DELIVERABLE

Project Acronym: EuDML
Grant Agreement number: 250503
Project Title: The European Digital Mathematics Library

Deliverable 4.2 – EuDML global system functional specification and design - Appendix

Revision: 1.0

Authors:

Aleksander Nowiński (ICM)
Wojtek Sylwestrzak (ICM)
Krzysztof Wojciechowski (ICM)
José Borbinha (IST)
Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Organisation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>30.08.11</td>
<td>José Borbinha</td>
<td>IST</td>
<td>Corrected “typos” and missing images from REPOX</td>
</tr>
<tr>
<td>v0.5</td>
<td>02.09.11</td>
<td>Aleksander Nowiński</td>
<td>ICM</td>
<td>Copyedit</td>
</tr>
<tr>
<td>v1.0</td>
<td>06-09-11</td>
<td>José Borbinha</td>
<td>IST</td>
<td></td>
</tr>
</tbody>
</table>

The document versions are controlled by the EuDML SVN version control system. The above revisions are for informational purpose only and do not reflect the actual SVN version numbers.

This document complements the “Deliverable 4.2 – EuDML global system functional specification and design”

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.
# Table of Contents

1. Introduction ................................................................................................................................... 9  
2. Content Aggregation Service ...................................................................................................... 10 
   2.1. REPOX Process Engine Architecture .................................................................................. 10 
   2.2. Architecture of REPOX Services .......................................................................................... 13 
   2.3. REPOX Information Model .................................................................................................. 14 
   2.4. REPOX Services and Interfaces ............................................................................................ 16 
3. Storage Service ............................................................................................................................ 18 
   3.1. Module responsibility ........................................................................................................... 18 
   3.2. Usage patterns ..................................................................................................................... 18 
   3.3. Non-functional requirements met by the module .................................................................... 18 
   3.4. Implementation details ......................................................................................................... 18 
   3.5. Data stored in record ............................................................................................................. 19 
   3.6. Record format ........................................................................................................................ 19 
   3.7. Store service implementations ............................................................................................. 20 
      3.7.1. YADDA MD-Storage/Archive implementation ................................................................. 20 
      3.7.2. YStore implementation .................................................................................................. 20 
   3.8. Content retrieval rules for item-record .................................................................................... 20 
   3.9. Dependencies and prerequisites ............................................................................................ 21 
   3.10. API description .................................................................................................................... 21 
   3.11. Important file/class/package locations ................................................................................ 21 
   3.12. Configuration ........................................................................................................................ 21 
   3.13. Deployment .......................................................................................................................... 22 
   3.14. Possible extensions .............................................................................................................. 22 
4. Processing Service ....................................................................................................................... 23 
   4.1. Module responsibility ............................................................................................................ 23 
   4.2. Non-functional requirements met by the module .................................................................. 23 
   4.3. Implementation details ......................................................................................................... 23 
   4.4. Core modules description ..................................................................................................... 23 
      4.4.1. ProcessService and ProcessFacade interfaces ................................................................. 23 
      4.4.2. Implementations ............................................................................................................ 24 
      4.4.3. IErrorHandler module description .............................................................................. 24 
      4.4.4. Checking errors fatality ................................................................................................. 24
4.4.5. errors threshold mechanism ................................................................. 25
4.4.6. implementations .................................................................................. 25
4.5. Gathering statistics description ............................................................... 25
4.6. implementations ...................................................................................... 26
4.7. Source module description ..................................................................... 26
4.8. MessageRegistry module description ..................................................... 26
4.9. regular messages creation ...................................................................... 26
4.10. orchestration messages creation ............................................................ 26
  4.10.1. maintaining number of created message instances ......................... 26
  4.10.2. sending events to registered event consumers ............................... 27
4.11. Flow definition ....................................................................................... 27
  4.11.1. orchestration description ................................................................. 27
  4.11.2. starting orchestration ..................................................................... 27
  4.11.3. finishing/interrupting orchestration ............................................... 27
  4.11.4. role of FlowSyncPanel .................................................................... 28
4.12. node wrappers functionality .................................................................. 28
4.13. custom namespaces definition .............................................................. 28
  4.13.1. chain .............................................................................................. 29
  4.13.2. recipient-list-router ....................................................................... 29
  4.13.3. source ........................................................................................... 30
  4.13.4. processor ....................................................................................... 30
  4.13.5. writer ............................................................................................ 31
  4.13.6. Properties ...................................................................................... 31
4.14. Dependencies and prerequisites ............................................................. 31
4.15. API description ...................................................................................... 31
  4.15.1. Sketch of the API .......................................................................... 31
  4.15.2. Nodes APIs ................................................................................... 32
    4.15.2.1 Stateful nodes .......................................................................... 32
    4.15.2.2 Source node ............................................................................ 32
    4.15.2.3 Processing nodes ...................................................................... 33
    4.15.2.4 Writing nodes .......................................................................... 33
4.16. Usage example ...................................................................................... 34
4.17. Important file/class/package locations .................................................... 37
4.18. Configuration ........................................................................................ 37
4.19. Possible extensions ............................................................................... 37
5. User Directory Service .............................................................................. 38
  5.1. Module responsibility .......................................................................... 38
  5.2. Non-functional requirements met by the module ............................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>Dependencies and prerequisites</td>
<td>52</td>
</tr>
<tr>
<td>7.5</td>
<td>API description</td>
<td>52</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Sketch of the API</td>
<td>52</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Important file/class/package locations</td>
<td>53</td>
</tr>
<tr>
<td>7.6</td>
<td>Configuration</td>
<td>53</td>
</tr>
<tr>
<td>8</td>
<td>Similarity Service</td>
<td>54</td>
</tr>
<tr>
<td>8.1</td>
<td>Module responsibility</td>
<td>54</td>
</tr>
<tr>
<td>8.2</td>
<td>Implementation details</td>
<td>54</td>
</tr>
<tr>
<td>8.3</td>
<td>YADDA implementation</td>
<td>54</td>
</tr>
<tr>
<td>8.4</td>
<td>GensimEudml implementation</td>
<td>55</td>
</tr>
<tr>
<td>8.5</td>
<td>Alternative implementations using standard API</td>
<td>55</td>
</tr>
<tr>
<td>8.6</td>
<td>Dependencies and prerequisites</td>
<td>56</td>
</tr>
<tr>
<td>8.7</td>
<td>API description</td>
<td>56</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Sketch of the API</td>
<td>56</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Important file/class/package locations</td>
<td>56</td>
</tr>
<tr>
<td>8.7.3</td>
<td>Similarity document</td>
<td>56</td>
</tr>
<tr>
<td>8.7.4</td>
<td>Similarity result</td>
<td>56</td>
</tr>
<tr>
<td>8.8</td>
<td>Similarity facade</td>
<td>57</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Filters</td>
<td>58</td>
</tr>
<tr>
<td>8.8.2</td>
<td>Similarity facade compatible with search API</td>
<td>58</td>
</tr>
<tr>
<td>8.8.3</td>
<td>Indexing facade</td>
<td>59</td>
</tr>
<tr>
<td>8.9</td>
<td>Configuration</td>
<td>61</td>
</tr>
<tr>
<td>8.9.1</td>
<td>Facade configuration</td>
<td>61</td>
</tr>
<tr>
<td>8.9.2</td>
<td>Skeleton configuration of the service</td>
<td>61</td>
</tr>
<tr>
<td>8.9.3</td>
<td>YADDA similarity module configuration</td>
<td>61</td>
</tr>
<tr>
<td>8.9.4</td>
<td>YADDA similarity search index configuration</td>
<td>62</td>
</tr>
<tr>
<td>8.9.5</td>
<td>YADDA language/categorization facade configuration</td>
<td>63</td>
</tr>
<tr>
<td>8.9.6</td>
<td>Possible extensions</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>Annotation Service</td>
<td>65</td>
</tr>
<tr>
<td>9.1</td>
<td>Module responsibility</td>
<td>65</td>
</tr>
<tr>
<td>9.2</td>
<td>Non-functional requirements met by the module</td>
<td>65</td>
</tr>
<tr>
<td>9.3</td>
<td>Implementation details</td>
<td>65</td>
</tr>
<tr>
<td>9.4</td>
<td>API description</td>
<td>66</td>
</tr>
<tr>
<td>9.5</td>
<td>Configuration</td>
<td>66</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Configuration of the YADDA MD-Storage</td>
<td>66</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Configuration of the Sesame RDF store</td>
<td>67</td>
</tr>
<tr>
<td>9.6</td>
<td>Possible extensions</td>
<td>67</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>10.ID Service</td>
<td>10.1 Module responsibility</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>10.2 Non-functional requirements met by the module</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>10.3 Implementation details</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>10.4 Dependencies and prerequisites</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>10.5 API description</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>10.6 Data Structures</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>10.7 Important file/class/package locations</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>10.8 Configuration</td>
<td>69</td>
</tr>
<tr>
<td>11.YADDA MD-Storage Service</td>
<td>11.1 Module responsibility</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>11.2 Non-functional requirements met by the module</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>11.3 Implementation details</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>11.4 Dependencies and prerequisites</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>11.5 API description</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>11.6 Important file/class/package locations</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>11.7 Configuration</td>
<td>71</td>
</tr>
<tr>
<td>12.YADDA Archive Service</td>
<td>12.1 Module responsibility</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>12.2 Non-functional requirements met by the module</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>12.3 Implementation details</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>12.4 Dependencies and prerequisites</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>12.5 API description</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>12.6 Important file/class/package locations</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>12.7 Configuration</td>
<td>75</td>
</tr>
<tr>
<td>13.YaddaWebLite</td>
<td>13.1 Module responsibility</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>13.2 Non-functional requirements met by the module</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>13.3 Implementation details</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>13.3.1 Detail service</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>13.3.2 Filtering service</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>13.3.3 Paging service</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>13.3.4 Notification service</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>13.3.5 Hierarchy Tree</td>
<td>79</td>
</tr>
</tbody>
</table>
13.3.6. Error handling service ................................................................. 79
13.4. Dependencies and prerequisites .................................................. 79
13.5. API description ........................................................................... 80
13.6. Important file/class/package locations ....................................... 80
13.7. Configuration .............................................................................. 80
13.7.1. Detail service ........................................................................... 80
13.7.2. Paging service ....................................................................... 81
13.7.3. Hierarchy tree service .............................................................. 82
13.8. Possible extensions ................................................................... 82
13.8.1. Application-specific datasources and data formats .................... 82
13.8.2. Application-specific way of displaying publications ................ 83
13.8.3. Application-specific source of hierarchy data and way of displaying hierarchy tree ................................. 83
14. Metadata Enhancement Services and Tools .................................... 84
14.1. MiaS ............................................................................................ 84
14.1.1. Design .................................................................................. 84
14.1.2. Indexing ................................................................................ 84
14.1.2.1. Tokenization .................................................................. 85
14.1.2.2. Formulae modifications .................................................... 85
14.1.2.3. Formulae weighting .......................................................... 85
14.1.3. Searching ............................................................................. 87
14.1.4. Evaluating ............................................................................ 87
14.2. Gensim ...................................................................................... 87
14.2.1. Module overview .................................................................. 88
14.2.2. Persistency .......................................................................... 88
14.2.3. Concurrency ........................................................................ 88
14.2.4. Security and access rights ...................................................... 88
14.2.5. Installation and deployment .................................................... 88
14.2.6. API description .................................................................... 89
14.2.6.1. Sketch of the API ........................................................... 89
14.2.6.2. Important file/class/package locations ............................... 89
14.2.7. Implementation details .......................................................... 89
14.2.8. Error control ....................................................................... 90
14.2.9. Testing procedure ................................................................. 90
14.2.10. Possible extensions ............................................................... 90
14.3. BibRefAnalysis .......................................................................... 90
14.3.1. Module responsibility ............................................................ 90
14.3.2. Non-functional requirements met by the module .................... 90
14.3.3. Implementation details .......................................................... 91
14.3.3.1. Workflows .................................................................... 91
14.3.3.2. Matching algorithm ......................................................... 91
14.3.4. Dependencies and prerequisites .................................................. 92
14.3.5. API description ............................................................................. 92
  14.3.5.1. Important file/class/package locations .......................................... 92
14.3.6. Configuration .............................................................................. 92
14.3.7. Possible extensions ..................................................................... 92

14.4. PdfJblm .......................................................................................... 93
  14.4.1. Module responsibility ................................................................. 93
  14.4.2. Non-functional requirements met by the module ......................... 93
  14.4.3. Implementation details .............................................................. 93
  14.4.4. Dependencies and prerequisites ................................................ 94
    14.4.4.1. Dependencies ................................................................. 94
  14.4.5. API description ........................................................................ 94
  14.4.6. Configuration and Requirements ............................................... 94
  14.4.7. Possible extensions .................................................................. 95

14.5. PdfToTextViaOCR ........................................................................ 95
  14.5.1. Module responsibility ............................................................... 95
  14.5.2. Non-functional requirements met by the module ......................... 95
  14.5.3. Implementation details .............................................................. 96
  14.5.4. Dependencies and prerequisites ................................................ 96
  14.5.5. API description ........................................................................ 96
    14.5.5.1. Sketch of the API ............................................................. 96
    14.5.5.2. Important file/class/package locations .................................. 96
  14.5.6. Configuration and Requirements ............................................... 96
  14.5.7. Possible extensions .................................................................. 96

15. Deployment of EuDML ..................................................................... 97
  15.1. Required resources ...................................................................... 97
  15.2. Example configuration for the backend .......................................... 97
  15.3. Example configuration for the User Interface ................................. 97
1. Introduction

This document complements “Deliverable 4.2 – EuDML global system functional specification and design” by describing in detail the key components and services of the EuDML software system. It is intended as the general reference document for development purposes. Therefore it is expected that the document is going to be continuously updated to reflect the evolution of the system.

The EuDML system follows the Service Oriented Architecture paradigm, and consists of a number of inter-communicating services. The basic set of EuDML services is introduced in this section. In the future the set can and is intended to be extended to support new or enhanced functionalities of the system.

Figure 1 depicts the general dependencies between the services.

![EuDML services diagram](image)

Figure 1: EuDML services diagram.
2. Content Aggregation Service

The EuDML Content Aggregation Service uses REPOX technology to manage the harvesting processes of the metadata and full-text provided by the Data Providers, plus the processes for data transformation and normalization.

In this role, REPOX provides the following main functionalities:

- The registration of data providers, of their collection descriptions (one data provider might make available to EuDML more than one collection), and the configurations for the harvesting of the relating metadata and full-text.
- The automatic and manual harvesting of the metadata by OAI-PMH (according to configurations and options provided by the data providers and the decisions of the administrators of the central service) or by HTTP or FTP. It is provided Support for multiple metadata formats (OAI-DC is assumed by default, but any other format is also possible). The harvesting of the full-text is expected to be done through HTTP or FTP.
- Monitoring of the quality of service of the OAI-PMH servers, including statistics.

2.1. REPOX Process Engine Architecture

REPOX is designed as a SOA infrastructure, managed by a Process Engine as shown in the Figure 2. The core of the Process Engine is the Process Orchestrator, composed by a REPOX Process Manager that launches new processes in the JBPM engine and the Process State Manager which monitors the state of each process. Both these managers share the access to a common list of processes, managed by the Process Planning.

Each process consists in an orchestration of a set of REPOX Services that are registered within the JBPM. The Process Orchestrator is constantly monitoring all the running processes.

The Process Orchestrator provides an interface so that new processes can be added. The processes can be designed visually in the BPMN 2.0 notation, which is then coded in XML, according to the format defined by the OMG – Open Management Group.

Regarding the web application, for human users, it is composed by a client side containing a Process Editor that supports the visual definition of new processes, and by a Process Instance Viewer that allows the run-time monitoring of each process instance (Figure 3 shows an example of how a list of actual running processes can be presented).

The processes defined on the client side are persisted in an extended BPMN 2.0, through the Process Definitions Manager. Initialized process instances are started through the Process Orchestrator, called by the Process Instances.

Figure 4 shows the particular example of a process containing the parallel ingest of two different data sets. On the right side of the image there is a panel showing the log messages occurring during the process. The top ingest has already finished the ingestion (green represents what already was executed), with the result of 36 records, and the bottom ingest is still working (blue represents the actual executions).

Figure 5 shows an example of the XML coding of a process for a single ingest.
Figure 2: REPOX Process Engine Architecture

Figure 3: Process runtime monitoring
**Figure 4:** Example of a visualization of a running process

```xml
    <process id="process_rpm_876221c0-0cff-4ac5-bdf6-c5e274b5cf4a" processType="Public">
        <property id="harParams_0"/>
        <property id="result_0"/>
        <startEvent name="" id="rpm_c5d83427-d953-4aa4-844a-7d5c83390225"/>
        <task id="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" name="" tns:taskName="IngestWI">
            <ioSpecification>
                <dataInput id="hrparam_0" name="harParams_0"/>
                <dataOutput id="reslt_0" name="HarvestResult"/>
                <inputSet>
                    <dataInputRefs>hrparam_0</dataInputRefs>
                </inputSet>
                <outputSet>
                    <dataOutputRefs>reslt_0</dataOutputRefs>
                </outputSet>
                <dataInputAssociation>
                    <sourceRef>harParams_0</sourceRef>
                    <targetRef>hrparam_0</targetRef>
                </dataInputAssociation>
                <dataOutputAssociation>
                    <sourceRef>reslt_0</sourceRef>
                    <targetRef>result_0</targetRef>
                </dataOutputAssociation>
            </ioSpecification>
            <dataIn...
        </task>
        <endEvent id="rpm_68dae2d2-3f8f-4d08-9981-32709e1d2a9b"/>
        <sequenceFlow sourceRef="rpm_c5d83427-d953-4aa4-844a-7d5c83390225" targetRef="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" name="" id="rpm_ddefa7d5-74e4-4267-870f-0eaad4483446" state="NOT_PASSED"/>
        <sequenceFlow sourceRef="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" targetRef="rpm_68dae2d2-3f8f-4d08-9981-32709e1d2a9b" name="" id="rpm_31d514ae-16fa-4337-b0c0-6710030fbcce" state="NOT_PASSED"/>
    </process>
    <processGUI processRef="process_rpm_876221c0-0cff-4ac5-bdf6-c5e274b5cf4a" id="process_rpm_876221c0-0cff-4ac5-bdf6-c5e274b5cf4a_gui">
        <startEventShape eventRef="rpm_c5d83427-d953-4aa4-844a-7d5c83390225" x="6" y="8" name="" id="rpm_c5d83427-d953-4aa4-844a-7d5c83390225" state="NOT_PASSED"/>
        <activityShape activityRef="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" type="INGEST" x="14" y="8" name="" id="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" state="NOT_PASSED"/>
        <endEventShape eventRef="rpm_68dae2d2-3f8f-4d08-9981-32709e1d2a9b" x="23" y="8" name="" id="rpm_68dae2d2-3f8f-4d08-9981-32709e1d2a9b" state="NOT_PASSED"/>
    </processGUI>
    <parameters>
        <parameter type="INGEST" taskID="rpm_4e05c04f-d137-4060-b2c5-790f01faffd9" sourceRef="harParams_0" resultID="result_0" sendMail="true" dsID="arquivocmlx" fullIngest="true"/>
    </parameters>
</definitions>
```

**Figure 5:** Definition of a single harvest process
2.2. Architecture of REPOX Services

The REPOX basic services, which are registered within the JPMN component, are (Figure 6):

- **The Data Manager** service aggregates the core components of the system:
  - The **Data Provider Manager**
  - The **Data Set Manager**
  - The **Harvester Manager** is the service responsible for harvesting the records from the data providers. It can use one of the following interfaces: OAI-PMH client, HTTP-get, file system folder or FTP client. REPOX could harvest files in the format ISO2709, MarcXML, MarcXchange, ESE, NLM or ultimately any XML format. For each record harvested, the Data Manager component creates and stores the indexes of the access points in Data Set Repository.

