synchronize these files. In addition of this authorization and security problems could occur as well.

d. Another option is to place these files and the configuration xml in the Exist DBMS. In case of multiple repository instances, their synchronization is not a problem and there is absolutely no problem using multiple application servers and one repository instance.

At start up the configuration file must be received from the Exist DBMS in order to create its representing data structure in the memory. If this data structure is changed synchronization is required with the configuration file stored in the repository as you can see on Figure 24.

![Figure 24](image-url)
Any application module can interact with this module and use its predefined content in a generalized way as you can see on Figure 25.

The first version of the interface of this module can be seen on Figure 26.

11.3. Interactivity with the repository via web services

The aim of a web service interface is to provide a common interface that can be used by any vendor specific application exploiting the functions offered by this Content Management System.

The web service of this pilot application has the following methods for basic operations:

```
long Login(String User, String Password)
```

The application has to authenticate itself with a username and a password. This account can be used as well on the web based user interface. If the authentication was successful, a unique identifier is returned that must be used in case of every operation through the web service interface.
void storeDocument(long Identity, Document uploadDocument)

This method is for creating new documents or save the modifications of an existing one. It has a parameter that is a locally created/manipulated document beside the unique identifier.

Document retireDocument(long Identity, String documentId)

We can remove any stored content with this method. It is not delete the selected document physically but retired content cannot be manipulated with CMS operations (modify, search, etc…) The unique identifier and the document’s identifier are given as parameters. The method returns the removed document.

QueryResult[] queryExist(long Identity, HashMap<String, String> keysValues)

We can query the Exis DBMS directly if we have the proper identifier and the filtering phrases presented as a HashMap. The returned value is an array of QueryResult objects.

QueryResult[] queryDaisy(long Identity, String whereCondition)

We can query the Daisy Repository directly if we have the proper identifier and the appropriated query string. The returned value is an array of QueryResult objects.

QueryResult[] Query(long Identity, HashMap<String, String> keysValues, String whereCondition)

We can also define complex queries which asks the Exis DBMS and based on the URI list that result Daisy repository will be queried as well. The returned value is an array of QueryResult objects.
12. ANNEX A – Use cases

12.1. Actors

12.1.1. repository manager

The user that manages the repository (upload and removal of documents and/or versions)

<table>
<thead>
<tr>
<th>specialises</th>
<th>end-user</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>upload legal resource</td>
</tr>
<tr>
<td>remove legal resource</td>
</tr>
</tbody>
</table>

12.1.2. system administrator

The user that administers the system

<table>
<thead>
<tr>
<th>specialises</th>
<th>end-user</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>install</td>
</tr>
<tr>
<td>uninstall</td>
</tr>
<tr>
<td>create new user</td>
</tr>
<tr>
<td>delete user</td>
</tr>
</tbody>
</table>
12.1.3. end-user

A generic end-user of the system (can be a PA user, a legal knowledge engineer, or a citizen)

<table>
<thead>
<tr>
<th>uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
</tr>
<tr>
<td>logout</td>
</tr>
<tr>
<td>set preferences</td>
</tr>
<tr>
<td>browse repository</td>
</tr>
<tr>
<td>search repository</td>
</tr>
<tr>
<td>show legal resource</td>
</tr>
<tr>
<td>show affected legal resources</td>
</tr>
<tr>
<td>show affecting legal resources</td>
</tr>
<tr>
<td>show version chain</td>
</tr>
<tr>
<td>show consolidation</td>
</tr>
<tr>
<td>show filtered versions</td>
</tr>
<tr>
<td>show repealed legal resources</td>
</tr>
<tr>
<td>show annexes</td>
</tr>
<tr>
<td>show meta-data</td>
</tr>
<tr>
<td>download legal resource</td>
</tr>
</tbody>
</table>

12.2. Use cases

12.2.1. login

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user wants to authenticate him/herself into the system.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The user is authenticated into the system.</td>
</tr>
<tr>
<td>Description</td>
<td>The user provides his/her credentials in response to an authentication request of the system.</td>
</tr>
</tbody>
</table>

12.2.2. logout

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is authenticated into the system.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The user is no more authenticated into the system.</td>
</tr>
<tr>
<td>Description</td>
<td>The system closes the session in response to a user request.</td>
</tr>
</tbody>
</table>
12.2.3. set preferences

**Actors**

<table>
<thead>
<tr>
<th>end-user</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>The user is logged into the system.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>The user’s preferences have been defined.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Starting from any page of the system, the user asks to change his/her preferences and the system provides him/her the preferences page.</th>
</tr>
</thead>
</table>

12.2.4. browse repository

**Actors**

<table>
<thead>
<tr>
<th>end-user</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>The user is logged into the system.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>The user reached the needed information.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>The user browses the repository of legal resources using the structure of the document (MetaLex/CEN) including also the metadata specification using the LKIF-OWL categorisation. He/she should pass through a special view to see the LKIF-OWL categories connected with the structural parts of the text. The user shall also to navigate through the updated text of the legal resources with a special &quot;navigation bar&quot; for seeing all the version chain of the legislative documents he/she is interested in.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contains</th>
<th>show legal resource</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
<th><strong>start</strong> starting from the main page of the system, the user asks the system for the list of the “top level” folders, and the system provides it to the user.</th>
</tr>
</thead>
</table>

