



Science Created by YOU

SCY Repository DIV.5

Authors

*Jan Engler (UDE), Stefan Weinbrenner (UDE),
Sven Manske (UDE)*

Science Created by You (SCY)
(Project number IST-212814)

Date: 28-02-2011

Dissemination level:

<input checked="" type="checkbox"/>	PU	Public
<input type="checkbox"/>	PP	Restricted to other programme participants (including the Commission Services)
<input type="checkbox"/>	RE	Restricted to a group specified by the consortium (including the Commission Services)
<input type="checkbox"/>	CO	Confidential, only for members of the consortium (including the Commission Services)

© 2011, SCY consortium

Page intentionally left blank

Executive summary

This document presents the technical basis and the design decisions of the SCY repository (Repository of open learning objects – RoOLO). Important features and implementations are discussed.

There are several details that are worth to examine in more details. Especially the underlying technology and the search mechanism are presented. As the different ways to access to the repository are interesting for everybody, who will develop internal or external tools, this deliverable documents the usage and shows examples.

In a previous deliverable, DVI.1, a preliminary design for the SCY repository was presented. We will describe why and how we changed that proposal. Especially the performance problems are solved with the current implementation.

The SCY repository is integrated in the SCY server architecture and ready to use in SCY.

Table of Contents

1	Introduction	5
2	RoOLO Requirements	5
2.1	Implications of the ELO Idea	5
2.2	Rights Management.....	6
2.3	Versioning	6
2.4	Handling of metadata	7
2.5	Searching in SCY	8
3	Frameworks	9
3.1	Storage.....	9
3.2	Search	11
4	Accessing RoOLO	12
4.1	Internal Access	12
4.2	External Access	13
4.3	Access for Agents.....	15
4.4	SCY-specific and generic Access	15
5	Search	16
5.1	Conceptual Issues	16
5.2	Technical Aspects	17
6	Implementations	18
6.1	Roolo-FS	19
6.2	RoOLO-JPA.....	19
7	Conclusion	19
	References	20

1 Introduction

This document presents the implementation of the SCY repository (Repository of Open Learning Objects – RoOLO). Some important discussions and decisions are described in a more detailed way. Especially the requirements of the ELO idea for the repository are discussed.

As the access to the repository is a crucial part of the implementation, several approaches to communicate with RoOLO are presented.

Another important part of the repository is the search facility. We will discuss our decisions and present the underlying technology.

Content:

Section 2 gives an overview about the requirements to the repository based on the design of ELOs that is described in DI-1. It outlines some elementary features that are important and discusses some decisions that have been made during the design and implementation process. Section 3 presents the underlying technical frameworks that have been used in RoOLO. Section 4 describes the several methods to access to repository from within SCY-Lab, from agents and from external tools. Section 5 presents the search facility inside the repository and the search interface. Section 6 gives an overview about the existing implementations (with different techniques) of the Repository interface. Section 7 gives a conclusion.

2 RoOLO Requirements

The concept of ELOs (Emerging learning objects) is one main pedagogical idea in the SCY project. These ELOs will be created by the learner and they will be changed during their lifecycle, not only by the creator but also by collaborating partners. The repository of open learning objects (RoOLO) is a central component in the SCY architecture as all ELOs that are created within SCY will be stored here. The changes during the lifecycle of ELOs should be traceable, so that the complete process of creating and emerging of the learning objects can be inspected by researchers. Furthermore, the repository has to provide a robust and fast access to the stored ELOs to allow for reuse and sharing of these artefacts. This dynamic nature of ELOs and the central role of the repository in SCY have implications on the design of the repository. In this section our approach of a design is described and some special decisions are explained more deeply.

2.1 Implications of the ELO Idea

The emerging character of ELOs is a central pedagogical part of the SCY idea. An ELO consists of two main parts, the content, which represents the payload of the learning object, and a metadata section, which represents all useful additional data about the ELO and the situation, in which the ELO was created. During the lifecycle of an ELO, both sections will alter. This process of emerging should be observable to researchers as this is important for analysing the learners' progress. This progress could then be visualised and displayed to a teacher. Another argument to have a version history of every ELO is that the learner can revert his or her work to a former state.

To allow that, every update of an ELO will create a new version that is stored separately in RoOLO. This idea of versioning will be explained in a more detailed view in Section 2.3.

Another challenge in the design of a repository is the heterogeneous structure of ELOs. Depending on the type of the content of a specific ELO, the metadata will differ. The repository must be able to deal with the multi-facetted structure. The implications of that structure will be presented in this section as well.

2.2 Rights Management

One question in the design of the repository is the location of a rights management concerning the access to the stored data. A learner will create ELOs that are public but there has to be the possibility to mark an ELO as private and therefore only readable by the creator. We investigated two different approaches to implement the rights management in RoOLO. The first proposal was to integrate the handling of access to the specific ELOs inside the repository. Whenever a user attempts to search or to retrieve an ELO, RoOLO would check if the user is allowed to access the ELO. If so, the ELO will be given back to the user; otherwise an “Access denied” message would be displayed. The second approach would handle the rights management inside the SCY-Tools (or the SCY-Lab), as these components know about the currently logged in user and can decide whether to display the ELO to the user or not.