- **The Data Providers Repository** consists of a set of XML files registering the necessary information about the Data Providers, their Data Sets, and the respective ingests.

- **The Data Set Repository** is a database storing the records of the data sets ingested.

- **The Full-text Manager** is the service responsible for analysing the ingested records and harvesting the contents linked from them, which is stored in the full-text Full-Text Repository.

- **The Full-Text Repository** provides the local storage for the harvest full-text. In its simplest version it is just a local system, but if required it can be replaced by a specific alternative repository.

- **The Data Manager Exporter** makes available the bibliographic metadata to external systems through the interface OAI-PMH server. It also allows the external access to those records through the file system, as also the access to the full-text. An example of an external client of this service can be **Profiling Service**, which can provide basic profiling for each interested data set.

- **The Content Transformation Manager** manages the transformations between metadata formats (schemas) and is also the responsible for the application of those transformations to specific metadata sets (it is expected, in a near future, to complement REPOX with a MDR - Metadata Registry, a service to store, support management and provide XSLT transformations of datasets, namely to NLM).

- **The Repox2Yadda** is a service that communicates with YADDA Storage Service using the corresponding API. The metadata and the full-texts harvested by REPOX are stored in YADDA Storage under the control of this service. This service is included according to the specific requirements of the EuDML project.

- **The Yadda2Repox** is a service that communicates with YADDA Storage Service using the corresponding API. The metadata consolidated by EuDML and stored in YADDA Storage that is to be made available to external parties (typically by OAI-PMH) is ingested by this service, in order to be subsequently provided to the external party through the REPOX OAI-PMH server, or other mechanism, as applicable.
2.3. REPOX Information Model

Figure 7 shows the REPOX Information Model (which is shared by the REPOX services).

Data Providers are entities with one or more collections of records (record sets), each with a Data Source.

Data Sources are either OAI-PMH or Directory Importer, the first meaning that the records will be harvested from an OAI-PMH server and the second meaning a folder in the file system. To ingest from folders, REPOX recognizes three strategies: Simple File Extract, ISO2709 File Extract and MarcXchange File Extract.

Simple File Extract is the default method, where there is no processing of the XML records. The only associated logic is the validation of the XML file itself. ISO2709 File Extract and MarcXchange specifically target those formats. ISO2709 File Extract requires the file Character Set and the format variant because even though ISO2709 is a standard, some institutions do not follow it exactly. Because ISO2709 is not an XML format and REPOX only handles XML, the format is ingested as MarcXchange because there is no data loss in that transformation. In the three scenarios the files may be zipped and unzipped prior to ingestion.

Data Sources can have associated ScheduledTasks, by scheduling an Ingest of records or an export of the records to the file system. A Task is a managed action in REPOX. Scheduled Tasks are tasks that occur at specific times with a periodicity (unique harvest, daily, weekly and every number of months).

Access Points (AP) enable the retrieval of the records by more than only their identifiers. For that purpose, Access Points are associated to Data Sources, to define how to process the pertinent information for indexing. These AP are used by the AccessPointsManager (APM) to extract the
relevant data from each record and build the respective indexes. Those indexes are maintained in a relational database for efficiency, as they are not part of the fundamental model.

A Metadata Transformation is a translation between two metadata formats (ex: local schema to NLM). Every Data Source can have any number of transformations. The transformations are stored as XSLT files, even though it is possible to create a visual mapping of them which is stored in an intermediate XML format internally to allow editing. The records can be retrieved by OAI-PMH in their original format or any format for which has been configured a mapping to. The mapping is performed by request and not stored, because the performance impact is not noticeable.

The record identifiers used in REPOX can be associated in two ways: generated by REPOX or extracted from each record using an XPath expression. The advantage of using extracted identifiers is that it is possible to update just the changes because the records can be recognized by the identifier.

Figure 7: REPOX Information Model
2.4. REPOX Services and Interfaces

The REPOX has a graphic user interface (example in Figure 8) to be used by humans.

REPOX also provides to the outside a set of REST services for Data Providers (Figure 9), Data Sources (Figure 10) and records (Figure 11).

Figure 8: REPOX Graphical User Interface

<table>
<thead>
<tr>
<th>Data Source Set</th>
<th>OA-PMH Schemes</th>
<th>Ingest Type</th>
<th>Last Ingest</th>
<th>Next Ingest</th>
<th>Records</th>
<th>Ingest Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gallica</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>2.061</td>
<td></td>
</tr>
<tr>
<td>CEDRAS</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>40.478</td>
<td></td>
</tr>
<tr>
<td>EDP_COCV</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>EDP_TTA</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>EDP_M2AN</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>EDP_MHN</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>EDP_PS</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>EDP_RO</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>EDP_BAS</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>EDP_M4A</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>EDP_M4M</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>EDP_M4P</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>57.179</td>
<td></td>
</tr>
<tr>
<td>EDP_M4Q</td>
<td>OA-PMH OA-PMH</td>
<td>OA-PMH OA-PMH</td>
<td>2011-07-14</td>
<td>2011-07-14</td>
<td>1.549</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Data Provider operations

- Retrieve all available Data Providers
  - Syntax: "REPOX installation"/rest/dataProviders/list
- Creates a new Data Provider
  - Syntax: "REPOX installation"/rest/dataProviders/create
- Updates a Data Provider
  - Syntax: "REPOX installation"/rest/dataProviders/update
- Deletes a Data Provider
  - Syntax: "REPOX installation"/rest/dataProviders/delete
- Retrieve all available Data Sources
  o Syntax: “REPOX installation”/rest/dataSources/list
- Creates/Updates a Data Source from OAI-PMH server
  o Syntax: “REPOX installation”/rest/dataSources/{createOai/updateOai}
- Creates/Updates a Data Source from FTP server
  o Syntax: “REPOX installation”/rest/dataSources/
    {createDataSourceFtp/updateDataSourceFtp}
- Creates/Updates a Data Source from HTTP server
  o Syntax: “REPOX installation”/rest/dataSources/
    {createHttp/updateHttp}
- Creates/Updates a Data Source from local folder
  o Syntax: “REPOX installation”/rest/dataSources/
    {createFolder/updateFolder}
- Deletes a Data Source
  o Syntax: “REPOX installation”/rest/dataSources/delete
- Starts the Data Source ingestion
  o Syntax: “REPOX installation”/rest/dataSources/startIngest
- Stops the Data Source ingestion
  o Syntax: “REPOX installation”/rest/dataSources/stopIngest
- Schedules a Data Source ingestion
  o Syntax: “REPOX installation”/rest/dataSources/scheduleIngest
- Retrieves the list of all schedules from a specific Data Source
  o Syntax: “REPOX installation”/rest/dataSources/scheduleList
- Retrieves the list of all harvesting Data Sources
  o Syntax: “REPOX installation”/rest/dataSources/harvesting
- Retrieves the Data Source status
  o Syntax: “REPOX installation”/rest/dataSources/harvestStatus
- Retrieves the last Data Source ingest log
  o Syntax: “REPOX installation”/rest/dataSources/log

Figure 10: Data Source operations

- Retrieves a specific record
  o Syntax: “REPOX installation”/rest/records/getRecord
- Save record
  o Syntax: “REPOX installation”/rest/records/saveRecord
- Delete record (mark as deleted)
  o Syntax: “REPOX installation”/rest/records/deleteRecord
- Erase Record (remove permanent)
  o Syntax: “REPOX installation”/rest/records/eraseRecord

Figure 11: Record operations
3. Storage Service

The Storage Service stores 'records' identified by the EuDML identifiers. Record is a multi-part construct, where parts may be text or binary files. Record parts are fetched separately, to avoid unnecessary transfer. Therefore, record can be complex, composed of multiple large files (like few scanned PDFs), but it is light in access - large parts are not fetched unless requested.

Records keep information about:

- parts
- timestamps
- timestamps of the parts
- specific tags assigned to each record.

Later on, full versioning information will be available.

The underlying information is already converted to the usable information by proper helper methods. For example, tags having specific meaning (like source provider or some boolean property) are accessible through proper properties of the record, not only as a text array to hide user complexity.

The storage service is oriented on get/put(id) operations. It may be queried, but this is an expensive operation and shall be used only in dedicated components.

The storage service is in fact an API over existing YADDA services and therefore it may be updated easily according to the system needs.

3.1. Module responsibility

A EuDML storage service is a “middle layer service”, which is a core repository for the system. It stores data in binary and text format and is used to store all data in the system, including (and other data as it will appear):

- metadata
- content
- other files (plaintext, math)

User management data is stored separately in user management system.

3.2. Usage patterns

Storage service is defined by:

- Storage Usage Patterns In UI

3.3. Non-functional requirements met by the module

It is possible to introduce security layer on store service allowing licensing of stored content and restricting access to resources based on some credentials.

3.4. Implementation details

Store service provides ItemRecord objects and each record is not lazy-fetching data, and this is deliberate decision. It is assumed that each query to the service shall be done with
understanding of the consequences and exception handling. Lazy fetching of the parts is convenient, but it will lead to abuse and may cause invocation of the service in unpredictable parts of the system (e.g. JSP or JSF views), where it is very hard to handle possible exceptions and not likely to be done properly.

Binary content handling is different. As the binary content files may be unpredictably long, and sometimes are expected only to be served as a stream, record binary parts may be fetched as:

- stream - direct invocation of service with part address is required
- byte[] - if it is known, that resource has limited size and may be handled this way (e.g. icon image)
- file - resource is downloaded and stored locally as file in temporary directory - useful for enhancers running programs on local resources.

**Note on file handling**: not a File instance, but a ContentFileHandle instance is returned when querying for the content file as file. This returns the handle to a local, read-only copy of the content file. This file is managed by the central cache management and will be removed when ContentFileHandle is released (release() method) or it is garbage collected. Therefore as long as ContentFileHandle is present, the file exists in filesystem and may be used.

### 3.5. Data stored in record

Records store the following content:

- **source metadata** (possibly multiple files) - data files in source formats, as obtained from the providers
- **base metadata** (NLM format) - data which is an official compilation of source metadata and is used by the system, including presentation and indexing (if enhanced version not present).
- **enhanced metadata** (NLM format) - if base metadata has been enhanced by some analysis within the repository (citation resolution, content analysis) then it is stored here. Used by the system as core metadata format.
- **plaintext/math indexing content** - content in plain text (or mathml) format, used to provide fulltext index.
- **content files** - PDFs and other contents used to deliver to users

It is important to understand that base metadata is core metadata for the record. It is obtained by conversion of the source files and possibly merging multiple content information. If source is a single NLM, then conversion is trivial, yet still must be done.

Enhanced metadata is assumed to be something that can be derived from base metadata. If base metadata is changed then, as enhancements lead to creation of the enhanced metadata, nlm must be performed again.

### 3.6. Record format

Record contains the following information:

- tags - short string which allows to query. To be used with care (not over used)
- timestamp - last record modification

Data is stored in parts and each part is identified by a string. Each part has its own timestamp. The following parts shall be used:
As the record already knows the structure and possible requirements, it also provides helper methods to obtain proper files:

- NLM to be used
- plaintext/math to be indexed
- content to be used to serve to the user

### 3.7. Store service implementations

#### 3.7.1. YADDA MD-Storage/Archive implementation

The current implementation is based on YADDA MD-Storage and YADDA Archive services. These are two similar services, where one is oriented on metadata storage and other on large data files storage. Both services rely on PostgreSQL database and Archive uses disk folder as well to store large data files. The Storage Service hides complexity of the management and assignment of the files from one to other.

#### 3.7.2. YStore implementation

As currently YStore, a new generation of YADDA storage service is being developed (supporting distribution and high throughput) and a new version of storage service will use it.

### 3.8. Content retrieval rules for item-record

<table>
<thead>
<tr>
<th>method name</th>
<th>metadata parts related rule</th>
<th>content parts related rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>CatalogArchiveStorage#fetchRecord()</td>
<td>applicable for metaToBeFetched meta-parts are attached</td>
<td>content info (excluding data itself) is attached when global attachContentInfoWhenFetchingRecord service flag is set</td>
</tr>
<tr>
<td>CatalogArchiveStorage#refreshRecord()</td>
<td>applicable for metaToBeFetched meta parts are attached to record provided overwriting existing entries if any found</td>
<td>no content parts are attached to record provided as parameter</td>
</tr>
<tr>
<td>CatalogArchiveStorage#iterateRecords()</td>
<td>applicable for metaToBeFetched meta-parts are attached</td>
<td>no content parts are returned</td>
</tr>
<tr>
<td>CatalogArchiveStorageWriter#removeRecord()</td>
<td>all meta parts are removed</td>
<td>all content parts are removed</td>
</tr>
<tr>
<td>CatalogArchiveStorageWriter#saveRecord()</td>
<td>all meta parts are saved, existing entries are overridden; all applicable partsToDrop are removed</td>
<td>all content parts are saved (data location stored within targetFileName field), existing entries are overridden; all applicable partsToDrop are removed</td>
</tr>
</tbody>
</table>
3.9. Dependencies and prerequisites

The current Store Service implementation is configured to use PostgreSQL database therefore, it requires to deploy such database on the server running the Store Service. When needed, the service can be configured in order to use sqlite or other database engine although PostgreSQL is currently recommended.

Service obeys the EuDML guidelines and utilizes slf4j as the main logging mechanism.

Quartz library is required to perform periodical cleanup of unused temporary files which are no longer referenced by weak-refs.

3.10. API description

Storage service offers two basic interfaces:

- StorageService
- StorageServiceWriter

Basic interface, StorageService allows read only access to the Storage. Writer extends the StorageService and it allows storing data as well.

Storage service stores records and each has its own EuDML identifier, (eID). Within records there are multiple parts, each identified by partId (String).

Storage service may be queried for part directly using record eID and partId.

StorageService may be queried for record by identifier. Within the query also metadata parts may be fetched, if proper metadata file category is specified (MetadataPartTypes enum). Then, desired Record instance is returned. Record contains required metadata in text format. If more data is required, the record may be updated by the service to fetch more data.

Note that content files are stored always separately.

3.11. Important file/class/package locations

Store service both configuration files and java classes are located in the eu.eudml.service.storage package, which is part of services-impl module.

3.12. Configuration

The current Storage Service implementation is based on YADDA MD-Storage and Archive Services, hence most of all the required properties are related to these two:

<table>
<thead>
<tr>
<th>property name</th>
<th>default value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jetty.port</td>
<td>8080</td>
<td>jetty related property describing port number</td>
</tr>
<tr>
<td>eudml.db.url</td>
<td>jdbc:postgresql://localhost/eudmlrepo</td>
<td>eudml database url to be shared by MD-Storage and archive YADDA services</td>
</tr>
<tr>
<td>eudml.db.username</td>
<td>eudml</td>
<td>eudml database username to be used by MD-Storage and archive YADDA services</td>
</tr>
<tr>
<td>eudml.db.password</td>
<td></td>
<td>eudml database password to be used by MD-Storage and archive YADDA services</td>
</tr>
<tr>
<td>eudml.data.dir</td>
<td>/tmp/eudml</td>
<td>data filesystem storage directory, used by archive service to store large content</td>
</tr>
<tr>
<td>yadda.archive.contentServerRoot</td>
<td><a href="http://localhost:=$%7Bjetty.port%7D/eudml-repo/part/">http://localhost:=${jetty.port}/eudml-repo/part/</a></td>
<td>archive content server root url to be used by archive to provide direct URL links to stored data</td>
</tr>
</tbody>
</table>
3.13. Deployment

The service can be deployed in any servlet container, in particular running Maven embedded jetty instance by executing:

```
mvn jetty:run
```

Currently store service is part of `eudml-backend` module and it is configured to be available for internal use only.

3.14. Possible extensions

There aren’t supported extensions that worth to be mentioned in this paragraph.
4. Processing Service

Processing framework is based on Spring Integration framework. Processing framework is used for reading entities, processing them and finally write to some place (mainly in persistence layer but it is not a requirement). Processing flow can be a directed tree. Flow mechanism is based on passing messages between nodes. Every message is processed in separate thread, taken from predefined thread-pool.

4.1. Module responsibility

In EuDML project processing framework is used for building relations from XML data (in NLM format) stored in EuDML Store. These relations are views for browsing items, indexing data for search service, clustering data (matching references and author distinction). Without using this framework it is not possible to display and search data in EuDML project.

4.2. Non-functional requirements met by the module

- Spring integration allows multiple threads to run concurrently on single node therefore all defined processing and writing nodes have to be thread-safe.
- Messages passed between nodes must be immutable.

4.3. Implementation details

Processing framework is intended to manage flow of messages between nodes. It takes care of:

- progress monitoring
- smoothly interrupting ongoing process
- error handling - you can define critical and non-critical exceptions. When non-critical error occurs during processing an element, it is registered and the process continues. Regarding critical exception, after registering the exception, the process is interrupted. It is possible to define threshold for non-critical errors. After it is reached, the process terminates.
- concurrency management - for process service, there is defined a global thread-pool which manages threads used during processing.

4.4. Core modules description

4.4.1. ProcessService and ProcessFacade interfaces

ProcessService interface is designated to start, interrupt processes and to obtain statistics from processes. It deals with ProcessRequestS and ProcessResponseS so it can be used as local or remote service.

ProcessFacade simplifies usage of ProcessService.
4.4.2. Implementations


4.4.3. IErrorHandler module description

Every exception occurred during processing flow is wrapped into MessagingException and send to ErrorChannel. MessagingException is internal SI class, additionally more specific AggregatedMessagingException class (which extends MessagingException class) is introduced by yadda-process to support exceptions occurred during writing chunk of data. Moreover there are several kinds of exceptions occurring not during processing flow e.g. EventListenerException which is thrown when there are problems with notification of IMessageRegistryListenerS or MessageRegistryException thrown when MessageResistry.

All exceptions wrapped into MessagingException are handled by the following method:

```java
/**
 * Handle errors occurred during flow.
 * Errors are supposed to be wrapped into SI error message.
 * @param message
 */
public void handleError(ErrorMessage message) throws ErrorHandlerException;
```

Exceptions during processing messages can be critical or non-critical. All exceptions are registered but with non-critical exceptions processing continues. Processing is terminated if a critical exception occurs or the number of non-critical exceptions exceeds the predefined threshold.

Other exceptions can be handled by the following method:

```java
/**
 * Handles exception occurred outside main flow.
 * @param procId
 * @param e
 */
public void handleException(String procId, Exception e);
```

Other exceptions (that don’t occur during processing elements) can also be critical or not-critical. The general rule of thumb is: if the exception is not-critical, the service can recover from it by handling it with IErrorHandler#handleException method. Sometimes critical errors also can be handled by this method before process terminates.

4.4.4. Checking errors fatality

Errors can be considered fatal or non-fatal. Checking of fatality of errors is performed by FatalityExceptionAdapterS. This is an interface with only one method:

```java
/**
 * checking if Throwable is fatal and processing should interrupt
 * @param t
 * @return
 */
boolean isFatal(Throwable t);
```

FatalityExceptionAdapterS placed in parent context and child (process-specific) context are aggregated by AggregatedFatalityExceptionAdapter and checking is
performed until error is fatal or until the end of rules is reached. So, the rule is the "first wins", because the implementation of the order of checking cannot be guaranteed.

At this time, ListBasedFatalityExceptionAdapter class is provided, which checks fatality over a fixed list.

In particular, InterruptProcessingException is always considered as fatal and can be used to "poison" processing.

4.4.5. errors threshold mechanism

In order to prevent some process "bad behavior" (e.g. all processed messages are processed with errors and hence processing last long with no effects), errors threshold is provided. After reaching it process terminates.