**choose folder** repeatedly, the user chooses a folder and the system provides him/her with the list of its sub-folders and of the legal resources belonging to it (and the possibility to navigate back in the folder’s path as well). |

**choose resource** the user chooses one of the legal resources in the current folder and the system shows its contents. |

Includes: show legal resource
### 12.2.5. search repository

![Diagram](image.png)

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The user found the legal resources he/she is interested in (or those legal resources are not in the repository).</td>
</tr>
</tbody>
</table>
| Description | The user search the legal resources by any combination of:  
- number  
- country  
- type of document  
- title  
- article/section n.  
- date of delivery  
- date of enter into force  
- date of publication  
- full-text  
- n. of publication  
Semantic searching should also be possible: selecting a OWLcategory into LKIF-core is possible to find all the document linking it. |
| Contains | show legal resource |

#### Scenario

**start** starting from the main page of the system, the user ask to perform a “simple” search, simply providing a text to search, or an “advanced search”.

(advanced search) optionally, the system shows an “advanced search” page, allowing the user to define a query mixing full-text search with meta-data search.

Condition: the user asked for an “advanced search”.

**show query results** the system shows a page with a link to all the legal resources matching the query, ordered by rank. The results are paginated according with the user preferences.

**choose legal resource** the user chooses one of the legal resources shown in the page and the system shows its contents. Includes: show legal resource
12.2.6. **show legal resource**

**Actors**

- **end-user**

**Preconditions**
The user is logged into the system and a legal resource have been selected.

**Postconditions**
The requested legal resource has been shown, following the user needs.

**Description**
A view of the requested legal resource, according with the user preferences, is shown by the system. The user can ask the system to show a different view.

Views of the legal resource include:
- links for downloading the stored versions of this legal resource,
as well as LKIF related documents;
- the list of all the legal resources affected by this one;
- the list of all the legal resources affecting this one;
- the version chain of the legal resource (all the versions);
- the current version (at a given time) of the consolidated legal resource provided the events that really occurred within that time;
- the current version (at a given time) of the consolidated legal resource provided the given events occurred (filtering);
- the list of all repealed legal resources;
- relationship between main document and annexes;
- information taken from LKIF related documents.
Moreover, the user can download the files describing the selected legal resource.

<table>
<thead>
<tr>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>show affected legal resources</td>
</tr>
<tr>
<td>show affecting legal resources</td>
</tr>
<tr>
<td>show version chain</td>
</tr>
<tr>
<td>show consolidation</td>
</tr>
<tr>
<td>show filtered versions</td>
</tr>
<tr>
<td>show repealed legal resources</td>
</tr>
<tr>
<td>show annexes</td>
</tr>
<tr>
<td>show meta-data</td>
</tr>
<tr>
<td>download legal resource</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>view</strong> the system shows the selected legal resource using the selected view (initially, the preferred view defined in the user’s preferences is selected).</td>
</tr>
</tbody>
</table>

**(select different view)** optionally, the user chooses to change the view of the selected resource and the system shows the selected view.

Extension point: show affected legal resources | show affecting legal resources | show version chain | show consolidation | show filtered versions | show repealed legal resources | show annexes | show meta-data

Condition: the user wants to change its view of the selected resource.

**(download)** optionally, the user downloads the source files describing the legal resource.

Extension point: download legal resource

Condition: the user wants to perform special operations (such as editing or analysing) on the source files.
12.2.7. show affected legal resources

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and a legal resource has been selected.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The list of the legal resources affected by the selected one has been shown.</td>
</tr>
<tr>
<td>Description</td>
<td>The system shows the list of the legal resources affected by the selected one. The user can ask the system to show any of them.</td>
</tr>
</tbody>
</table>

12.2.8. show affecting legal resources

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and a legal resource has been selected.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The list of the legal resources affecting the selected one is shown.</td>
</tr>
<tr>
<td>Description</td>
<td>The system shows the list of the legal resources affecting the selected one. The user can ask the system to show any of them.</td>
</tr>
</tbody>
</table>

12.2.9. show version chain

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and a legal resource has been selected.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The version chain of the selected resource has been shown.</td>
</tr>
<tr>
<td>Description</td>
<td>The system shows the version chain of the selected resource. The user can ask the system to show any version, organised following the legal event occurred to the document: official modifications produce these kind of events.</td>
</tr>
</tbody>
</table>
12.2.10. **show consolidation**

![Diagram](image1)

### Actors

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and the legal resource has been selected.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The consolidated version (at the given time) of the legal resource is shown.</td>
</tr>
<tr>
<td>Description</td>
<td>The system shows the consolidated version of the selected legal resource at the &quot;current&quot; time. The user can change the current time, that, initially, is the system time, and the system update accordingly the view. The current time is maintained for the whole session, unless the user decide to change it.</td>
</tr>
</tbody>
</table>