The repository as the central storage for all ELOs would argue for the first solution. The tool developer would not be bothered with checking for the permission of the user. However, this solution bears one general drawback: the repository has to provide a session handling. As we do not want to replicate the session information of SCY-Lab inside the repository, we decided to place the “rights” check inside the tools and the search interface of SCY-Lab. That solution allows not only tools to decide whether to show a result to the user or not. Additionally, we donot need an “admin” concept for components like the agents, which are not directly connected to a specific user but must be able to search and retrieve every ELO.

2.3 Versioning

As described above, one requirement is a versioning system that is based on the ELO idea of SCY. A naive approach would be to store every change as a new ELO inside the repository. This version will then be linked to the predecessor through an entry in the metadata section called “*versioned_by*”. While this is a good solution for every change in the ELO that really affects the content, there are some changes that will only change a metadata entry. An example for this is the enrichment of the keyword metadata entry by an agent. After a learner saved an ELO, an agent will extract important keywords from the content and will add them to the ELO (see DV.3). It is obvious that this enrichment does not affect the content of the ELO. In this example case, it is not only inconvenient to have a new version after the agent-based enrichment, it would lead to another problem: The user that saved the ELO would not work on the latest version anymore and will therefore not be able to save this ELO again as he or she is obviously only able to change the latest version to prevent conflicts between two versions. It is not possible to overwrite versions in the repository as this would possibly destroy the learners work.

To prevent this problem, we integrated two basic methods for the RoOLO:

- *updateELO* – This method will perform an update including a new version in the repository.
- *updateELOWithMinorChange* – This method only updates metadata of the ELO and will not create a new version in the repository. The latest ELO's metadata will be updated and stored in the same version as before.

As *updateELOWithMinorChange* is only meant for updating the metadata (in most cases the change will only affect one metadata entry), a change of the content is not allowed here. The overwrite issue does not appear with minor changes as the metadata can be overwritten by a new version.

The separation of the update function in these two ways leads to another problem: When the learner saved an ELO and the agent in the backend enriches the ELO, how will the learner be notified about that change? Maybe the added information is important to know for the learner, e.g., for displaying the actual keywords. During the implementation of RoOLO, two possible solutions for that problem emerged; the first solution would be to change RoOLO in that direction, that it can actively notify the user¹, if there was a change in the ELO he or she is working on.

The second solution is a “fingerprint” method inside the repository that calculates a unique value based on the ELO (content and metadata). This fingerprint is stored as a metadata entry inside the ELO. A component (like a tool or the SCY-Lab) is now able to compare the fingerprint of the ELO that is opened with the calculated value of the “same” ELO that is stored in RoOLO. If the two values differ from each other, the tool could retrieve the latest state of the ELO and display the changes to the user.

As the implementation of RoOLO as an active component would lead to a number of changes as an own backchannel to the tools, we decided to implement the second solution.

2.4 Handling of metadata

As described above, an ELO in SCY consists of two main components: the content and the metadata. We decided to make use of the LOM standard [LOM] for our metadata, but of course there are some metadata keys that are not defined there. To handle the different metadata types and to make the access to the metadata values more unified, we implemented a component called *MetadataTypeManager*. During the startup of the repository, this component is configured with two files called *CoreMetadataKeys* and *CustomMetadataKeys*. The core keys consist of a set of basic metadata keys that are mandatory for every ELO. The custom keys define metadata keys that are SCY specific. Examples for core metadata keys are:

- *identifierKey* (e.g., “roolo://scy.collide.info/scy-collide-server/150.150#0”)
- *titleKey* (e.g., “Problem presentation”)
- *descriptionKey* (e.g., “The problem description for the eco mission...”)

¹To be correct: the tool, in which the learner is working on an ELO, will be notified, not the user directly, as there would not be any direct communication between the repository and the user.

2.5 Searching in SCY

Another core functionality of a repository in SCY is the search facility. As we want to force the concept of reuse, there is a need for a fast and easy-to-use search mechanism. The technical basis for the mechanism must be provided by the repository. The technical details of this mechanism will be explained in Section 3.2 and Section 5 of this document. Additionally to the server-side searching facility, we decided to have a central searching tool inside SCY-Lab. A search in SCY-Lab can be a simple “Google-like” search. A learner can just enter one or more keywords and the results will be presented to him or her as a list. Another way to search for ELOs is the advanced search. Here, a learner can specify multiple facets of the object he or she is searching for by defining metadata in detail. For instance he or she can specify a search like: “*author is bob and type is concept map*”. Of course, the learner does not have to enter a search string like the example. As you can see in Figure 1, there is a graphical interface for the search.