There are three parameters you can set (in brackets are the default values):

- ERRORS_THRESHOLD_AS_VALUE (0)
- ERRORS_THRESHOLD_AS_PERCENTAGE (10)
- ERRORS_THRESHOLD_AS_PERCENTAGE_TRIGGER (1000)

You can set them in three places:

- in Spring configuration as ProcessManager properties
- in Spring process flow definition using proc:properties element
- as runtime parameters

The mechanism works as follows:

- Runtime parameters have precedence over proc:properties values and parameters defined in the latter are considered more important than ProcessManager default properties.
- ERRORS_THRESHOLD_AS_VALUE and ERRORS_THRESHOLD_AS_PERCENTAGE should be used exclusively. If both are provided (at the same configuration stage e.g. as runtime parameters) ERRORS_THRESHOLD_AS_VALUE has precedence.
- if ERRORS_THRESHOLD_AS_VALUE is active, then processing is interrupted if errors number reach (>= comparison) ERRORS_THRESHOLD_AS_VALUE.
- if ERRORS_THRESHOLD_AS_PERCENTAGE is active behavior depends on our knowledge of total messages to be processed. If this value is known, the errors occurred are compared with the percentage of total messages that are provided for processing. Otherwise, we compare it with the percentage of messages processed so far (in order to avoid fluctuation, on the beginning of process this comparison starts when the number of processed elements exceeds ERRORS_THRESHOLD_AS_PERCENTAGE_TRIGGER)

4.4.6. implementations

IErrorHandler is implemented by ProcessManager.

4.5. Gathering statistics description

Statistics for process are gathered by ProcessStatsEntry.
4.6. implementations

ProcessManager map Map<String, ProcessingStatsEntry> is responsible for holding statistics for all processes.

4.7. Source module description

Source module is responsible for feeding the flow with elements to be processed. It contains an iterator which does this work. It sends elements to the channel where processing starts and also sends orchestration messages to be processed by the flow.

4.8. MessageRegistry module description

MessageRegistry module was introduced in order to handle messages being processed within infrastructure, in particular:

- regular messages creation, including supplementing headers with all required attributes;
- orchestration messages creation;
- maintaining number of created message instances;
- sending events to registered event consumers;

4.9. regular messages creation

MessageRegistry creates each message based on id and payload provided in request. In addition, attaches shared processing context, sequence number and error channel for each message. Creates an entry in message registry and monitors number of created instances until all instances are consumed which triggers proper event creation. Unconsumed messages threshold value can be overridden (default value is set to 10 000) in order to impose limit on unconsumed messages being held in memory at the same time. When exceeded, an exception will be thrown.

4.10. orchestration messages creation

Each one of the orchestration messages:

- PROCESS_STARTING
- PROCESS_FINISHING
- PROCESS_INTERRUPTING

is supplemented with shared processing context and error channel. Each orchestration message will be created only once for a process. Each instance is monitored in registry until final consumption, when proper event is generated.

4.10.1. maintaining number of created message instances

Both regular and orchestration messages instances are being registered and the number of instances is noted. It should be increased by router modules when single message can be cloned and sent to multiple output channels and consumed on writers, which is performed by node wrappers instantiated automatically during flow initialization. When message is consumed, proper event is generated according to the message type.
4.10.2. sending events to registered event consumers

Four kinds of events can be generated by MessageRegistry:

- **PROCESS_STARTED**
- **PROCESS_FINISHED**
- **PROCESS_INTERRUPTED**
- **MESSAGE_CONSUMED**

Each one is generated when all instances of a given message are consumed. Process finished will be sent only when all instances of **PROCESS_FINISHING** orchestration message are consumed and all regular messages are consumed as well. Event listeners have to implement IMessageRegistryListener interface and be either preregistered in event dispatcher or register itself dynamically using IEventDispatcher API. The following IEventDispatcher implementations are available:

- **DirectEventDispatcher** - which sends events directly, in blocking mode, using caller's thread;
- **AsynchronousEventDispatcher** - which utilizes asynchronous task executor and sends events in asynchronous mode using blocking queue as intermediate;

*Additional notes: to improve MR performance write-locks usage is limited. Therefore, after processing is finished, an entry remains in registry and has to asynchronously be removed by cleanup mechanism (e.g. using quartz mechanism).*

4.11. Flow definition

4.11.1. orchestration description

Before starting and after finishing/interrupting processing elements some actions are taken to put IStatsCollector, IMessageRegistry and processing, writing nodes in appropriate state.

4.11.2. starting orchestration

When there is request to start process, information about it is passing to ISourceNode (1) and further to IMessageRegistry (2) where, among other actions, special orchestration message **PROCESS_STARTING** is created. Then execution goes back to ISource (3) and orchestration message is sent to processing flow to (possibly) initialize flow's nodes. After message is consumed in all branches of flow (6), IMessageRegistry through IEventDispatcher sends the event **PROCESS_STARTED**(7) to all (interested in this type of event) IMessageRegistryListenerS. In particular ProcessingStatsEntry for started process is created and ProcessManager sends information to ISource to start iteration with elements to be processed.

4.11.3. finishing/interrupting orchestration

Finishing/interrupting orchestration flow is similar to starting orchestration flow. In this orchestration flow, the message **PROCESS_FINISHING** is sent after the iterator is exhausted, and the message **PROCESS_INTERRUPTING** after receiving call from ProcessService. In the second case, iterating over source elements is smoothly stopped. The main difference between interrupting and finishing process is that finishing is waiting for each thread processing
node to end its work before the finalizing node, interrupting does not wait. After messages 
PROCESS_FINISHING, PROCESS_INTERRUPTING are consumed by 
MessageRegistry events PROCESS_FINISHED, PROCESS_INTERRUPTED are 
generated respectively.

4.11.4. role of FlowSyncPanel

FlowSyncPanel holds information on if processing, writing nodes have been 
initialized, finalized or interrupted and on how many threads are operating on a given node. It is 
located in the node wrapper, so it is unique for node regarding process execution. It is used 
mainly in the following situations:

- nodes not initialized will not be finalized
- nodes interrupted will not process elements after interruption
- nodes which are busy in some another threads will not be finalized during process 
  finishing until all threads finish their work

4.12. node wrappers functionality

Each defined node, including all three: source, processor and writer, is automatically 
encapsulated within orchestrable wrapper. There are a number of operations handled by wrapper 
which depends on nested node type:

- understanding spring-integration message based model, extracting payloads and passing 
  them to processor/writer;
- autowiring MessageRegistry for message creation (source), multiplication 
  (recipient-list-router) and consumption (writer);
- orchestration handling by executing proper initialization/finalization methods on nodes 
  implementing IInitializableFinalizableNode interface;
- flow synchronization by applying processing rules:
  - monitoring number of concurrently executing threads;
  - marking node which was successfully started/finished/interrupted
  - allowing finalization only when:
    - node was started;
    - all processing operations were finished (waiting until all concurrent 
      operations are finished);
    - node was not either interrupted or finished before;
  - discarding received processing results whenever node was interrupted in the 
    meantime;
  - flushing cache in writers on node finalization/interruption

4.13. custom namespaces definition

In order to simplify flow definition dedicated namespace 
http://yadda.icm.edu.pl/schema/processing was introduced. It defines schema 
for the following elements:

- chain
- recipient-list-router
- source
- processor
- writer
4.13.1. chain

Extension of spring-integration chain element, that allows custom yadda-process elements to be nested inside. Supports very same attributes as the SI chain element:

<table>
<thead>
<tr>
<th>attribute</th>
<th>mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>no</td>
<td>chain identifier</td>
</tr>
<tr>
<td>input-channel</td>
<td>yes</td>
<td>chain's input channel</td>
</tr>
<tr>
<td>output-channel</td>
<td>yes</td>
<td>chain's output channel</td>
</tr>
<tr>
<td>order</td>
<td>no</td>
<td>invocation order when connected as a subscriber to a channel</td>
</tr>
<tr>
<td>auto-startup</td>
<td>no</td>
<td>true by default, indicates whether endpoint should start automatically upon initialization</td>
</tr>
</tbody>
</table>

Supports same nested elements as the SI chain element, supplemented with yadda-process specific elements such as:

- chain
- recipient-list-router
- processor
- writer

4.13.2. recipient-list-router

Extension of spring-integration recipient-list-router element supplemented with handling MessageRegistry communication for message instances counter incrementation. Supports the very same attributes as the SI recipient-list-router element supplemented with cloning related attributes mutable-input and cloner-ref:

<table>
<thead>
<tr>
<th>attribute</th>
<th>mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>no</td>
<td>router identifier</td>
</tr>
<tr>
<td>ignore-send-failures</td>
<td>no</td>
<td>set to false by default, if enabled does not throw exception when message not sent</td>
</tr>
<tr>
<td>apply-sequence</td>
<td>no</td>
<td>set to false by default, if enabled attaches sequence-size, sequence-number and correlation id to the message headers</td>
</tr>
<tr>
<td>timeout</td>
<td>no</td>
<td>maximum timeout when sending messages to recipients</td>
</tr>
<tr>
<td>input-channel</td>
<td>yes</td>
<td>router input channel</td>
</tr>
<tr>
<td>order</td>
<td>no</td>
<td>invocation order when connected as a subscriber to a channel</td>
</tr>
<tr>
<td>auto-startup</td>
<td>no</td>
<td>true by default, indicates whether endpoint should start automatically upon initialization</td>
</tr>
<tr>
<td>mutable-input</td>
<td>no</td>
<td>statically defines whether elements entering router are expected to be mutable or immutable. When undefined objects mutability status will be inferred. Especially useful when flow designer can assure mutability status which will positively influence processing performance.</td>
</tr>
<tr>
<td>cloner-ref</td>
<td>no</td>
<td>reference to cloner module which should be used to perform cloning of entry objects. When not defined default cloner module (defined in root context) will be used when necessary.</td>
</tr>
</tbody>
</table>
Supported nested elements are the same as for the SI recipient-list-router:

<table>
<thead>
<tr>
<th>element</th>
<th>restrictions</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipient</td>
<td>minOccurs=&quot;1&quot;</td>
<td>list of recipient channels</td>
</tr>
</tbody>
</table>

### 4.13.3. source

Source element encapsulating iterator builder providing data to be processed. Iterator builder can be provided either as reference using `iterator-builder` attribute or as nested `<bean>` element inside `source`. The following attributes are supported:

<table>
<thead>
<tr>
<th>attribute</th>
<th>mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>no</td>
<td>source identifier</td>
</tr>
<tr>
<td>channel</td>
<td>yes</td>
<td>source output channel</td>
</tr>
<tr>
<td>iterator-builder</td>
<td>no</td>
<td>to be used interchangeably with nested <code>&lt;bean&gt;</code> element</td>
</tr>
<tr>
<td>task-executor</td>
<td>yes</td>
<td>asynchronous task executor reference to be used for iteration</td>
</tr>
</tbody>
</table>

Supported nested elements:

<table>
<thead>
<tr>
<th>element</th>
<th>restrictions</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bean</td>
<td>can defined</td>
<td>interchangeably with <code>iterator-builder</code> attribute</td>
</tr>
</tbody>
</table>

### 4.13.4. processor

Processor is partially based on transformer element and supports the following attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>No</td>
<td>source identifier</td>
</tr>
<tr>
<td>ref</td>
<td>No</td>
<td><code>IProcessingNode</code> instance reference, to be used interchangeably with nested <code>&lt;bean&gt;</code> element</td>
</tr>
<tr>
<td>input-channel</td>
<td>No</td>
<td>processor's input channel, not required when defined within chain element</td>
</tr>
<tr>
<td>output-channel</td>
<td>No</td>
<td>processor's output channel, not required when defined within chain element</td>
</tr>
<tr>
<td>send-timeout</td>
<td>No</td>
<td>processor's sending timeout, set to -1 (unlimited) by default</td>
</tr>
</tbody>
</table>

Supported nested elements:

<table>
<thead>
<tr>
<th>element</th>
<th>Restrictions</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bean</td>
<td>can be defined interchangeably with <code>ref</code> attribute</td>
<td><code>IProcessingNode</code> instance.</td>
</tr>
<tr>
<td>si:poller</td>
<td>element from spring-integration namespace; not required, when undefined default poller will be used</td>
<td>poller to be used for retrieving data from asynchronous channel</td>
</tr>
</tbody>
</table>
4.13.5. \textbf{writer}

Writer element is based on channel-adapter SI type and one of its main features is \textit{MessageRegistry} communication required for message \textit{consumption}. The following attributes are supported:

<table>
<thead>
<tr>
<th>attribute</th>
<th>mandatory</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>no</td>
<td>writer identifier</td>
</tr>
<tr>
<td>channel</td>
<td>yes</td>
<td>writer input channel</td>
</tr>
<tr>
<td>packageSize</td>
<td>no</td>
<td>utilized only when \textit{ICollectionWriterNode} is used, defined single package size being passed to collection writer.</td>
</tr>
</tbody>
</table>

Supported nested elements:

<table>
<thead>
<tr>
<th>element</th>
<th>restrictions</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bean</td>
<td>can defined interchangeably with \textit{ref} attribute</td>
<td>either \textit{IWriterNode} or \textit{ICollectionWriterNode} instance.</td>
</tr>
<tr>
<td>si:poller</td>
<td>element from spring-integration namespace; not required, when undefined default poller will be used</td>
<td>poller to be used for retrieving data from asynchronous channel</td>
</tr>
</tbody>
</table>

4.13.6. \textbf{Properties}

This is an optional element allowing default properties definition for particular process. It can be defined as one of the flow definition elements. This element is based on Spring \textit{util:map} therefore all nested elements such as \textit{<entry>}, \textit{<key>}, \textit{<value>} with all supported attributes can be used. All defined properties are stored within \texttt{ProcessContext#auxParams} and the following requirements are undertaken:

- only \texttt{String} keys are allowed;
- only \texttt{Serializable} values are allowed;

When client defines auxiliary property in request then default value defined within \textit{proc:properties} is discarded.

4.14. \textbf{Dependencies and prerequisites}

No dependencies and prerequisites.

4.15. \textbf{API description}

4.15.1. \textbf{Sketch of the API}

API contains interfaces of source, processing and writing nodes that make complete set of blocks to define processing workflow.
4.15.2. Nodes APIs

4.15.2.1 Stateful nodes

All defined nodes can be either stateful or stateless. When node is considered to be stateful it should implement \texttt{IInitializableFinalizableNode} exposing the following methods:

```java
/**
 * Initializes processing.
 * @param ctx
 * @throws Exception
 */
public void initialize(ProcessContext ctx) throws Exception;

/**
 * Finalizes processing.
 * @param ctx
 * @throws Exception
 */
public void finalize(ProcessContext ctx) throws Exception;
```

Those methods will be executed at flow initialization and finalization phases accordingly. \texttt{IInitializableFinalizableNode#finalize(ctx)} method will be executed also when processing is interrupted. \texttt{ProcessContext} conveys process identifier and simple auxiliary parameters. Each of those methods will be executed only once per process which is guaranteed by design. Whenever an exception is thrown, it is carried by orchestration message and propagated to ErrorHandler when writer node is reached.

4.15.2.2 Source node

Source node is wrapping layer encapsulating \texttt{ISourceIteratorBuilder} which currently exposes two methods:

```java
/**
 * Creates \texttt{ISourceIterator} based on parameters given in \texttt{ProcessContext}.
 * @param ctx
 * @return \texttt{ISourceIterator} based on parameters given in \texttt{ProcessContext}
 */
public ISourceIterator<E> build(ProcessContext ctx);

/**
 * Returns identifier extractor.
 * @return identifier extractor
 */
public IIdExtractor<E> getIdExtractor();
```

\texttt{ISourceIterator} is an iterator instance providing elements which should propagated for processing. It extends generic \texttt{Iterator} by providing additional methods:

```java
/**
 * Returns estimated number of elements to be processed.
 */
public int getEstimatedSize();

/**
 * Method for releasing resources where iteration is interrupted.
 */
public void clean();
```
The first one returns estimated total number of elements which will be returned by iterator, the second one allows performing cleanup operation whenever process is interrupted. IIdExtractor is utilized for identifier extraction from elements returned by iterator. It is required by yadda-process for messages creation due to its generic nature.

4.15.2.3 Processing nodes

Processing nodes should implement IProcessingNode which is quite simple and exposes single method:

```java
/**
 * Performs processing operations.
 * @param input
 * @param ctx
 * @return output object
 * @throws Exception
 */
public Object process(I input, ProcessContext ctx) throws Exception;
```

Each exception thrown is sent along with message causing exception to the ErrorHandler. Processing path is interrupted at this point.

Notice: processing nodes should be thread-safe because they can be executed concurrently by multiple threads!

4.15.2.4 Writing nodes

Two interfaces can be used when defining writer node:

- IWriterNode
- ICollectionWriterNode

The first one is dedicated to single element consumers and it exposes single method:

```java
/**
 * Stores given data element.
 * @param data
 * @param ctx
 * @throws Exception
 */
public void store(I data, ProcessContext ctx) throws Exception;
```

While the second one accepts the collection:

```java
/**
 * Stores given collection of data.
 * @param data
 * @param ctx
 * @throws Exception
 */
public void store(Collection data, ProcessContext ctx) throws Exception;
```

When the latter is used, WriterNodeWrapper caches all received elements and sends them to ICollectionWriterNode when packageSize limit is reached.