12.2.11. **show filtered versions**

![Diagram](image2)

### Actors

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and a legal resource has been selected.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The consolidated version of the selected legal resource, according to some query event, is shown.</td>
</tr>
<tr>
<td>Description</td>
<td>The system shows the consolidated version of the selected legal resource to a due time and according to a selected event (e.g. publication, creation, modification, etc.). The user can change the current time, that initially is the system time, and the current query events. Both are maintained until the end of the session, unless the user changes them.</td>
</tr>
</tbody>
</table>
### 12.2.12. show repealed legal resources

**Actors**
- end-user

**Preconditions**
The user is logged into the system and a legal resource has been selected.

**Postconditions**
The list of all the repealed legal resources has been shown.

**Description**
The system shows the list of all the repealed legal resources. The user can ask the system to show any of them. Often the repealed legal document are no longer reachable. This function aims to permit to have the list about all the documents removed by the Legal System.

### 12.2.13. show annexes

**Actors**
- end-user

**Preconditions**
The user is logged into the system and a legal resource has been selected.

**Postconditions**
The annexes of the selected legal resource have been shown.

**Description**
The system shows the annexes of the selected legal resource following the hierarchy and in accordance with the time of navigation. It is not a rarity that the case where an annex in a due time is moved to the original hierarchy position. So this information is very important for maintain the coherence of the cross-references both in the future and in the past.

### 12.2.14. show meta-data

**Actors**
- end-user

**Preconditions**
The user is logged into the system and a legal resource has been selected.

**Postconditions**
The meta-data of the selected legal resource, taken from LKIF related documents, have been shown.

**Description**
The system shows the meta-data of the selected legal resource, taken from LKIF related documents.
12.2.15. **download legal resource**

![Diagram: end-user downloads legal resource](image)

<table>
<thead>
<tr>
<th>Actors</th>
<th>end-user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and the URL of a legal resource has been individuated.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The requested resource has been downloaded.</td>
</tr>
<tr>
<td>Description</td>
<td>The legal resource, as well as the LKIF related documents, is downloaded providing the appropriate URL to the system. Moreover, a Web Service allows external applications to download the resource and the related documents as well.</td>
</tr>
</tbody>
</table>

12.2.16. **upload legal resource**

![Diagram: repository manager uploads legal resource](image)

<table>
<thead>
<tr>
<th>Actors</th>
<th>repository manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The user is logged into the system and has the right to upload a legal resource.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The legal resource has been added to the system and it is accessible by means of a URL (the URL can contain the version number; if it is not specified, the URL points to the latest version).</td>
</tr>
<tr>
<td>Description</td>
<td>The user adds a new legal resource to the system uploading the Metalex-CEN document, that is immediately validated against the XSD. The system understand from the document URI whether it is a new document, version, or a replacement of an existing document or version. Moreover, the user can also upload, with a similar versioning system, the LKIF-OWL, the LKIF-rules, LKIF-argument graph, LKIF-evidence, etc., describing the Metalex-CEN document. These operations can be performed by programs too by means of Web Services.</td>
</tr>
</tbody>
</table>
### 12.2.17. remove legal resource

**Actors**
- repository manager

**Preconditions**
The user is logged into the system and has the right to remove legal resources.

**Postconditions**
The legal resource has been removed from the repository.

**Description**
The user chooses a (version of a) legal resource browsing or searching the repository and eventually asks the system to remove it.

This operation can be performed by programs too by means of Web Services.

**Contains**
- browse repository
- search repository

**Scenario**

(browse or search) optionally, the user browses or searches the repository.

Extension point: browse repository | search repository

remove the user asks the removal of the reached legal resource.

confirm the system asks a confirmation to the user, warning him/her that all the information connected with the resource will be removed too.

(actual removal) optionally, the system actually removes the legal resource.

Condition: the user confirmed the operation.

### 12.2.18. install

**Actors**
- system administrator

**Preconditions**
The user has the administration rights and wants to install the system.

**Postconditions**
The system is installed.

**Description**
The user performs the install procedure.
12.2.19. uninstall

Actors  | system administrator
---|---
Preconditions  | The user has the administration rights, the system is installed and the user wants to uninstall it.
Postconditions | The system is no more installed.
Description  | The user performs the uninstall procedure.

12.2.20. create new user

Actors  | system administrator
---|---
Preconditions  | The user is logged in and wants to create a new user for the system.
Postconditions | A new user for the system has been created.
Description  | The system administrator creates the new user and assigns him/her the credentials and the roles. Default preferences are assigned to the new user.

12.2.21. delete user

Actors  | system administrator
---|---
Preconditions  | The user is logged in and wants to delete an existent user.
Postconditions | The user is deleted from the system (and his/her preferences as well).
Description  | The system administrator deletes all the data of the user.
13. **APPENDIX B - Presentation of the content**

13.1. **Login**

![Login page](image1)

Figure 27

13.2. **Search**

![Search page](image2)

Figure 28
13.3. Preferences

![Preferences Page](image)

Figure 29