Notice: writing nodes should be thread-safe because they can be executed concurrently by multiple threads!
4.16. Usage example

Example of defining a process workflow:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:sil="http://www.springframework.org/schema/integration"
      xmlns:stream="http://www.springframework.org/schema/integration/stream"
      xmlns:proc="http://yadda.icm.edu.pl/schema/processing"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans.xsd
                          http://www.springframework.org/schema/context
                          http://www.springframework.org/schema/context/spring-context-2.5.xsd
                          http://www.springframework.org/schema/integration
                          http://www.springframework.org/schema/integration/spring-integration-1.0.xsd
                          http://www.springframework.org/schema/integration/stream
                          http://www.springframework.org/schema/integration/stream/spring-integration-stream-1.0.xsd
                          http://yadda.icm.edu.pl/schema/processing
                          http://yadda.icm.edu.pl/schema/processing/yadda-process.xsd">

  <!-- optional process parameters: FROM_DATE - optional date which denotes
      start of iteration TO_DATE - optional date which denotes end of
      iteration META_TO_BE_FETCHED - optional meta parts to be fetched
      during iteration, overwrites default bean value, String or String[]
      expected disableAggregations - [browse] markViewsAsReady - [browse]
      lazyMaterializeViews- [browse] disable_index_fulltext - [index]
  -->

  <!-- required only when message registry bean is named differently than
      "messageRegistry"
  -->
  <context:annotation-config />

  <bean id="dateRangeItemRecordIteratorBuilder"
        class="eu.eudml.process.source.DateRangeItemRecordIteratorBuilder">
    <property name="eudmlStorage" ref="storage" />
    <property name="metaToBeFetched">
      <list>
        <value>NLM</value>
      </list>
    </property>
  </bean>

  <!-- beans for YElementIndexWriterNode-->
  <bean id="yElementToSElementConverter"
        class="pl.edu.icm.yadda.process.search.YElementToSElementConverter">
    <property name="categoryService" ref="categoryService" />
  </bean>

  <bean id="searchObjectConverter"
        class="pl.edu.icm.yadda.process.bwmeta.index.SObject2IndexDocumentConverter">
    <property name="priorities">
      <value>
        all.name = 15
        all.description = 1
        all.author = 4
        all.coauthor = 3
        all.contributor = 2
        all.identifier = 1
        all.keyword = 3
        all.ancestor = 1
        name.defaultName = 3
        contributor.author = 3
        contributor.coauthor = 2
      </value>
    </property>
    <property name="boostModifier" ref="mainBoostModifier" />
  </bean>

  <bean name="mainBoostModifier" />
</beans>
```
<property name="boostModifiers">
  <list>
    <bean class="pl.edu.icm.yadda.process.bwmeta.index.boost.EmptyBoostModifier">
      <property name="boost" value="0.002" />
      <property name="field" value="authorCoauthor" />
    </bean>
    <bean class="pl.edu.icm.yadda.process.bwmeta.index.boost.FieldBoostModifier">
      <property name="boost" value="0.05" />
      <property name="field" value="name" />
      <property name="containsList">
        <list>
          <value>Author index</value>
          <value>Subject index</value>
          <value>Table of contents</value>
          <value>Meeting information</value>
          <value>Annual index</value>
          <value>Index to advertisers</value>
          <value>Abstracts</value>
          <value>Keyword Index</value>
          <value>Master Index</value>
          <value>Master listing</value>
          <value>Master keyword index</value>
          <value>Cumulative Indexes</value>
          <value>Cumulative Subject Index</value>
        </list>
      </property>
    </bean>
    <bean class="pl.edu.icm.yadda.process.bwmeta.index.boost.FieldBoostModifier">
      <property name="boost" value="0.5" />
      <property name="field" value="journalName" />
      <property name="containsList">
        <list>
          <value>Fuel and Energy Abstracts</value>
        </list>
      </property>
    </bean>
  </list>
</property>

<bean id="simpleIndexResolver" class="pl.edu.icm.yadda.process.bwmeta.index.SimpleIndexResolver">
  <property name="mainIndexName" value="index.eudml" />
</bean>

<bean id="refIndexResolver" class="pl.edu.icm.yadda.process.bwmeta.index.SimpleIndexResolver">
  <property name="mainIndexName" value="index.refs" />
</bean>

<si:channel id="channel1">
  <si:queue capacity="200" />
</si:channel>

<si:channel id="channel2">
  <si:queue capacity="200" />
</si:channel>

<si:channel id="channel3">
  <si:queue capacity="200" />
</si:channel>

<si:channel id="channel4">
  <si:queue capacity="200" />
</si:channel>
D4.2 – EuDML global system functional specification and design

<si:channel id="channel5">
  <si:queue capacity="200" />
</si:channel>

<si:channel id="channel6">
  <si:queue capacity="200" />
</si:channel>

<si:channel id="channel7">
  <si:queue capacity="200" />
</si:channel>

<proc:source id="someSource" channel="channel1" iterator-builder="dateRangeItemRecordIteratorBuilder">
</proc:source>

<proc:chain id="mainFlow" input-channel="channel1">
  <proc:processor>
    <bean class="eu.eudml.process.nodes.ItemRecordToYElementConverterNode">
      <property name="levelsToInclude">
        <list>
          <value>bwmetal.level.hierarchy_Journal_Article</value>
          <value>bwmetal.level.hierarchy_Journal_Number</value>
          <value>bwmetal.level.hierarchy_Journal_Volume</value>
          <value>bwmetal.level.hierarchy_Journal_Year</value>
          <value>bwmetal.level.hierarchy_Journal_Journal</value>
        </list>
      </property>
    </bean>
  </proc:processor>
  <proc:processor>
    <bean class="pl.edu.icm.yadda.process.node.NameTrimmerNode">
      <property name="maxLength" value="125" />
    </bean>
  </proc:processor>
  <proc:recipient-list-router mutable-input="false">
    <proc:recipient channel="channel2" />
    <proc:recipient channel="channel3" />
    <proc:recipient channel="channel5" />
    <proc:recipient channel="channel6" />
  </proc:recipient-list-router>
</proc:chain>

<proc:writer channel="channel2" packageSize="20">
  <!-- notice: real package size may reach up to packageSize*entryPayloadsCount -->
  <!-- using array based browser supporting caching of already written elements -->
  <bean class="pl.edu.icm.yadda.process.node.ArrayEntryBasedBrowserViewsWriterNode">
    <property name="levelsToMonitorDuplicates">
      <list>
        <value>bwmetal.level.hierarchy_Journal_Journal</value>
        <value>bwmetal.level.hierarchy_Journal_Number</value>
      </list>
    </property>
  </bean>
</proc:writer>

<bean id="indexingTaskExecutor" class="org.springframework.core.task.SyncTaskExecutor" />

<!-- extracting article element only -->
<proc:processor input-channel="channel3" output-channel="channel4">
  <bean class="eu.eudml.process.nodes.LevelBasedElementFilterNode">
  </bean>
</proc:processor>
4.17. Important file/class/package locations

Configuration files are located in: pl.edu.icm.yadda.process.config package.

4.18. Configuration

Service needs no external configuration.

4.19. Possible extensions

Implement checkpoint manager to resume interrupted processing.
5. User Directory Service

User Directory Service is dedicated to store information about users, groups and relations between them. It can also authenticate user using token/credential mechanism.

Information concerning users is:

- user domain (as User Directory implementation can support multiple domains)
- user identifiers (from different namespaces)
- user attributes
- user flags
- user credentials (used in user authentication process)
- groups to which user belongs

Information concerning groups is:

- group domain
- group name
- group roles
- if group has parent group (from which it inherits roles)

5.1. Module responsibility

Used for storing basic data about users and authentication of users.

5.2. Non-functional requirements met by the module

All operations must be wrapped in transaction.

5.3. Implementation details

Implementation is based on Hibernate ORM framework.

5.4. Dependencies and prerequisites

PostgreSQL database is required to run service.

5.5. API description

OPEN ISSUE: The details of the API of this services still needs to be define, according to the requirements to be expressed during the integration process by the other services clients of this one...

5.6. Users

5.6.1. User and UserData

There are two classes representing users: User and UserData. The first keeps only basic information about user. More advanced information as effective groups, credentials etc. are
included in the latter. As a rule of thumb User is used to store and update data, UserData for retrieving.

5.6.2. Identifiers

There are two kinds of user identifiers:

- **internal identifier**, must be unique in User Directory implementation scope
- **external identifier**, in form `<namespace>:<proper_identifier>`, pair (external identifier, domain) must be unique in User Directory implementation scope

5.6.3. CRUD

Among classical CRUD operations, the ones worth of mentioning are:

```java
void deleteUser(String identifier, String domain) throws UserNotFoundException;
UserData loadUser(String identifier, String domain, UserData.UserDataParts... fetchParameters);
```

When loading or deleting users, firstly internal identifier is tried and, if not successful, a combination of external identifier' and 'domain' is used.

During loading user data, it is possible to specify which information should be retrieved by specifying `fetchParameters`. Possible values are stated in `UserData.UserDataParts` enum, and have self explaining names (at least after reading rest of this page :)):

- EFFECTIVE.Roles
- EFFECTIVE.Groups
- SENSITIVE.Data
- SAFE_SENSITIVE_DATA
- ALL

It is also possible to list users by specifying list of ids (both internal and external) by:

```java
List<UserData> listUsers(List<String> ids, String domain, UserData.UserDataParts... fetchParameters) throws DomainNotSpecifiedException;
```

The result list keeps the order of the specified ids.

Identifiers from given namespace of given domain can be retrived by method:

```java
Set<String> fetchUserIndentifiers(String userId, String namespace);
```

Here identifier must be internal identifier.

5.6.4. Search users

Method for searching users in User Directory is:

```java
PaginationResult<UserData> searchUsers(String domain, Set<String> groups, Set<String> roles,
Map<String, String> attributes, Set<String> flags, int firstResult, int maxResults,
UserData.UserDataParts... fetchParameters) throws DomainNotSpecifiedException;
```

Semantics is as follows: all conditions are connected using OR conjunction, except attributes where between them AND is used. Result can be paginated.
5.6.5. Credentials and tokens

Credentials and tokens are used in user authentication process. Security token is passed during authentication process to service and there token is validating against credential to authenticate user.

Credentials can be loaded with `UserData` objects with `fetchParameters` equals to `SENSITIVE_DATA` or `SAFE_SENSITIVE_DATA`. The latter hides critical data e.g. when credential is used in login/password scenario password is not exposed.

5.7. Groups

5.7.1. Identifiers

Similarly to users, groups have two kinds of identifiers:

- **external identifiers** - must be unique in User Directory implementation scope
- **internal identifiers** - represented by class `GroupName` containing group name and group domain, must be unique in User Directory implementation scope

5.7.2. CRUD

All CRUD operations use internal identifiers.

5.7.3. Additional group-related methods

To get all users belonging to group, the following paginated method is used:

```java
PaginationResult<UserData> fetchGroupUsers(GroupName groupName, int firstResult, int maxResults, UserData.UserDataParts... fetchParameters)
throws GroupNotFoundException;
```

It is also possible to get all groups that are member of a given group (i.e. for which a given group is parent group):

```java
Set<Group> fetchChildGroups(GroupName groupName) throws GroupNotFoundException;
```

5.7.4. Relations between users and groups

A user can be a member of a group. Also a group can be a member of a group. User Directory methods for governing these relations are:

```java
void assignUser(String userId, GroupName group) throws UserNotFoundException,
GroupNameNotFoundException,
CrossDomainOperationException;
void unassignUser(String userId, GroupName group) throws UserNotFoundException,
GroupNameNotFoundException,
CrossDomainOperationException;
void assignGroup(GroupName group, GroupName parent) throws GroupNotFoundException,
GroupNameNotFoundException,
GroupAssignmentException,
CrossDomainOperationException;
void unassignGroup(GroupName group, GroupName parent) throws GroupNotFoundException,
GroupNameNotFoundException,
CrossDomainOperationException;
```

It is easily visible that hierarchy of groups (in respect to parent-child relation) constitutes a forest.
5.7.5. Effective groups and roles
If a user belongs to a group, it inherits all the group's roles. The same is true for a group. If a user
is member of a group which has parent group, it also is member of the parent group.
This assumption allows speaking about effective groups and roles, which means all group and
roles that a user (or group) inherits from the relation discussed above.

5.8. Pagination
Some methods (which are supposed to return big sets of objects with data e.g. users during
search) take as arguments two integer parameters firstResult and maxResults and return
PaginationResult<T> object.
Parameter firstResult states for the first returned element from whole list of results,
whereas maxResults is max number of returned objects. If maxResults is set to integer
lesser or equal zero then all results are returned.
Class PaginationResult<T> keeps data objects as protected List<T> results
filed and info about total number of objects as protected int totalCount field.

5.9. Configuration
Model configuration file is:

```xml
<yadda-user/src/test/resources/pl/edu/icm/yadda/service2/user/hibernate/usercatalog-beans-
test.xml
```
6. Search Service

The Search service is used to store any kind of information about documents in the form of a 'set of fields' - title, abstract, publication date, ISSN number... Then it allows to search over all these documents (using special query) and retrieve relevant documents and any of its stored fields values.

The Search service is based on Apache Solr - the open source enterprise search platform. It serves as a simplified outer layer (completely hides Solr's API and configuration – see Figure 12) which provides:

- simple Java API for document indexing and searching
- simple XML (and Java) index metadata configuration (index fields declaration)
- fully automatic Solr instance configuration and initialization

6.1. Module responsibility

The Search Service comprises:

- document indexer - stores documents metadata (title, authors, abstract...) and full-texts in the index
- document searcher - provides API to searching in the index for documents similar to specified query (using provided java query model)
- index management - e.g. index configuration, initialization.

6.2. Non-functional requirements met by the module

None.

6.3. Implementation details

- All requests come to the search service (yadda-solr) through the public API - implementation of yadda-services2 interfaces using yadda-model model.
- Then they are modified according to configuration and mapped to more complex Solr requests.
• Solr performs requested actions on Lucene index (optionally customized in plug-ins from \textit{yadda-solr-plugin}) and returns the result back to the search service.

Solr is currently set up using:

• SolrJ java client
• EmbeddedSolrServer implementation (Solr server embedded in the application and not run within separate Servlet Container)
• some custom plug-ins (e.g. math formulae plug-in)
• configuration - predefined and dynamically created

6.4. Search service details

We can distinguish 5 main parts of search service (yadda-solr project):

• **API** (\texttt{pl.edu.icm.yadda.service2.search.*} which hides more internal implementation of \texttt{pl.edu.icm.yadda.search.Searcher[Updater]}) - public interface
• **Solr servers** (\texttt{pl.edu.icm.yadda.search.solr.index.*}) - implementations/connections to Solr server instances
• **Servers management** (\texttt{pl.edu.icm.yadda.search.solr.manage.*}) - Manages Solr servers instances/connections
• **Configuration** (\texttt{pl.edu.icm.yadda.search.solr.model.index.*}) - Handles indexes configurations and schemas (metadata)
• **Mapper** (\texttt{pl.edu.icm.yadda.search.solr.model.mapping.*}) - Transforms API requests to/from Solr requests

Additionally there are **plug-ins** (yadda-solr-plugin project) which are used to customize internal Solr/Lucene request processing.

6.5. Dependencies and prerequisites

Search service depends on two projects:

• **yadda-services2** - providing public interfaces
• **yadda-model** - providing model classes used by public interface
6.6. API description

6.6.1. Sketch of the API

The public API for indexing is based on (Figure 14):

- **add/delete document**
- **commit/rollback changes**


The public API for searching is based on (Figure 15):

- **search requests** (Figure 16) - sending criteria for searching documents in the index.
- **filter management** - storing the same criteria as in search requests in order to use them later in searching (they will be referred by names).
In search request, one has to specify index name and provide SearchQuery object.

This class is the most important in search service API, because it holds all information about user search criteria:

- `filterName` - the name of filter (kind of optimized subquery) to be used
- `criteria` - the list of SearchCriterion objects. Here are specified all search keywords
- `orders` - requested result sort order policy
- `facet` - special Facet query (which gives us groups of documents (their count) meeting the specified requirements)
- `subqueries` - additional SearchQueries further limiting the results

### 6.6.2. Important file/class/package locations

Most important classes and packages in yadda-solr:

- `pl.edu.icm.yadda.search.solr.SolrSearcher` - entry point for search query handling
- `pl.edu.icm.yadda.search.solr.index.update.SolrIndexSession` - responsible for index update (adding documents to index)
- `pl.edu.icm.yadda.search.solr.index.EmbeddedIndex` - implementation of index with EmbeddedSolrServer
- `pl.edu.icm.yadda.search.solr.manage.SingleIndexManager` - responsible for single index setup
- `pl.edu.icm.yadda.search.solr.manage.DefaultFilterManager` - responsible for filters (often used, optimized search queries) management
- `pl.edu.icm.yadda.search.solr.model.mapping` - contains classes responsible for core requests translation between API and Solr
- `pl.edu.icm.yadda.search.solr.model.index.config.util.ConfigTranslator` - responsible for dynamic configuration (generate Solr files, read configuration files...)
- `pl.edu.icm.yadda.search.solr.model.index.metadata` - classes responsible for schema file transformations
6.7. Configuration

In order to setup search service one has to prepare spring beans configuration and declare for each index:

- **Index name [mandatory]** - name to use for indexing, searching and index management
- **Index path [optional]** - filesystem path to directory, where all configuration and runtime data will be stored. [if omitted: temporary directory will be used]
- **Schema path [optional]** - resources path to file containing index metadata information. [if omitted: default schema file will be used - /yadda-solr/src/main/resources/yadda-schema.xml]

*On first application startup, all necessary configuration files will be automatically generated and Solr server initialized. On all subsequent startups, if the service encounters previously generated files, it will use them for Solr initialization.*

### 6.8. Single index

Sample Spring beans configuration for single index:

```xml
<bean id="exampleIndexManager" class="pl.edu.icm.yadda.search.solrmanage.SingleIndexManager" init-method="initialize">
    <property name="indexName" value="index.example" />
    <property name="indexPath" value="/path/to/index/home/" />
    <property name="inputSchemaPath" value="/example-schema.xml" />
</bean>

<bean id="searcher" class="pl.edu.icm.yadda.search.solr.SolrSearcher">
    <property name="indexManager" ref="exampleIndexManager" />
    <property name="filterManager" ref="filterManager" />
    <property name="mapper" ref="mapper" />
</bean>

<bean id="updater" class="pl.edu.icm.yadda.search.solr.SolrUpdater">
    <property name="indexManager" ref="exampleIndexManager" />
    <property name="mapper" ref="mapper" />
</bean>

<bean id="queryBuilder" class="pl.edu.icm.yadda.search.solr.model.mapping.QueryBuilder" />

<bean id="mapper" class="pl.edu.icm.yadda.search.solrmodel.mapping.Mapper">
    <property name="queryBuilder" ref="queryBuilder" />
</bean>

<bean id="filterManager" class="pl.edu.icm.yadda.search.solrmanage.DefaultFilterManager" />
```
Modify only exampleIndexManager bean to adopt the service to your needs.

### 6.9. Multi index

In order to use the service with more than one index, you have to change the beginning of single index configuration, for example:

```xml
<bean id="mixedIndexManager" class="pl.edu.icm.yadda.search.solr.manage.MixedIndexManager">
    <property name="indexManagers">
        <list>
            <ref bean="eudmlIndexManager" />
            <ref bean="refsIndexManager" />
        </list>
    </property>
</bean>

<bean id="eudmlIndexManager" class="pl.edu.icm.yadda.search.solr.manage.SingleIndexManager"
    init-method="initialize">
    <property name="indexName" value="index.eudml" />
    <property name="indexPath" value="${eudml.search.indexPath}/eudml" />
</bean>

<bean id="refsIndexManager" class="pl.edu.icm.yadda.search.solr.manage.SingleIndexManager"
    init-method="initialize">
    <property name="indexName" value="index.refs" />
    <property name="indexPath" value="${eudml.search.indexPath}/refs" />
</bean>

<bean id="searcher" class="pl.edu.icm.yadda.search.solr.SolrSearcher">
    <property name="indexManager" ref="mixedIndexManager" />
    <property name="filterManager" ref="filterManager" />
    <property name="mapper" ref="mapper" />
</bean>

<bean id="updater" class="pl.edu.icm.yadda.search.solr.SolrUpdater">
    <property name="indexManager" ref="mixedIndexManager" />
    <property name="mapper" ref="mapper" />
</bean>

<!-- the rest of configuration the same as with single index -- >
```

The main difference with respect to single index configuration is the additional MixedIndexManager bean which combines all single index declarations.

### 6.10. Schema file customization

By default (if you do not specify schema file in configuration), the service will use the same schema file (index metadata declaration) for all indexes:

- /yadda-solr/src/main/resources/yadda-schema.xml
If you would like to change it or create your own schema XML file, here is the brief instruction (sample complete schema XML file):

```xml
<schema>
  <templates>
    <template name="template_text">
      <type value="text" />
      <tokenized />
      <exact-searchable />
      <keep-stopwords />
      <short-penalty value="10" />
      <span value="100" />
    </template>
    <template name="template_number">
      <unindexed />
      <single-valued />
      <type value="number" />
    </template>
  </templates>
  <fields>
    <field name="field1" template="template_text" />
    <field name="field2" template="template_text" />
    <field name="field3" template="template_text" />
    <field name="field4" template="template_number" />
  </fields>
</schema>
```

Generally we have two options for fields declaration:

- We can fill only `<fields>` section and directly declare fields properties for all field names
- We can first fill `<templates>` section and then use it in `<fields>` section as predefined fields properties sets.

```xml
<schema>
  <templates>
    <!-- Preconfigured field properties [OPTIONAL] -->
  </templates>
  <fields>
    <!-- Field declarations with properties and/or template reference -->
  </fields>
</schema>
```

Templates structure

```xml
<templates>
  <template name="template1">..properties..</template>
  <template name="template2">..properties..</template>
</templates>
```

Sample template declaration:

```xml
<template name="template1">
  <type value="text" />
  <tokenized />
  <stored />
  <span value="100" />
</template>
```

Fields structure

```xml
<field>
  <field name="field1">..properties..</field>
  <field name="field2" template="_templateName_" />
  <field name="field3" template="_templateName_">..properties..</field>
</field>
```
Fields can be declared in 3 ways:

- With direct properties declaration [field1]

```
<field name="field1">
  <type value="text" />
  <tokenized />  
  <stored />  
  <span value="100" />
</field>
```

- With template reference declaration [field2] ('template2' has to be declared first)

```
<field name="field2" template="template2" />
```

- Mixed – with both, template and direct properties declaration

```
<field name="field1" template="template1">
  <single-valued />
  <span value="90" />
</field>
```

**Note:** If you use this style (mixed), 'final field properties' = 'template properties' + 'direct properties'. If there is a conflict, 'direct property' is used, for example [look at span value]:

```
<schema>
  <templates>
    <template name="template_text">
      <type value="text" />
      <tokenized />
      <span value="100" />
    </template>
  </templates>
  <fields>
    <field name="field" template="template_text">
      <sortable />
      <span value="77" />
    </field>
  </fields>
</schema>
```

Result field property set will be: TYPE="text", INDEXED, TOKENIZED, SORTABLE, SPAN="77"

All acceptable properties:

- `<type value="text|number|date|math" />` - [mandatory] Declare field content type. Currently supported types: "text", "number", "date", "math" (mathematics)
- `<unindexed />` - Field will not be indexed
- `<stored />` - Field will be stored
- `<dynamic-suffix />` - Field will be dynamic: Index will accept any field which name starts with this fields name. **For example:** this fields name="myField". Then any field matching myField* pattern will be accepted during indexing/searching.
- `<sortable />` [Available when type="text"] - Enable query result sorting support over this field. **Note:** You don't have to set this property when type="number" or type="date" (they are always sortable).
- `<tokenized />` [Available when type="text"] - Field will be tokenized. If set, additional properties are available:
D4.2 – EuDML global system functional specification and design

- <keep-stopwords /> - Stopwords (like: 'a', 'or', 'I', 'am'...) will not be abandoned during tokenization
- <simple-tokens /> - Simple tokenization will be used: tokenize on whitespaces + lowercase
- <single-valued /> - Field will permit only one value to be indexed
- <exact-searchable /> - Enable matchWholeField query support
- <short-penalty value="_long_" /> - Lower search result score when this field value has length shorter then given value threshold
- <span value="_long_" /> - Set multiple values position increment gap. If we will try to search multiple words, they will be found only in increment gap range – it means, they will be found within one indexed value. For example: If you index 2 values: ["John W. Smith","Albert Yang"], Query for „John Smith” will be found, but „Smith Albert” won't.

6.11. Possible extensions

In the future it is planned to:

- Add dynamic index management to have more control over index lifecycle
- Keep up with Solr new versions releases
7. Browse Service

The Browse Service is a supporting service responsible for dealing with relations between objects. This service allows defining relations (in a relational database sense) indexed for fast access, and querying the data in these relations. It also allows efficient querying of aggregated data, for example fetching the count of objects fulfilling specific criteria, by maintaining lazily materialized aggregated data views. The Browse Service supports various data types.

7.1. Module responsibility

This service is used to provide effective browsing over data objects within the storage. It is filled by processes and read via middle layer services by the user interface.

7.2. Non-functional requirements met by the module

None.

7.3. Implementation details

The browse service stores the data in a SQL database. It creates several static tables for service metadata, i.e., relation and aggregating view definitions. The service implements the `pl.edu.icm.yadda.bean.Configurable` interface in order to be able to create the tables if they do not yet exist in the database.

The tables and indexes for each browse relation (including additional tables for array fields data) are created when a relation is defined.

Relations are versioned. If the creation of an already existing relation is requested, a new version is created and used by default, but the old one still exists in the database and can be accessed by using an explicit version number.

Null values are not supported. If a null value is used in an update operation, the previous value is left unchanged at this position in the tuple.

The service queries are translated to SQL queries enriched for paging with page boundary conditions stored in opaque cookies passed to and from the client. In order to guarantee consistent paging, the queries always use sorting. If the client does not specify any sorting order an internal tuple identifier is used as the sort key. It is also used as the last resort key added to the sorting order specified by the client.

7.3.1. Aggregation support

It is possible to create aggregating views for a relation. An aggregating view can perform a `count`, `count distinct`, `min`, `max` or `sum` operation on a specified relation field while grouping the data on a set of other fields and selecting a subset of the rows matching a specified condition. Access to the aggregated date may be optimized by using different materialization strategies for the views.

- none - implemented as a regular database view,
- full - implemented as a database table recreated from scratch after each change,
7.4. Dependencies and prerequisites

The browse service implementation is using a SQL database to store the data. PostgreSQL and SQLite are supported, with PostgreSQL recommended and used in EuDML. The service needs to be configured with access to an existing PostgreSQL database and can create all the necessary tables for itself.

7.5. API description

7.5.1. Sketch of the API

All data that can be stored in the browse service is `Serializable`, a row (tuple) in a browse service relation is represented by an array `Serializable[]`. The supported types include:

- short, indexable strings,
- short, indexable case-insensitive strings,
- long, non-indexable strings,
- long integers,
- doubles,
- booleans,
- timestamps,
- arrays of short strings,
- bitsets (for use as licence-sets)

The entry point to the Browse Service facade is through the `IBrowserFacade` interface, which contains operations to

- create a new Browse Service relation,
- access a facade for an existing relation.

The `Relation` interface can then be used to

- retrieve the relation definition,
- add specified data tuples to the relation,
- delete from the relation all tuples matching a condition,
- replace tuples matching a condition,
- replace tuples matching a condition or add new data if none matches,
- count the number of tuples matching a condition,
- select data matching a condition from the relation,
- add/remove an aggregating view,
- count the number of aggregation results matching a condition,
- select data matching a condition from an aggregating view,
- remove the relation from the service.

The `Query`, `Selection` and `Condition` classes provide a fluent interface for convenient creation of select queries.

The relation methods that select data return a `Fetcher` object, which can be used to page through the results by

- retrieving the current result page,
• going to the beginning of the result and fetching the first page,
• going to the end of the result and fetching the last page,
• going forward to a page of the result (possibly skipping a number of tuples) and fetching it,
• going backward to a page of the result (possibly skipping a number of tuples) and fetching it.

The interface for remoting access corresponding to all the facade operations is **IBrowser**.

### 7.5.2. Important file/class/package locations

The browse service facade interfaces and implementation are located in the `pl.edu.icm.yadda.service2.browse.facade` package, and the remoting service interfaces and implementation in the `pl.edu.icm.yadda.service2.browse` package. The query model and support classes are located in the `pl.edu.icm.yadda.service2.browse.query` package.

### 7.6. Configuration

The service is set up for EuDML purposes in the Spring application context configuration in the `src/main/webapp/WEB-INF/application-context/yadda-services.xml` file of the eudml-backend module.

The setup consists of the yaddaBrowse facade bean using the browseService bean as the actual service implementation with help from several supporting beans.

Because the implementation requires edit operations to be transactional, the setup uses Spring transaction support to configure the methods on the **IBrowserBackend** interface to be performed in transactions.

The browse service requires database connection details for operation. The service set up in the EuDML backend application context uses the same configuration properties as the Storage Service for this purpose.
8. Similarity Service

Similarity service detects similarities between text documents. Given a document, either already stored in the service or specified in the request (in the latter case document's text must be contained in the query), is used as a basis of similarity query which finds other similar documents stored earlier in the service. What exactly 'similar' means, depends on the implementation.

8.1. Module responsibility

Operations on similarity service may be divided into two major groups: searching for similar documents and indexing (addition of documents to similarity service).

- **Similarity search.** This is main task of similarity service - to find similar documents to the specified single document. Similarity API provides two ways of specifying document on which similarity search should be performed - a complete document may be passed to search method (in this case document does not have to be indexed before search) or just its id (in this case document with specified id must be already indexed).

- **Indexing.** In order to make any document available in similarity search it must be indexed first. One can think of indexing as a process of addition (removal) of documents to (from) similarity service. It should be noted here that one of the basic concepts of similarity search is division of documents into indexes ('buckets' of documents which group documents with similar features). Indexes are parameterized by language and category name (and these are features of documents which should be detected first in similarity search). To modify specified index one obtains connection to it and operates on that connection to add or remove documents. Indexing API allows also batch deletions by providing two special methods - one drops all documents from the service, another drops all documents from specified index.

**Note on language/category splitting:** Concept of language/category splitting is supposed to separate documents which are not similar regarding to document features known \( a \ priori \). Current similarity implementation uses single dummy category but splits documents with regard to their language (language is detected automatically if not specified). It is possible that other implementations would completely ignore language/category data - the only requirement is to implement methods which take language and category as arguments (for detail, see Indexing API).

Similarity service allows UI to implement "find similar documents" functionality.

8.2. Implementation details

There are two implementations of the service:

- YADDA similarity
- GensimEudml

8.3. YADDA implementation

YADDA similarity implementation uses Lucene search engine and its "more like this" mechanism. For every language/category pair, separate search index is kept. Also, there exists an
additional index which maps document id to name of index to which document belongs - we will call it 'mapping index'. Similarity algorithm is described below:

- For id similarity search mapping index is used to determine index to which document belongs. In case of SimilarityDocument search, categorization service is used to determine language and category of document - index with matching language and category is used in this case.
- Similarity search is narrowed to search index determined in previous step.
- Lucene's "more like this" query is constructed and executed on search index.

8.4. GensimEudml implementation

GensimEudml uses a Python library, gensim, to do the indexing and querying. Internally, it uses the vector space model to represent documents. Documents are indexed as semantic vectors (by default, using Latent Semantic Analysis) and cosine similarity is used to assess document similarity.

Because of its use of semantic models, the GensimEudml implementation uses an additional step: model training. Before any document can be indexed, a semantic model must be created. This is done only once, over a set of training documents, using the same procedure as for indexing, except the method `train(Collection<SimilarityDocument> docs)` is used instead of `add()`. Once the model is trained, any document dispatched for indexing or for querying will be converted into a semantic vector using this model. Each index (language/category pair) has its own model.

8.5. Alternative implementations using standard API

It should be noted that similarity API is in line with general yadda-services2 service implementation strategy. Alternative implementations should not re-implement API from scratch but instead implement another interface which is main point of current implementation. In that way, some inconveniences related to low level nature of service API can be avoided.

To be more precise, a brief list of key classes/interfaces follows:

- ISimilarityFacade/SimilarityFacade - similarity search facade with generic implementation (should not be changed/rewritten by alternative implementations)
- ISimilarityService - similarity search service API (should not be changed/rewritten by alternative implementations)
- ISimilarityIndexFacade/SimilarityIndexFacade - similarity indexing facade with generic implementation (should not be changed/rewritten by alternative implementations)
- ISimilarityIndexService - similarity indexing service API (should not be changed/rewritten by alternative implementations)

Generic implementations of service interfaces:

- SimilarityServiceImpl
- SimilarityIndexServiceImpl

Both classes basically obtain parameters from low level service requests and delegate to SimilarityModule interface. Alternative implementations should implement this interface. To use alternative implementation Spring configuration of the service should be changed in such a way that SimilarityServiceImpl and SimilarityIndexServiceImpl use alternative implementation of SimilarityModule.
8.6. Dependencies and prerequisites

Current YADDA implementation depends on YADDA Lucene search project (lucene-search) which is used as similarity search engine.

YADDA implementation does not require any external resource apart from access to file system for storing Lucene index.

8.7. API description

8.7.1. Sketch of the API

As mentioned above, the service has two main types of operations: adding documents (indexing) and searching similar documents. Before we pass to details of similarity search and indexing, similarity document model must be described.

8.7.2. Important file/class/package locations

Similarity service is implemented by yadda-simcat module. Main package of the service is pl.edu.icm.yadda.similarity.

8.7.3. Similarity document

Similarity service operates on very simple document model (similarity document is represented in Java by SimilarityDocument class of yadda-model project). Description of document's attributes follows.

- `String id` - document's id (should be unique across the service)
- `String name` - name/title
- `String text` - textual content
- `List<String> authors` - authors
- `List<String> licenses` - labels of licenses associated to the document
- `List<String> values` - list of values stored with a document (the list may be returned with similarity search results if requested)
- `String language`
- `String category`

8.7.4. Similarity result

Similarity search methods return iterators of SimilarityResult objects. SimilarityResult represents single document which has been detected as similar to specified one (the class is part of yadda-model). It has the following fields:

- `String id` - id of document
- `float score` - similarity score (float between 0 and 1; the higher the better)
- `List<String> values` - list of string values (SimilarityDocument.values); the list is not empty if and only if `returnValues` parameter of similarity method is true (see below).
8.8. Similarity facade

Main interface of similarity search API is ISimilarityFacade which is facade to ISimilarityService.

Following methods can be used for finding similar documents:

```java
/**
 * Finds similar documents to specified one. 'returnValues' flag determines if similarity results contain associated document values.
 */
CountingIterator<SimilarityResult> findSimilar(SimilarityDocument doc, boolean returnValues) throws ServiceException;

/**
 * Find similar documents to document with specified id. 'returnValues' flag determines if similarity results contain associated document values.
 */
CountingIterator<SimilarityResult> findSimilar(String id, boolean returnValues) throws ServiceException;

/**
 * Find similar documents to document or document id. SimilarityQuery allows also usage of minimal score, return values flag and filter name.
 */
CountingIterator<SimilarityResult> findSimilar(SimilarityQuery similarityQuery) throws ServiceException;
```

The first two methods have variants without returnValues parameter (they work like original methods with false returnValues parameter).

The returned documents are sorted by similarity score.

Third method uses SimilarityQuery object which has following fields:

- `String id` - id of the document to which similar documents should be found
- `SimilarityDocument document` - complete document to which similar documents should be found (Note exactly one of id and document fields must be non null)
- `float minimalScore` - documents with lower score are not returned (score is between 0 and 1 and so minimalScore parameter equal to 0.0 has no effect on results)
- `boolean returnValues` - if true documents' values are included in the results
- `String filterName` - name of filter which should be applied to the results (see below)

Example

In this example we find similar documents to document stored in index with id "id1". Values associated with found documents are printed out.

```java
ISimilarityFacade facade;
// ...
CountingIterator<SimilarityResult> r = facade.findSimilar("id1", true);
System.out.println("Found "+r.count+" similar documents");

// process results
for (SimilarityResult res : r) {
    System.out.println("Document "+res.getId()+" score: "+res.getScore());
    if (res.hasValues()) {
        System.out.println("Values:");
        for (String val : res.getValues()) {
            System.out.println(val);
        }
    } else {
        System.out.println("No associated values");
    }
}```
8.8.1. Filters

Returned results may be filtered with regard to document licenses. To achieve that, `findSimilar(SimilarityQuery)` method should be used together with filter name specified in the query. Such a filter must be earlier registered. There are two methods which are used to manage filters:

```java
/**
 * Adds specified filter definition to similarity service. If definition with matching
 * name already exists it is replaced if and only if replaceIfExists
 * parameter is true. Only SimilarityFilterDefinition and BooleanFilterDefinition
 * are supported (if different filter definition is passed exception is thrown).
 */
void addFilterDefinition(FilterDefinition filterDefinition, boolean replaceIfExists)
  throws ServiceException;

/**
 * Removes from service filter definition with specified name.
 * Note that boolean definitions containing definitions with matching
 * name will be also removed.
 * @param filterName
 * @throws ServiceException
 */
void removeFilterDefinition(String filterName) throws ServiceException;
```

Similarity service supports only `SimilarityFilterDefinition` which contains a list of license labels and boolean combination of filters (`BooleanFilterDefinition`). Other filter definitions are not supported (exception is thrown if other filter definition are used).

8.8.2. Similarity facade compatible with search API

`BwmetaSimilarityFacade` is a wrapper for `ISimilarityFacade` which returns results compatible with search API. This is a convenience class which makes development of similarity search easier if search is already implemented. The most important method of the facade follows.

```java
/**
 * Search API wrapper for similarity. First and size parameters may be used for paging purposes
 * in the same way as in SearchQuery class of search API.
 */
public SearchResults findSimilar(String id, int first, int size)
  throws ServiceException, YaddaException
```

This method executes `findSimilar` method from `ISimilarityFacade` with `returnValues` parameter set to `true`.

In this example we use results from search API to process similarity results.

```java
ISimilarityFacade facade;
// ...
BwmetaSimilarityFacade searchFacade = new BwmetaSimilarityFacade();
searchFacade.setSimilarityFacade(facade);

// find first 10 results
SearchResults r = searchFacade.findSimilar("id1", 0, 10);
System.out.println("Found "+r.getCount()+" similar documents");
System.out.println("Processing first "+r.getSize()+" results");
for (SearchResult res : r.getResults()) {
    System.out.println("Document "+res.getDocId()+" score: "+res.getScore());
    System.out.println("Associated fields: ");
    for (ResultField rf : res.getFields()) {
        System.out.println("Field name: "+rf.getName()+" values:");
    }
}
```
for (String val : rf.getValues()) {
    System.out.println(val);
}

Associated values are converted to ResultField objects. Note that search api uses concept of field being name/value pair. In order to make BwmetaSimilarityFacade work properly, associated values of documents must be created with help of NameValueUtils.prepareValue(String name, String value) method.

8.8.3. Indexing facade

Two main interfaces in indexing API are ISimilarityIndexFacade and generic ISessionFacade<SimilarityDocument>. A set of documents on which similarity service operates may be modified by addition or removal of SimilarityDocument objects. Modifications can be made through open session to particular index. Session is obtained from ISimilarityIndexFacade with connect methods:

```java
/**
 * Opens new session to similarity index with matching language and category. Both language and * category must be not empty.
 */
public ISessionFacade<SimilarityDocument> connect(String language, String category)
    throws ServiceException;
/**
 * Opens new session to similarity index without specified language and category.
 */
public ISessionFacade<SimilarityDocument> connect() throws ServiceException;
```

Only one session may be opened at the same time. The attempt to open session when another session is opened results in exception.

Before we describe similarity session interface, two batch deletion methods of ISimilarityIndexFacade should be mentioned. One removes all documents from the service; with another one it is possible to remove documents from one particular index or group of indexes:

```java
/**
 * Drops all documents from similarity service.
 */
public void dropAllDocuments() throws ServiceException;
/**
 * Drops documents matching specified language and category from similarity service. * At least one of language and category parameters must be not empty. 
 */
public void dropDocuments(String language, String category) throws ServiceException;
```

As mentioned earlier, documents are added to the service through ISessionFacade<SimilarityDocument>. Take a look at its methods modifying the set of documents in the index.

```java
/**
 * Adds specified documents to the index
 */
void add(Collection<SimilarityDocument> docs) throws ServiceException;
/**
 * Removes from the index all documents with specified ids. 
 */
void delete(Collection<String> ids) throws ServiceException;
```
If id of added document already exists in the index, a new document replaces old document (there will never be two documents in the single index with the same id).

All changes done to the index must be committed or rolled back - after any of these operations, the session is closed and cannot be used.

Note that language and category fields are ignored during addition to session which has been opened for given language and category - because in this case session is created for fixed category and language.

Id field is mandatory when $\text{SimilarityDocument}$ object is passed to the session.

Note 1 There is no predefined set of indexes. The attempt of connecting to an index that does not exist results in creation of new index with specified language and category.

Note 2 In similarity searches using document's id, category and language detection are irrelevant. In searches using $\text{SimilarityDocument}$ for which language and category are not specified, service uses categorization service to detect language and category of document - in order to make it work properly, language and category specified during indexing should be determined with the same instance of categorization service.

Example 1

Connection is obtained and documents are added to the index.

```java
ISimilarityIndexFacade serviceFacade;
// ...
List<SimilarityDocument> docs = new ArrayList<SimilarityDocument>();
// ... create documents and add them to docs list
// all documents are from category 'category' and 'PL' language
// open session and add documents
ISessionFacade<SimilarityDocument> session = serviceFacade.connect("PL", "category");
session.add(docs);
session.commit();
// session can not be used any more
```

Example 2

In this example, different ways of removing documents are shown.

```java
ISimilarityIndexFacade serviceFacade;
// ...  
ISessionFacade<SimilarityDocument> session = serviceFacade.connect("PL", "category");
// remove documents by id - id1 and id4 are removed
List<String> ids = new ArrayList<String>();
ids.add("id1");
ids.add("id4");
session.delete(ids);
session.commit();
// session can not be used any more
// remove all documents having specified category and language
serviceFacade.dropDocuments("PL", "category2");

// remove all documents having language "PL"
serviceFacade.dropDocuments("PL", null);

// remove all documents from all documents having category "category2"
serviceFacade.dropDocuments(null, "category1");

// remove all documents
serviceFacade.dropAllDocuments();
```
8.9. Configuration

As mentioned earlier, key classes implementing similarity service use underlying SimilarityModule implementation.

8.9.1. Facade configuration

Facade configuration consists of generic similarity facades which have given similarity services:

```xml
<bean id="similarityIndexFacade"
     class="pl.edu.icm.yadda.service2.similarity.impl.SimilarityIndexFacade">
  <property name="service" ref="similarityIndexService"/>
</bean>

<bean id="similarityFacade"
     class="pl.edu.icm.yadda.service2.similarity.impl.SimilarityFacade">
  <property name="service" ref="similarityService"/>
</bean>
```

Passed services may be local implementations or proxies to remote services.

8.9.2. Skeleton configuration of the service

Skeleton Spring configuration of the service follows:

```xml
<bean id="similarityIndexService"
     class="pl.edu.icm.yadda.service2.similarity.impl.SimilarityIndexServiceImpl">
  <property name="similarityModule" ref="similarityModule"/>
</bean>

<bean id="similarityService"
     class="pl.edu.icm.yadda.service2.similarity.impl.SimilarityServiceImpl">
  <property name="similarityModule" ref="similarityModule"/>
</bean>
```

As mentioned earlier generic API implementation needs only implementation of SimilarityModule to work. Further configuration requirements depend on implementation of SimilarityModule.

8.9.3. YADDA similarity module configuration

YADDA similarity must have configured following parts:

- index manager and search module from lucene-search project to manage and use Lucene search index
- index metadata name - name of spring bean representing index metadata (technical requirement to manage indexes for new languages)
- categorization facade to allow grouping documents by language/category (current implementation uses only grouping by language)

```xml
<bean id="similarityModule"
     class="pl.edu.icm.yadda.similarity.impl.SimilarityModuleImpl">
  <property name="indexManager" ref="similarityPersistentIndexManager"/>
  <property name="searchModule" ref="similaritySearchModule"/>
  <property name="indexMetadataRef" value="similarityIndexMetadata"/>
  <property name="categorizationFacade" ref="similarityCategorizationFacade"/>
</bean>
```
8.9.4. YADDA similarity search index configuration

YADDA similarity search index configuration configures all the beans responsible for Lucene full-text search. The only parameters which should be changed are paths to the following directories:

- Lucene search index directory (PATH_TO_SEARCH_INDEX_DIR in following configuration)
- Index configuration directory (PATH_TO_CONFIG_DIR in following configuration)

```xml
<bean id="similarityIndexMetadata" class="pl.edu.icm.yadda.service.search.module.config.impl.IndexMetadataImpl">
    <property name="configurationProperties">
        <props>
            <prop key="default_analyzer">accentAnalyzer</prop>
            <prop key="field.text">tokenized, unstored, termVector</prop>
            <prop key="field.values">unindexed, stored</prop>
            <prop key="field.filter">untokenized, unstored</prop>
        </props>
    </property>
</bean>

<bean id="indexDocToIndexMapping" class="pl.edu.icm.yadda.service.search.module.impl.IndexSingleImpl">
    <property name="name" value="indexDocToIndexMapping"/>
    <property name="indexConfiguration">
        <bean class="pl.edu.icm.yadda.service.search.module.config.impl.IndexConfigurationSingleImpl">
            <property name="indexMetadata">
                <bean class="pl.edu.icm.yadda.service.search.module.config.impl.IndexMetadataImpl">
                    <property name="configurationProperties">
                        <props>
                            <prop key="default_analyzer">accentAnalyzer</prop>
                            <prop key="field.indexName">untokenized, stored</prop>
                        </props>
                    </property>
                </bean>
            </property>
            <property name="indexPath" value="PATH_TO_SEARCH_INDEX_DIR/indexDocToIndexMapping"/>
        </bean>
    </property>
</bean>

<bean id="similaritySearchModule" class="pl.edu.icm.yadda.service.search.module.impl.SearchModuleImpl" init-method="init" destroy-method="destroy">
    <property name="filterFactory">
        <bean class="pl.edu.icm.yadda.service.search.filter.impl.FilterFactoryImpl">
            <property name="filtersLimit" value="100"/>
            <property name="booleanFiltersLimit" value="100"/>
        </bean>
    </property>
    <property name="moreLikeThisQueryFactory">
        <bean class="pl.edu.icm.yadda.service.search.module.impl.MoreLikeThisQueryFactoryImpl">
            <property name="minDocFreq" value="5"/>
            <property name="minTermFreq" value="2"/>
            <property name="minWordLen" value="3"/>
            <property name="maxWordLen" value="0"/>
            <property name="maxQueryTerms" value="25"/>
            <property name="maxNumTokensParsed" value="5000"/>
        </bean>
    </property>
    <property name="analyzerFactory">
        <bean class="pl.edu.icm.yadda.service.search.module.config.impl.AnalyzerFactoryImpl">
            <property name="analyzers">
                <list>
                    <bean class="pl.edu.icm.yadda.service.search.analyzer.AnalyzerInfo">
                        <property name="name" value="accentAnalyzer"/>
                    </bean>
                </list>
            </property>
        </bean>
    </property>
</bean>
```
8.9.5. YADDA language/categorization facade configuration

Currently, YADDA implementation only uses language grouping. Example configuration follows.
8.9.6. Possible extensions

YADDA implementation could be extended with categorization service which not only detects language but also determines category to which document belongs. With such grouping similarity searches may be possibly much more accurate.
9. Annotation Service

Annotations enable users to add information (comment or annotate) to existing URIs (DL units, comments, "personal bookshelves").

9.1. Module responsibility

Annotation service is a low-level module that can store annotations. Each annotation has at least an owner, target (URI), state and body. Annotation body is stored in YADDA MD-Storage, while its relations (ownership, target, annotation type, annotation format, language, visibility, state) are stored in Sesame RDF storage. Annotations are never thought to be deleted, instead the state has to be changed to "deleted". So far, the service does not contain methods for changing existing annotations.

9.2. Non-functional requirements met by the module

The annotation service provides methods to store relations of an annotation such as ownership (creator=owner), target URI, annotation type, annotation format, annotation language, visibility and state. All these attributes are in form of strings, so that a high level service can create and handle categories (e.g. annotation visibility "private" or "public"). The annotation service does not handle user authentication or authorization.

The module has to be code-injected with the appropriate storages information (both YADDA and Sesame).

9.3. Implementation details

Interfaces

• eu.eudml.annotationservice.IAnnotation
• eu.eudml.annotationservice.IAnnotationBody
• eu.eudml.annotationservice.IAnnotationService

Classes

• eu.eudml.annotationservice.impl.Annotation and eu.eudml.annotationservice.impl.AnnotationBody are serializable objects that represent annotation and its body respectively. These objects can be created or modified (remotely) by the service.
• eu.eudml.annotationservice.impl.AnnotationService handles two storages (YADDA MD-Storage and Sesame RDF Storage). It provides several methods:

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init()</td>
<td>should be called at the beginning</td>
</tr>
<tr>
<td>addAnnotation(String userId, String target, String type, String body, String format, String language, String visibility, String state)</td>
<td>add annotation</td>
</tr>
<tr>
<td>Collection&lt;IAnnotation&gt; getAnnotationsByTarget(String target)</td>
<td>return annotations related to a given target</td>
</tr>
<tr>
<td>Collection&lt;IAnnotation&gt; getAnnotationsByAuthor(String creator)</td>
<td>return annotations related to a given creator</td>
</tr>
</tbody>
</table>
Storages (the service uses 3 configuration beans):

- catalogFacade - YADDA MD-Storage
- editorFacade - YADDA editor
- sailFactory - Sesame RDF repository

Dependencies and prerequisites

- YADDA MD-Storage, YADDA editor
- Sesame RDF storage
- XStream for serialization

9.4. API description

Example of usage is in annotation-backend package.

The use of the service is intended as follows:

- create the IAnnotationService interface variable for instantiation
- get the service by using
  org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean (in the following example this step is hidden in the remoteAnnotatationService bean
- initialize the service by the init() method
- use the service by calling methods addAnnotation(), getAnnotationByTarget() etc.

```java
IAnnotationService remoteCall = (IAnnotationService)ctx.getBean("remoteAnnotationService");
try{
    remoteCall.init();
    System.out.println(remoteCall.addAnnotation("", "http://target.url", ", "body of the annotation", ", ", ", ");
} catch (Exception e){
    e.printStackTrace();
}
```

The remoteAnnotatationService bean in the example above has two properties:

- serviceUrl: e.g. http://localhost:8080/annotation-service/remoting/annotation-service
- serviceInterface: eu.eudml.annotationservice.IAnnotationService

9.5. Configuration

The service is set up for EuDML purposes in the Spring application context configuration in the following files:

- catalog-beans.xml - configuration of the YADDA MD-Storage and editor
- triple-store-beans.xml - configuration of the Sesame RDF store
- springXMLConfig.xml - properties for catalog-beans.xml

9.5.1. Configuration of the YADDA MD-Storage

The setup consists of yaddaCatalog and yaddaEditor facade beans using a catalogEditorService bean of the RecorddbEditor class as the actual service implementation using several supporting beans.

For more information about configuring the YADDA MD-Storage and editor see section 5.11 YADDA_MD-Storage.
9.5.2. Configuration of the Sesame RDF store

The bean `sailFactory` contains a particular class that constructs the Sail. For the purpose of remote calling it must be a factory. `org.openrdf.sail.config.SailFactory` is an interface that takes care of creating and initializing a specific type of Sails based on RDF configuration data. `org.openrdf.sail.Sail` is another interface that has implementations such as MemoryStore, NativeStore etc. Sail is a result of a sail factory.

The `rdfRepository` bean contains a class that constructs a SailRepository. Depending on the type of the repository, additional arguments can be passed to the constructor.

For more information about sail types and sail factory types, see the Sesame documentation.

9.6. Possible extensions

- editing existing annotations
- return collection of annotation depending on a SeRQL query
10. ID Service

ID Manager is a utility for managing proper IDs for publications. It can be used for assigning new EuDML identifiers to native publication identifiers, as well as for retrieving them. Every Eudml identifier has a set of identifiers harvested from publications assigned to it.

10.1. Module responsibility

Module responsibility is to manage identifiers. It allows for:

- automatic assigning new EuDML identifiers,
- adding/removing harvested identifiers to existing EuDML identifier,
- editing identifiers assigned to EuDML identifier,
- joining sets of identifiers into one EuDML identifier.

10.2. Non-functional requirements met by the module

Managing identifiers is transactional.

10.3. Implementation details

Identifiers are stored in relational database in format: id_type | id_value | eid Where <id_type, id_value> is native identifier, and eid is newly assigned EuDML identifier.

Id Manager retrieves data from database using IdsDao interface which exposes simple operations on identifiers. It is implemented by JdbcIdsDao which uses Spring Framework for managing database queries and transactions.

Id Manager functionality is enclosed in JdbcIdManagerFacade which implements IdManagerFacade.

10.4. Dependencies and prerequisites

Current ID Manager implementation is configured to use PostgreSQL database, therefore it is required to deploy such database on server running manager. JdbcIdManagerFacade has to be initialized before using. It implements Configurable interface for that purpose.

10.5. API description

Actual implementation of ID Manager is available through IdManagerFacade interface. IdManagerFacade provides following methods:

- IdQueryResult queryId(Collection<Identifier> ids)
  Queries for sets of identifiers which point to the same object as elements of ids.

- Identifier requestId(Collection<Identifier> ids)
  Requests one eid for all identifiers in ids.

- void mergeIds(Collection<Identifier> ids, Identifier eid)
  Merges identifiers into one EuDML identifier.
Merges identifiers in ids collection and assigns them to pointed eid.

- void updateIdentifiers(Identifier eid, Collection<Identifier> ids, Action action)

Performs pointed action on identifiers set represented by eid and ids.

Actions are:
- SPLIT - substracts set of ids and assigns new eid to it
- SUBTRACT - substracts given ids
- ADD - adds given ids

For more detailed description of methods look into JavaDoc. For usage and deployment tips look into tests.

**10.6. Data Structures**

Identifier - it is a pair of type and value. First represents type of identifier, second it's value.

IdentifierSet - consists of eid and identifiers which are assigned to it.

IdQueryResult - it is a result for queryId. Contains List<IdentifierSet> identifierState - list of identifier sets which are connected to queried ids

**10.7. Important file/class/package locations**

API is in the services-api in eu.eudml.service.idmanager package, and implementation in services-impl in eu.eudml.service.idmanager.* packages.

For working configuration you can look at eu/eudml/service/idmanager/idmanager-facade-test-config.xml in Other Test Sources. It is used by JdbcIdManagerFacadeTest.

**10.8. Configuration**

IdManagerFacade is configured through Spring configuration file:

eu/eudml/service/idmanager/idmanager-jdbc-facade.xml. File needs bean dataSource of type DataSource, and exports idManagerFacade bean - instance of JdbcIdManagerFacade.
11. YADDA MD-Storage Service

The MD-Storage Service is a storage component for textual information. Its purpose is to store small data items with supporting meta-information. Objects stored in the service follow a simple model containing:

- an identifier
- status information (whether it is active or deleted)
- version information
- modification timestamp
- a set of labels (tags) which may be used for queries
- a set of typed (labeled) data records

The objects can be retrieved both individually, based on their identifiers, and iterated with automatic paging using queries based on tags, types and timestamps. Objects in the service can be stored, edited and deleted both individually and in batches, with optional transactional semantics for the batch operations.

11.1. Module responsibility

The MD-Storage Service is used by the Storage Service as the main storage for metadata records for publications.

11.2. Non-functional requirements met by the module

None.

11.3. Implementation details

The MD-Storage Service stores the data in a SQL database. There are three tables used, each with appropriate indexes:

- `object_metadata` for the basic metadata (identifier, status information, version information, timestamp),
- `object_tags` for the tags,
- `object_parts` for the actual data labeled with types.

The service implements the `pl.edu.icm.yadda.bean.Configurable` interface in order to be able to create the tables if they do not yet exist in the database.

The editor service strictly requires operations to be performed in transactions. Because like all YADDA service interfaces designed for remoting the `IEditor` interface does not allow throwing exceptions, to facilitate the use of Spring-managed transactions with rollback on exceptions an `IEditorBackend` interface is created for the purposes of transaction management.

For concurrent edit management the editor service uses an optimistic soft locking scheme. If an edit operation specifies the version information for the object to modify and the actual version encountered in the transaction is different, the operation is rolled back and an error is signaled.
11.4. Dependencies and prerequisites

The MD-Storage service implementation is using a SQL database to store the data. PostgreSQL and SQLite are supported, with PostgreSQL recommended and used in EuDML. The service needs to be configured with access to an existing PostgreSQL database and can create all the necessary tables for itself.

11.5. API description

There are two convenient interfaces provided for users of the service:

- **ICatalogFacade** for read-only access,
- **IEditorFacade** for data storing and modification.

There are also two corresponding interfaces for remote access: **ICatalog** and **IEditor**.

The read-only interfaces provide methods for:

- retrieving all the information or metadata only for an object with a specific identifier,
- iterating through metadata of objects with specified tags, timestamps or containing data of a specified type,
- retrieving data record of a specific type from an identified object,
- iterating through data records of a specific type from a collection of objects with given identifiers.

For the remote access interfaces, iteration is implemented as paging, using tokens returned by the service to request subsequent pages.

The editor interfaces provide methods for:

- adding a new object or adding/replacing/removing data records in an existing object,
- adding or removing tags of an existing object,
- removing an existing object (marking it as deleted),
- performing any of these operations in batch.

The data model used for objects stored in the service consists of the **CatalogObjectMeta** class representing all the object metadata and the **CatalogObject** class extending it with a collection of **CatalogObjectPart** objects that contain the actual type-labeled data.

11.6. Important file/class/package locations

The Regsitry interfaces are located in the **pl.edu.icm.yadda.service2.catalog** package, and the editor interfaces in the **pl.edu.icm.yadda.service2.editor** package. Common model classes are in the **pl.edu.icm.yadda.service2** package.

Implementation classes can be found in the **pl.edu.icm.yadda.service2.catalog.recorddb** and **pl.edu.icm.yadda.service2.editor.recorddb** packages.

11.7. Configuration

The service is set up for EuDML purposes in the Spring application context configuration in the **src/main/webapp/WEB-INF/application-context/yadda-services.xml** file of the eudml-backend module.
The setup consists of yaddaCatalog and yaddaEditor facade beans using a catalogEditorService bean of the RecorddbEditor class as the actual service implementation using several supporting beans.

Because the implementation requires edit operations to be transactional, the setup uses Spring transaction support to configure the methods on the IRecorddbEditorBackend interface to be performed in transactions.

Since the MD-Storage Service is used as the backend for the EuDML Storage Service, the configuration properties used in this application context setup are described in section 5.3 Storage Service. They are used to provide the service with database connection details.
12. **YADDA Archive Service**

The Archive Service is a storage component for binary structured content. Its purpose is to store large data files with supporting meta-information. Data is stored in the service following a simple model containing:

- object identifier
- status and version information
- modification timestamp
- object type
- a set of tags which may be used for queries
- a set of typed data records
- information about relations to other objects in a tree structure

The internal tree structure allows storage reflecting publication structure (pages, chapters etc.). Because the service is used to store and serve large binary files it is expected to handle up to several TB of data and is able to transfer it to clients in chunks using a variety of protocols.

The objects can be retrieved individually, based on their identifiers, alone or together with their children in the tree structure. They can also be iterated with automatic paging using queries based on tags or timestamps.

Objects in the service can be stored, edited and deleted both individually and in batches, with optional transactional semantics for the batch operations.

12.1. **Module responsibility**

The YADDA Archive is used by the Storage Service as a backend for the storage of publication data including especially full texts of publications.

12.2. **Non-functional requirements met by the module**

Describe the expected performance of the module, and other non-functional aspects like security, fail-over and HA, transactivity.

12.3. **Implementation details**

The YADDA Archive service stores the data in a SQL database and in the filesystem. All the object metadata and data up to a size limit is kept in the database. There are four tables used, each with appropriate indexes:

- **archive_object** for the basic metadata (identifier, status information, version information, timestamp, object type, parent object reference),
- **archive_tag** for the tags,
- **archive_content_part** for the actual data if it is small enough or data location in the filesystem otherwise, plus data size, type, MIME type and checksum,
- **archive_index** for locating objects based on their position in the tree structure.
For data that is too large to be kept in the database, the YADDA Archive uses a configured filesystem location where the files are stored in a directory structure with random filenames and stores the filenames in the database.

The service implements the `pl.edu.icm.yadda.bean.Configurable` interface in order to be able to create the tables if they do not yet exist in the database and to create the storage directory.

For direct access to object data via HTTP, the service provides the `HttpProvidePartServlet`, which can be used to respond to requests of the form `objectPath?type=partType`.

### 12.4. Dependencies and prerequisites

The archive service implementation is using a SQL database to store the data. PostgreSQL and SQLite are supported, with PostgreSQL recommended and used in EuDML. The service needs to be configured with access to an existing PostgreSQL database and can create all the necessary tables for itself. The service also requires a writable filesystem directory for storage of larger data.

### 12.5. API description

There are two convenient interfaces provided for users of the service:

- `IArchiveFacade2` for read-only access,
- `IStorageFacade2` for data storing and modification.

There are also two corresponding interfaces for remote access: `IArchive2` and `IStorage2`.

The read-only interfaces provide methods for:

- retrieving the metadata for an object with a specific identifier, including children objects or not,
- iterating through the metadata of objects with a specified relation to an ancestor object in a tree structure,
- iterating through metadata of objects with specified tags or timestamps, possibly limited to tree roots,
- retrieving data from a record of a specific type in an identified object.

For the remote access interfaces iteration is implemented as paging, using tokens returned by the service to request subsequent pages.

The storage writing interfaces provide methods for:

- adding a new object or adding/replacing/removing data records in an existing object,
- removing an existing object (marking it as deleted),
- performing any of these operations in batch.

The data model used for objects stored in the service consists of the `ArchiveObjectMeta` class representing the object metadata, the `ArchiveObject` class extending it with a content reference, the `ArchiveContent` class containing a collection of data record metadata, the `ArchiveContentPartMeta` class that contain contains data record metadata (type, MIME type, checksum, size) and the `ArchiveContentPart` class used for transfers of actual data.
12.6. Important file/class/package locations

The archive interfaces are located in the `pl.edu.icm.yadda.service3.archive` package, and the storage interfaces in the `pl.edu.icm.yadda.service3.storage` package. Common model classes are in the `pl.edu.icm.yadda.service2` package. Implementation classes can be found in the `pl.edu.icm.yadda.service3.archive.db` and `pl.edu.icm.yadda.service3.storage.db` packages.

12.7. Configuration

The service is set up for EuDML purposes in the Spring application context configuration in the `src/main/webapp/WEB-INF/application-context/yadda-services.xml` file of the `eudml-backend` module.

The setup consists of `yaddaArchive` and `yaddaStorage` facade beans using an `archiveService` bean of the `DbArchive2` class and the `storageService` bean of the `DBStorage` class as the actual service implementations using several supporting beans.

Because the implementation requires edit operations to be transactional, the setup uses Spring transaction support to configure the methods on the `IStorageBackend` interface to be performed in transactions.

Since the YADDA Archive Service is used as the backend for the EuDML Storage Service, the configuration properties used in this application context setup are described in section 5.3 Storage Service. They are used to provide the service with database connection and data directory location details and the address to use for the content servlet.
13. **YaddaWebLite**

YaddaWebLite (YWL) is a framework allowing to easily implement web based repository browsing applications. YWL is not bound to any specific data source, metadata format and user interface layout. It can be easily bound to existing applications in order to provide repository browsing functionalities.

13.1. **Module responsibility**

YaddaWebLite is responsible for implementing typical use cases of the repository frontend application. It provides components which allow to:

1. Search for publications
2. Page lists retrieved from different sources
3. Display details of particular publications in convenient way
4. Filter and enrich text entities from the publication metadata to be displayed on the webpage (highlight search texts, convert domain specific formatting into HTML)
5. Consistently handle system errors, access violations and trials to access non-existent resources.
6. Display AJAX tree with hierarchy of publications

YaddaWebLite is a Java framework intended to be used along with Spring and Spring MVC frameworks. YWL is distributed as a set of JAR packages which may be included in particular Java Web applications. Having included YWL in the application is possible to take use of concrete classes or components to implement particular functionalities. In most cases it will be necessary to implement some domain-specific environment of particular components like data-sources, metadata de-serialization components, view templates etc. Since all components belonging to the YWL are loosely coupled, it's possible to use only some of them. Typical YaddaWebLite based application looks as in Figure 17.

![Figure 17: YaddaWebLite with EuDML Backend Services](image-url)
13.2. Non-functional requirements met by the module

All components belonging to the yaddaweb-lite should easily integrate with wide range of web applications realized in different technologies. The application based on YWL should easily handle heavy load and large numbers of concurrent users. In order to achieve that, the footprint of each user session should be minimized.

13.3. Implementation details

All components of the YWL framework are simple POJO components intended to be integrated into single Spring MVC based web applications. All of the YWL Services support Inversion Of Control pattern to set up application environment and run-time configuration.

13.3.1. Detail service

The service allows fetching and displaying metadata of specified publication in proper form. The service takes the following actions:

- takes an identifier of the publication
- retrieves its metadata from the data-source,
- de-serializes metadata
- resolves type of the publication
- resolves proper way of displaying the publication (proper view)
- transforms native metadata format into the format specific for requested view (and possibly fetches some additional data about publication from different sources)
- forwards result of the former step along with requested view name to the view rendering subsystem (in most cases JSP)

Detail service may take use of the filtering service in order to filter captions belonging to the view model.

Typically, each view model of the publication is build from multiple independent parts. Splitting the process of view model building into parts allows reusing components generating particular aspects of publications like cover (occurring for books and journal volumes) or authors (occurring for books and journal articles).

13.3.2. Filtering service

The main purpose of the filtering service is to convert and postprocess captions belonging to the view model. Particular filters convert domain specific text formatting into client specific formatting (mostly HTML), remove/escape characters and chunks which are illegal for the client (e.g. protection against JavaScript injection). Filters also allow enriching captions with some context specific information like highlighting of the search terms, transforming keywords in text into links to description of particular keywords.

Filter service is usually invoked from detail service or paging service. Filter service allows to defining multiple schemes of filtering; each scheme consists of the chain of particular token processors (implementations of ITokenProcessor interface). Using multiple schemes allows the user to differentiate captions occurring in different contexts. The two main chains of filters are usually defined:

1. the chain which results in valid HTML markup text
2. the chain which results in plain text without any XML tags
In addition, it's possible to the context into the filtering chain. The context can contain any information which may be relevant for particular token processors. For example, search term highlighting token processor expects search terms to be available in the context.

Filtering service operates on tokens. Each token processor in chain transforms the list of input tokens into a list of output tokens. At the beginning of the filtering chain, the text is represented as a singular token. Each token processor from the chain may split input token into multiple result tokens, and may merge multiple input tokens into single result token. The result of the filtering chain should consist of one token.

Each token has a content and the type which allows to differentiate the way how particular token processors treat different tokens. For example, when processing XML content typical token types are:

- opening tag
- closing tag
- content tag

13.3.3. Paging service

Paging service provides generic functionality of paged iteration through multiple kinds of lists. It may be used to page dictionaries or search results. The service supports both stateful and stateless data-sources.

The main object to deal with in the service is a paging context (IPagingContext). Paging context represents current state of the paging, allows traversing to other pages of the data set and allows fetching contents of the current page. Two kinds of context can be defined: stateful and stateless. Stateful context must hold some resources all the time during the iteration session (e.g. SQL database cursor), while stateless context may be represented by a textual token and reconstructed in any time. Depending on the type of the context, Paging Service will keep all conversation contexts bound to the user session (usually HTTP Session) or during each operation on the pager, the token representing current state of pagination will be posted to the user, and on each operation user will provide the token as a request parameter.

Paging service seamlessly integrates with Spring MVC framework, providing generic MVC controller which allows to present the pagination result and to perform any actions on the pager (moving to the previous/next/first/last page).

Paging service integration with MVC uses the Redirect-After-Post pattern of performing HTTP Requests. When user wants to iterate to the next page of the list it clicks on the button which causes sending POST request of going to the next page to the browser. The server changes the state of the paging context (moves the cursor one page next), and replies with HTTP redirect request which expects the web browser to show current page of the specified paging context. When (in that case) user presses "refresh" button, the current page that is being refreshed and the action of going to the next page isn't performed.

The service consists of the following components:

- IPagingContext and its implementations which are responsible for handling paging
- IPagingContextFactory which builds IPagingContext from specified query
- IPagingService and its implementation SimplePagingService which allows to build new paging contexts and retrieve existing ones
- PagingController, SpringRegisterPagingInterceptor and PagingStateHandler which glues IPagingService and existing contexts with Spring environment

The PagingService is/will be used in EuDML UI to:
13.3.4. Notification service

Notification service is a simple component which allows publishing text notifications which will be displayed on the web page. Notification service is available through Spring on each abstraction level (especially in middle-layer level) which allows to consistently publish notifications directly from business logic. When the notification is published, it's bounded to the HTTP session of the current user. When the first web page after notification publishing is being displayed to the user, all notifications from the session are injected to the view model and displayed on the proper area of the page. After displaying, notifications are removed from the session. Notification service supports internationalization (each notification contains the identifier of the localized label).

13.3.5. Hierarchy Tree

Hierarchy Tree is a component which allows browsing the hierarchy tree of publications on the web page. The nodes of the tree might be publishers, journals, volumes and articles. Exact hierarchy, form of each node and used data-source is defined by designer of the application which uses Hierarchy Tree. Hierarchy Tree is dynamic, which means that expanding and collapsing of each node won't cause reloading the whole page. The service is developed on a base of jQuery framework and uses AJAX pattern.

In order to implement a Hierarchy Tree, it is necessary to implement vendor specific implementation of TreeNodeBuilder interface.

13.3.6. Error handling service

The set of error handling components extends standard error handling mechanisms existing in Spring MVC framework. Standard way of handling exceptions in Spring MVC allows to present particular JSP page with error message when an exception occurred during request handling. YaddaWebLite error handling component extends standard Spring MVC flow with:

- reporting of exception occurrences in logs along with generated unique identifier of the problem. Unique identifier of the problem is also provided to the result page context and might be displayed on the error page. The user who got the error may refer to this identifier when reporting a bug.
- mapping some standard exceptions (object not found, access violation) to proper HTTP status codes. With this functionality, standard errors like trying to reach to nonexistent publication can be interpreted by harvesters (e.g., Google), proxy servers and other automatic tools.

13.4. Dependencies and prerequisites

The YaddaWebLite is based on the Spring Framework and Spring MVC and is expected to be used with Java Servlets technology. Although so far all applications based on YaddaWebLite used JSP and Tiles as view rendering technologies, the YWEB Lite itself is not bounded to any view template tool, so a wide range of libraries like Apache Velocity or Freemarker can be used. The YaddaWebLite doesn't make any assumptions on technologies used to serve data and
provide searching and browsing. Developers of particular applications have to implement connectors serving data in expected technologies.

13.5. **API description**

The YaddaWebLite is a framework not a concrete service. Typical usage scenario of the framework is to implement some domain-specific components and configure it along with the framework core.

13.6. **Important file/class/package locations**

Relevant packages and classes:

<table>
<thead>
<tr>
<th>Package/class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pl.edu.icm.yadda.ui.content</td>
<td>Set of components responsible for serving publication contents</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.details</td>
<td>Detail providing service</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.details.model.ymodel.*</td>
<td>Sample detail rendering components based on detail service which use YModel as a data source model</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.details.spring.DetailsController</td>
<td>Integration of detail service with Spring MVC Framework</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.details.filter</td>
<td>Detail text entities postprocessing and enriching framework</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.details.filter.impl.processor</td>
<td>Set of basic text entity postprocessors and enrichers</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.dw.r.*</td>
<td>Misc AJAX components</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.exceptions.*</td>
<td>Components responsible for error handling and displaying error messages</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.mail.*</td>
<td>Template based email sending component</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.newmessaging.*</td>
<td>User notification service</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.pager.*</td>
<td>List paging service</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.pager.spring.*</td>
<td>Integration if list pagers with Spring MVC Framework</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.search.*</td>
<td>Sample implementation of search query forms with validation support</td>
</tr>
<tr>
<td>pl.edu.icm.yadda.ui.tree.*</td>
<td>Renderer of the AJAX hierarchy tree</td>
</tr>
</tbody>
</table>

13.7. **Configuration**

In typical application, all parts of the framework are bounded together using a Spring framework and configured there.

13.7.1. **Detail service**

The main entry point of the details service is MVC subsystem of the Spring Framework, which defines how particular HTTP requests are handled. Here is the definition of Detail Service controller:

```xml
<bean name="/details/element.action"
class="pl.edu.icm.yadda.ui.details.spring.DetailsController">
  <property name="elementHandler" ref="elementHandler" />
  <!-- reference to the element handler service -->
</bean>
```

Here is an example of the element handler, resolver and view builders definition:

```xml
<bean id="elementHandler"
  class="pl.edu.icm.yadda.ui.details.impl.CatalogBasedElementHandler"
  scope="request">
```
13.7.2. Paging service

The entry point in Spring MVC looks as follows:

```xml
<bean name="/page.action" class="pl.edu.icm.yadda.ui.pager.spring.PagingController">
  <property name="pagingService" ref="pagingService" />
</bean>
```

The rest of the paging service environment consists of the following beans:

```xml
<bean id="pagingService" class="pl.edu.icm.yadda.ui.pager.SimplePagingService" scope="session">
  <property name="pagingContextFactories">
    <list>
      <ref bean="simpleSearchPagingContextFactory"/>
      <ref bean="browserPagingContextFactory"/>
      <ref bean="similarSearchPagingContextFactory"/>
    </list>
  </property>
</bean>
```
13.7.3. Hierarchy tree service

Hierarchy tree configuration and instantiation code is located in Spring MVC context file. An example can be seen below:

```xml
<bean name="hierarchyTreeController" class="pl.edu.icm.yadda.ui.tree.HierarchyTreeController">
    <property name="view" value="jsonView"/>
    <property name="hierarchyService" ref="HierarchyService"/>
    <property name="infoService" ref="InfoService"/>
    <property name="treeNodeBuilders">
        <list>
            <ref bean="bwmetaTreeNodeBuilder"/>
            <ref bean="fallbackTreeNodeBuilder"/>
        </list>
    </property>
</bean>

<bean id="bwmetaTreeNodeBuilder" class="pl.edu.icm.yadda.ui.tree.bwmeta.BwmetaTreeNodeBuilder">
    <property name="hierarchyService" ref="HierarchyService" />  
    <property name="configurationService" ref="configService"/>
</bean>

<bean id="fallbackTreeNodeBuilder" class="pl.edu.icm.yadda.ui.tree.SimpleTreeNodeBuilder"/>
```

13.8. Possible extensions

YaddaWebLite itself is intended to be a framework, so extending it is the main pattern usage. YaddaWebLite allows performing the following customizations of particular services:

13.8.1. Application-specific datasources and data formats

We can flexibly define source of the publication details by providing proper implementation of some interfaces of YaddaWebLite framework. In order to use specific publication data source it’s necessary to:

- Create proper implementation of IElementHandler (desirably inherited from AbstractElementHandler) which retrieves and deserializes publication data from expected data-source.
- Configure detail service to use element handler implemented in previous step.
- Update all view part builders which retrieve additional data from external data-sources to use expected data-source.
It's also possible to use custom search and paging engines. In order to use custom paging data source it's necessary to:

- Implement paging context component specific to the datasource (implementation of IPagingContext interface).
- Implement paging context factory class which builds the context from the query string (implementation of IPagingContextFactory class).
- Register paging context factory in the Paging Service (as a property of PagingService Spring bean).
- Prepare view template specific to the paging context and configure paging context to use this view template (view property of the IPagingContext bean).
- Create HTML form/link which initiates paging (search form/menu).

### 13.8.2. Application-specific way of displaying publications

In order to customize the way how publications are displayed by detail service it's necessary to:

1. Prepare view template (e.g. using JSP technology)
2. Find proper standard view part builders, or implement own ones. It is also possible to extend existing part builders using standard Java inheritance.
3. Configure view builder specific to the custom view in order to use the chosen view part builders and view template defined in previous steps.
4. Configure view resolver bound to the display service in order to use specified view builder to render specified type of publications.

### 13.8.3. Application-specific source of hierarchy data and way of displaying hierarchy tree

- Provide proper HierarchyService/InfoService components which provide data about hierarchies.
- Provide proper implementation of TreeNodeBuilder which builds tree nodes basing on the data fetched from HierarchyService/InfoService.
- Configure HierarchyTreeController in SpringMVC in order to use implementations defined in previous steps.
14. Metadata Enhancement Services and Tools

14.1. MiaS

MIaS is a math aware searching engine. It provides the ability to search documents for mathematical formulae. It's based on full-text search core Apache Lucene. MIaS indexes documents containing mathematical notation in Presentation MathML. It allows users to search for mathematical formulae as well as textual content of documents. As far as mathematics is concerned, MIaS matches exact formulae, formulae with a certain degree of similarity and also sub-formulae - logical sub-parts of formulae up to single variables, symbols, constants, etc. For calculating the relevancy of matched expressions to user’s query, MIaS uses heuristic weighting of indexed terms, which accordingly affects scores of matched documents and thus order of results.

14.1.1. Design

Figure 18 shows a scheme of the system workflow:

![MiaS Workflow Diagram](image)

**Figure 18: MiaS Workflow**

14.1.2. Indexing

MIaS is currently able to index documents in XHTML, HTML and txt formats. As the picture shows, input document is at first split into textual and mathematical part. Textual content is indexed in conventional way, using Lucene’s Standard Analyzer. Mathematical expressions, on the other way, are pre-analyzed in several steps to allow mentioned above – search not only for exact whole formulae, but also for subparts (tokenization) and for equal expressions written differently (formulae modifications) and thus address the issue...
of static character of full-text search engine. This creates several representations for each input formula, all of which are indexed. Along this process, weights are assigned to each obtained expression based on these processing steps following the general rule – the lower level of a sub-formulae and the more modified one, the less weight assigned.

14.1.2.1 Tokenization

This is a straightforward process of obtaining sub-formulae from input formula. MIaS makes use of Presentation MathML markup where all logical units are enclosed in tags so obtaining of all sub-formulae is a question of tree traversing. The inner representation of each formula is an XML Node encapsulating all the belonging child nodes. This means the highest leveled formula – as in the input document – is represented by node named “math”. All obtained expressions are stored in the memory and passed on to modification algorithms.

14.1.2.2 Formulae modifications

MIaS performs three types of unification algorithms, goal of which is to create several unified representations of all gained formulae from the tokenization process and to allow a certain amount of distinctness between user query and the term in the document allowing the system to return also similar matches to the user while preserving the formula structure and \(\alpha\)-equality.

Ordering:

Let’s imagine a simple example: \(a + 3\) and the query \(3 + a\). This wouldn’t match even though it’s perfectly equal. That’s why a simple ordering of the arguments of the commutative operations addition and multiplication is used. It tries to order arguments of these operations in the alphabetical order of the XML nodes denoting the arguments whenever possible – it considers the priority of the surrounding operators. System applies this function to the formula going to the index as well as the query expression meaning the two formulae with swapped such arguments will match this time. Applied to the example above: \(3 + a\) will be swapped for \(a + 3\) because the node denoting variable \(a\) is named “mi” and the node denoting number 3 is named “mn” and “mi”<”mn”.

Unification of variables:

Let’s take an example again: \(a + b\) and \(x + y\) again wouldn’t match even though it only used different variables. That’s why MIaS unifies variables in the expressions, also considering bound variables. All variables are swapped for unified symbols (ids) in the indexing and also in the searching phase. Applied to the example, both expressions will unify to \(id_1 + id_2^{id_1}\) and therefore will match. Single symbols are not unified – this would lead into indexing millions of ids and searching for such symbol would end up matching all the documents containing any symbol.

Unification of constants:

This is the simple process of swapping all the numerical constants for one unified symbol (const). This eases away the need of the exact numbers in formulae. As well as the case of the variables, stand-alone numerical constants are not unified.

14.1.2.3 Formulae weighting

In the searching phase a query can match several terms in the index. However, one match can be more important than the other and the system must consider this information when
scoring matched documents accordingly. For mathematical expressions, the system must make use just of the things it has done with them in the processing phase.

The input unchanged formula should of course weight the most, but the exactness of the representation with ordered arguments is not compromised at all, so it weights the same. In fact, the original formula isn’t indexed at all. Starting weight for this representation is 1.

The tokenization process should naturally lower the weight of the subformulae. When user searching for \( a + b \) finds two documents, the first containing \( a + b \) and the second containing \( \frac{2}{a+b} \), the first should score more and appear higher in the results. So, the tokenization process reduces weight of the subformulae of each lower level by one half.

Then the system makes use of the unification algorithms. Of course every representation created by these algorithms is more unified and should score less, because they have higher probability of matching. The weight of each such representation is lowered to four fifths of the previous weight by both modification algorithms. It’s based on a simple speculation: An expression that ran through both unification algorithms should score more than an expression with exact variables and constants matched on the lower level of the derivation tree (\( 0.8 \times 0.8 > 0.5 \)). Example: query for \( x + y \) should return document containing \( a + b \) higher than the document containing \( \frac{3}{x+y} \). This assumption may be a subject of discussion and is about to be evaluated by created utility.

According to this model, each formula has a weight attribute indexed next to itself, which belongs to the interval \((0, 1)\).

The example in Figure 19 shows whole formula pretreatment process together with weighting. Ordered pairs are (expression written naturally, its weight):

![Figure 19: Formula pretreatment process together with weighting](image)
The whole Lucene’s scoring function is as follows:

\[ \text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t \cdot \text{getBoost}() \cdot \text{norm}(t, d)) \]

and it’s described here:

http://lucene.apache.org/java/3_0_2/api/core/index.html?org/apache/lucene/search/Similarity.html

The way how the formulae weights are considered in this function is simple – \( \text{tf}(t \text{ in } d) \) is multiplied by it. If a not completely unified formula finds a match in the index, it will, of course, match also the more unified representation/s. This won’t affect relevancy scoring, since scoring values of all matched terms are summed.

Let’s imagine formulae weights are the only scoring factors of matched documents. Following the example above, let’s say the expression \( c + 2 \) is matched – the score of matched document will be \( 0.25 + 0.2 + 0.2 + 0.16 = 0.81 \). If a document contains the same expression more than one time (each occurrence can have different weight assigned), the average value of all the weights is considered.

14.1.3. Searching

In the searching phase, the user input is again split into mathematical and textual part. Mathematical part of the query is then run through the same process as in the indexing phase, except tokenization – we can hardly suspect user querying for example for \( \frac{a+b}{c} \) wanting to search documents only for occurrences of the variable \( c \). This means the queried expressions are first ordered and then unification algorithms are done. This again produces several representations which are connected in the final query by logical OR operator.

Textual query terms are connected to the final query by logical AND operator so by specifying a text term we can narrow down the results, because each returned one must have the term contained. Specifying more than one text term, these are implicitly connected to the query by OR operator which means they should occur in the result and when at least one does, the score of the document is improved. We can also explicitly say preferences about each text term – whether it must or must not occur in the result.

That said, the final query without explicit stated occurrences of text terms is in a logical form of:

\[ (\text{formula}_1 \lor ... \lor \text{formula}_n) \land (\text{term}_1 \lor ... \lor \text{term}_n) \]

14.1.4. Evaluating

For testing purposes there is a simple demo web interface located at http://nlp.fi.muni.cz/projekty/eudml/mathsearch/index.php. It currently works over 324 060 documents from arXMLiv (XHTML + MathML) project searching over more than 112 milion mathematical expressions.

14.2. Gensim

Gensim is an open-source software for topic modelling and document similarity querying. It offers similar functionality to the yadda-simcat module. The two are, however, API incompatible and written in different languages. For this reason, a wrapper around the gensim library was
created, that mimics the way YADDA similarity is already used in EuDML. The result is called GensimEudml and is described here.

14.2.1. Module overview

Similarity service allows UI to implement the "find similar documents" functionality. GensimEudml aids this goal by providing three main sub-services:

1. convert plain-text documents to their semantic representation
2. index documents in the semantic representation, for faster retrieval
3. for a given query document (either plain-text or an already indexed one), return the most similar documents from the index

These three general functions are realized inside a Python server, which offers several concrete functions through RPC calls. There is also a Java client that can communicate with the server, using YADDA's language (SimilarityFacades and Interfaces and the like).

GensimEudml honours YADDA's choice of multiple indexes, where each language/category pair defines a separate index. Searching for similar documents only happens within the query's respective index. Documents from other indexes are not returned, even if they are similar. See Similarity Service documentation for more information.

14.2.2. Persistency

The GensimEudml component uses the SQLite transactional database to store data persistently to disk. SQLite is serverless (runs embedded within the application process), so it doesn't need to be started or maintained by a separate process. It is also built into Python's standard library, so it doesn't need any installation.

Currently, GensimEudml cannot handle disk failures or power outages during data write (indexing). These may result in corrupt data. More robust dealing with HW failures can be added in the future, if requested.

14.2.3. Concurrency

The GensimEudml server can be accessed by several threads/processes at the same time for queries (read-only mode). Accessing the server in write mode -- indexing new documents, deleting documents, creating semantic models -- is theoretically possible as long as the writes affect different indexes, but is not recommended.

14.2.4. Security and access rights

The module needs write access to the directory where indexes and models are stored. The location of this directory is specified at server start-up as a command line parameter. It doesn't need any special OS user privileges beyond binding >1024 ports and running python.

The RPC communication between client and server poses a potential security threat. If not done over a trusted LAN, an HMAC mode should be enabled, using a shared secret key, by specifying the Config.HMAC_KEY variable (currently commented out in both server and client code).

14.2.5. Installation and deployment

GensimEudml needs:

- 2.5 <= Python < 3.0 (pre-installed on most systems), as the execution environment
• gensim >= 0.8.1 (installation instructions), as the service back-end, LGPL license
• **Pyro** >= 4.8 and Pyrolite (comes bundled with gensim), for network communication, MIT license

Both gensim and Pyro are pure Python packages (require no compilation). However, gensim depends on NumPy and SciPy, two common Python libraries for scientific computing. These exist prepackaged on most systems; see http://www.scipy.org/Download

It is also recommended that the machine where the server will be running on has a fast **BLAS** library installed, to massively improve performance. Options include your vendor’s BLAS library (Intel’s MKL, AMD’s ACML, Apple’s vecLib, Sun’s Sunperf, ...) or some open-source alternative (ATLAS, GotoBLAS). Fast BLAS should be installed prior to installing NumPy and SciPy, so that they can automatically detect it and link against the BLAS during installation.

### 14.2.6. API description

#### 14.2.6.1 Sketch of the API

1. To perform any of the three main functions, the server must be running (see the run_server.sh script in EuDML SVN repository).
   1. Server resumes SQLite indexes from disk (or creates a new database if there is no database yet).
   2. Server goes into daemon mode, listening for requests.
2. Client communicates with the server:
   1. Train semantic model for a given index (language/category pair), using a set of training documents.
   2. Index some documents. Indexing without a trained semantic model results in error.
   3. Find similar documents, using either document id or a SimilarityDocument with plain text as query.
   4. Index some more documents, or delete some documents.
   5. Find similar.
   6. Index/replace/delete.

For details and particulars, see either Similarity Service or the test scripts in EuDML SVN repository.

#### 14.2.6.2 Important file/class/package locations

The server code lives entirely in gensim_server.py. The server itself is run with run_server.sh, which accepts the database directory as its only command line argument. Full access to this directory is required (read/write/delete).

### 14.2.7. Implementation details

- The SQLite wrapper from Python's stdlib is re-wrapped in sqlitedict.py, to add multi-threading capability and to present a more Pythonic, SQL-free interface.
- Client/server (Python/Python or Java/Python) communication happens over TCP sockets, using Pyro for the server and Pyrolite for the thin Java client.
- Gensim doesn't directly support deleting documents from an index (due to way index files are memory-mapped). To allow YADDA-like delete functionality of individual
documents, the deleted documents are only "masked" (marked to be ignored by queries, if
deleted or replaced by document with the same id). This can lead to inefficiencies if there
are massive re-adding/deleting sessions. If this is a problem, a remedy is to drop the
affected index and then re-index its documents from scratch.

14.2.8. Error control

PyroExceptions (communication errors) are caught and re-thrown as SimilarityException, so that
the "throws" API doesn't change. The original exception still appears in the full traceback. Pyro
also supports showing the server-side traceback (Python) on the client when a client call fails,
but the java code doesn't use that atm.

14.2.9. Testing procedure

There is a testing script included in the EuDML SVN repository,
GensimSimilarityTest.java (Java) or gensim_client.py (Python). Both assume
a server is already running (see the run_server.sh script) and proceed to test its
indexing/querying functionality.

14.2.10. Possible extensions

The most similar articles can be either pre-computed (as done in DML-CZ) or computed online,
on-the-fly (done now in GensimEudml). The first option makes more sense with a reasonably
static database and heavy query load. The second is simpler and allows faster indexing. Switching between the two approaches is relatively simple, and can be implemented in the
future, depending on the expected volume of documents and required query latencies.

14.3. BibRefAnalysis

Bibliographic references extraction and analysis tools.

14.3.1. Module responsibility

The module is responsible for:

• extracting bibliographic references from full-text,
• parsing bibliographic references,
• matching parsed bibliographic references with documents in the EuDML collection.

Bibliographic reference matches are presented in the web interface in the form of hyperlinks.

14.3.2. Non-functional requirements met by the module

The module should match at least 10 references per second. It would be very satisfactory if the
performance were in the order of hundreds of references per second. All the tasks in the module
are expected to be performed:

• sequentially (no transactions),
• and as back-end processes (no authorization/authentication).
14.3.3. Implementation details

14.3.3.1 Workflows

The module consists of a number of processing nodes, which are orchestrated into three processing workflows.

The first process (parsing-references.xml) iterates over all the documents, and for each document:

- extracts plain text from the document,
- extracts bibliographic references from the text,
- and parses the references.

All the activities are performed on an as-needed basis, for example, plain text extraction takes place only if no plain text is present for the given document.

The second process (matching-references.xml) iterates over all the documents and for each reference in each document it looks for documents matching the reference. The resulting matches are stored using BibReferenceBrowseRelationManager.

The third process (storing-references.xml) iterates over all the documents containing matched references and for each document it enhances the document's metadata by adding identifiers of matched documents in matched references.

14.3.3.2 Matching algorithm

To quote section 3 of D8.2 (Toolset for Entity and Semantic Associations – Initial Release):

The goal of bibliographic reference matching is to assign to a bibliographic reference an identifier of the referenced document. To achieve this goal, each document and each of its bibliographic references are indexed upon addition to the metadata storage. Following that, for each added document we are looking for:

- bibliographic references matching the document being added;
- documents matching the bibliographic references of the document being added.
In this mode of operation, it is possible to match the incoming documents on-the-fly, as they are incrementally added to the metadata storage. There is no need to recalculate the entire collection after a small update of the storage, such as adding or removing a couple of documents. The faster updates come at a cost, though. The metadata index must now contain not only the documents, but also bibliographic references, so it grows in size by approximately an order of magnitude.

14.3.4. Dependencies and prerequisites

The bibliographic reference extraction and analysis module depends on:

- the storage service (fetching full-text for bib. reference extraction),
- the indexing service (querying for documents in the matching phase),
- the relation browsing service (for storing bib. reference matches).

14.3.5. API description

The module defines three processing workflows which modify metadata of documents stored in the system. The processes are expected to be run by an operator. Thus the module does not provide an API, since it is not meant to be interfaced directly with other modules.

14.3.5.1. Important file/class/package locations

Most of the implementation is located in the `analysis-icm` project, in the `eu.eudml.metadata.enhancement` package. Some classes are also in the `process-nodes`, in the `eu.eudml.process.nodes` package. The key nodes are implemented in the following classes:

- `PlainTextExtractorNode` — extracts plain text from document
- `BibReferenceExtractorNode` — extracts bibliographic references from plain text
- `BibReferenceParserNode` — parses bibliographic references
- `BibReferenceMatchingWriterNode` — stores bibliographic reference matches
- `DirtyItemRecordIteratorBuilder` — iterates over stored bibliographic reference matches
- `EnhancedDataBibReferenceWriterNode` — enhances document metadata with bibliographic reference matches

Some of the above nodes use `BibReferenceRelationManager`, which manages bibliographic reference matches.

14.3.6. Configuration

The module does not require any configuration.

14.3.7. Possible extensions

The extraction and parsing submodules may in the future become part of a larger metadata extraction module.
14.4. PdfJbIm

PdfJbIm is a tool written in Java for (re)compression of PDF documents containing scanned bitmaps. It uses benefits of standard JBIG2 and thus achieves great compression ratios. It reduces the size of PDF documents in average by 30% and if used together with pdfsizeopt they together reduce size by 60%.

It is being developed under AGPL ([http://www.gnu.org/licenses/agpl.html](http://www.gnu.org/licenses/agpl.html)) because of using IText library.

It consists of two parts:

- part written in Java which handles manipulation with PDF documents such as image extraction, replacement of images by their (re)compressed version
- encoder jbig2enc (part written in C/C++) which is run at extracted images and which is modified to achieve even better compression ratio.

![PdfJbIm workflow](image)

**Figure 21: PdfJbIm workflow**

14.4.1. Module responsibility

The purpose of this tool is to optimize size of PDF documents and thus allow faster access to the PDF via the Internet.

14.4.2. Non-functional requirements met by the module

Time to process PDF documents highly depends on number of images in PDF. It is mainly because a global dictionary for equivalent symbols is made through all pages in PDF (images).

Tool contains some handling of encrypted PDF documents but it wasn't much tested therefore it is suggested rather to use already decrypted PDF documents.

14.4.3. Implementation details

PdfJbIm is implemented in Java using two libraries for manipulating with PDF documents:

- PDFBox
- IText
PDFBox is used to extract images out of PDF document and convert them to suitable format for further processing. IText library is used to replace old images by their optimized version (compressed according standard JBIG2). The compression highly depends on the jbig2 encoder.

PdfJbIm uses open-source encoder jbig2enc which is run on extracted images. By default optimizes only bitonal images because JBIG2 standard is designed only for compression of bitonal images. If the image is not bitonal, it can be allowed to compress it as well but with loss of information because image would be transformed to bitonal image.

**14.4.4. Dependencies and prerequisites**

PdfJbIm takes on input a PDF document which is compressed according to JBIG2 standard using jbig2enc encoder

**14.4.4.1 Dependencies**

- Jbig2enc encoder which is used for compressing images extracted from PDF according to JBIG2 standard.
- Libraries for manipulation with PDF documents:
  - PDFBox is used for image extraction. The main reason why PDFBox is used for image extraction is its support of conversion to specific image format.
  - IText is used for replacing images by their compressed version. In iText is support for storing JBIG2 in PDF with support of storing global dictionary in separate PDF object.

**14.4.5. API description**

PdfJbIm takes PDF file as input and returns (re)compressed PDF file with reduced size. PDF files can be put either as files or as streams (Java streams - InputStream and output returned in OutputStream).

Using pdfJbIm inside another API consists of four main steps:

1. image extraction
   - PDF as input can be given as InputStream, File or String (representing input PDF file)
   - while extracting image informations and names of extracted images is stored
2. running jbig2enc
   - setting parameters for running jbig2enc
   - running jbig2enc with set parameters and images extracted from PDF
3. associating output of jbig2enc with information about images
   - output of jbig2enc is created in working directory
   - based on given basename output of jbig2enc is loaded to memory
   - output is associated with image information
4. replacing images in PDF by their (re)compressed version
   - images are put back to PDF file

These steps are in more detail described in README file in the EuDML SVN repository under usage part.

**14.4.6. Configuration and Requirements**

Libraries for manipulation with PDF documents (see maven pom.xml):
• PdfBox
• Itext
• Encoder jbig2enc

For getting full functionality (even the modification which improves jbig2enc) is necessary to use modified version of jbig2enc

It is possible to download from http://code.google.com/p/pdfrecompressor/, there is also available modified version of jbig2enc encoder

For installing encoder jbig2enc:

• Installation of libraries for handling multiple image formats (i.e. libpng, libjpeg, libtiff, libgif, zlib, libz) and the Leptonica library + its dependencies (see http://leptonica.com/source/README.html#DEPENDENCIES) -- these are libraries for handling image I/O
• library Leptonica
• to compile jbig2enc there is necessary either to compile Leptonica library in (parent) directory relevantly pointed as ../ or set the variable in Makefile (second row) LEPTONICA to location where Leptonica is
• Makefile is set to use of Leptonica library (version 1.67)

14.4.7. Possible extensions

• Combine detection of equivalent symbols with information accessed from OCR to improve even further the compression ratio
• Add support to (re)compress images which were already compressed using JBIG2 standard - it is in progress.

14.5. PdfToTextViaOCR

PdfToTextViaOCR is a tool written in Java which extracts images from PDF and extracts text from them using OCR (currently used Tesseract);

It is being developed under Apache Licence 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

It consists of two parts:

• Image extraction written in java
• OCR engine which is run from command-line

14.5.1. Module responsibility

Renders text out of images to allow indexing and searching in PDF documents containing only scanned text.

14.5.2. Non-functional requirements met by the module

Which images can be extracted out of PDF documents highly depends on PDFBox library. PDFBox doesn't support extraction of images consisting of more than one object.

Doesn't automatically recognize language of the document.
14.5.3. Implementation details

PdfToTextViaOCR renders images from PDF using PDFBox library. It accesses objects in PDF document from the root object to image objects which are then extracted and converted to suitable format for further processing.

For each rendered image Tesseract\(^1\) is run with defined language of the document (if not set, than used English as default language). It is suitable to use at least Tesseract version 3.0 because of big changes in language support and format of language dictionaries.

Because Tesseract is run using Java Runtime API, output is stored in files which are than parsed and result text is concatenated from them.

14.5.4. Dependencies and prerequisites

- Library PDFBox for PDF image extraction
- OCR engine (Tesseract) which is used for text extraction out of rendered images

14.5.5. API description

PdfToTextViaOCR takes PDF file on input either as a Java InputStream or file and returns either set of files containing text or directly the text.

14.5.5.1 Sketch of the API

Using pdfToTextViaOCR consists of two main steps:

1. image extraction
   - extracts images from a PDF and returns list of images
2. running Tesseract
   - runs Tesseract with appropriate parameters such as language of a document and returns list of files containing OCRed text

These steps are in more detail described in README file in the EuDML SVN repository under usage part.

14.5.5.2 Important file/class/package locations

- Class Tesseract allows to set parameters which shell be used while running Tesseract on rendered images such as language of the text in the image.
- Class Run shows how the PdfToTextViaOCR can be run.

14.5.6. Configuration and Requirements

- PdfBox -- for extracting images out of PDF
- Tesseract -- extracts text from images (the effectiveness depends also at correctly set language of a document)

14.5.7. Possible extensions

Adding a step to the workflow to recognize the language of a document.

\(^1\) http://code.google.com/p/tesseract-ocr/
15. Deployment of EuDML

15.1. Required resources

- A servlet container (Tomcat 6 is typically used) for the backend service
- A servlet container (Tomcat 6 is typically used) for the user interface
  - Should be different of the backend one because of startup-time dependencies
- A PostgreSQL server with databases for the repository and the User Directory Service
- YADDA CLI tools for running import and enhancement processes
- Disk space for search indexes and YADDA Archive storage

15.2. Example configuration for the backend

When using Tomcat the following context definition can be used to provide configuration properties for the backend server.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/backend">

<!-- common base directory for the repository -->
<Parameter name="eudml.data.dir" value="/srv/eudml/backend"/>

<!-- common db settings -->
<Parameter name="eudml.db.url" value="jdbc:postgresql://localhost/eudml_backend"/>
<Parameter name="eudml.db.username" value="eudml"/>
<Parameter name="eudml.db.password" value="${EUDML_DB_PASSWORD}"/>

<!-- user db settings -->
<Parameter name="usercatalog.db.url" value="jdbc:postgresql://localhost/eudml_users"/>
<Parameter name="usercatalog.db.username" value="${eudml.db.username}"/>
<Parameter name="usercatalog.db.password" value="${eudml.db.password}"/>

<!-- url for archive data retrieval published to clients -->
<Parameter name="yadda.archive.contentServerRoot" value="http://localhost:8080/backend/part/"/>
</Context>
```

15.3. Example configuration for the User Interface

When using Tomcat, the following context definition can be used to provide configuration properties for the user interface.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ui">

<!-- ui.mail.hostLogin -->
<Parameter name="ui.mail.hostLogin" value="/"/>
<br />Parameter name="ui.mail.hostPass" value="/"/>
<br />Parameter name="ui.mail.hostIP" value="localhost"/>
<br />Parameter name="ui.mail.hostPort" value="25"/>
<br />Parameter name="ui.applicationRootUrl" value="http://demo.eudml.eu/ui"/>
<br />Parameter name="ui.backendRootUrl" value="http://localhost:8080/backend"/>
</Context>
```