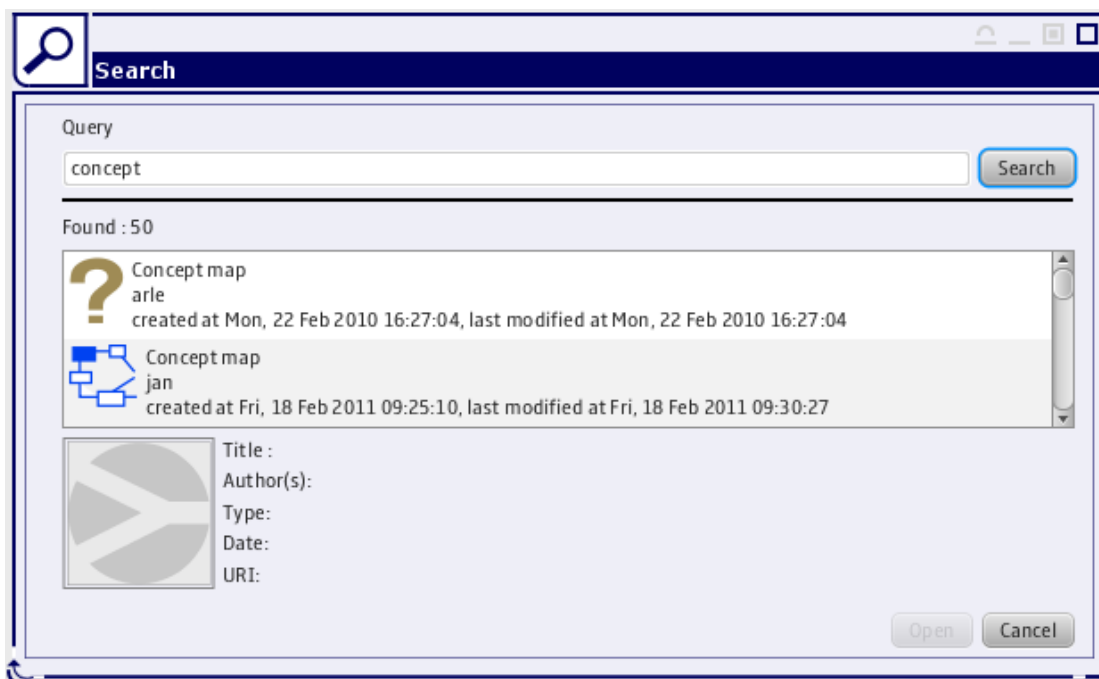


Figure 1: Basic search interface

A third way to search for ELOs is to make use of the ELO-centric search. A learner just drags an ELO that he or she created or opened earlier over the search button in SCY-Lab. A new window is opened with several options like “find ELOs of the same author”. Figure 2 shows this search facility inside SCY-Lab.

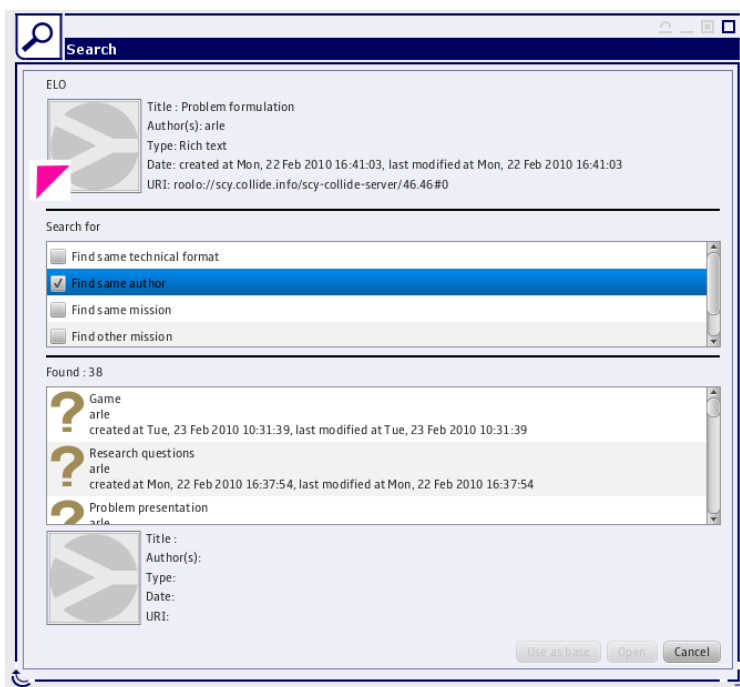


Figure 2: The ELO-centric search in SCY-Lab

3 Frameworks

3.1 Storage

One important decision for the design of the repository is to determine, which underlying technology should be used to persist the ELOs. The technology should be hidden from the user and tool developer as they will only communicate through the RoOLO API.

Problems with JCR

As described in DVI.1, the initial plan for the underlying technology for RoOLO was to use the Java Content Repository API (JCR)². One advantage using JCR is that it has a built-in versioning system and a Lucene³-based search facility, which are, as described in Section 2, essential parts of the requirements for RoOLO. But alongside this advantages a few problems emerged during the implementation of a JCR-based prototype. It turned out, that there was a need for an object mapper to map the complex structure of ELOs to the repository. Two alternatives were discussed in that stage of RoOLO:

- Using JCROM (JCR object mapper)⁴ as the mapping facility for ELOs
- Mapping the ELO objects manually to the tree structure of JCR

As JCROM seemed a promising approach, we started using this technology. Unfortunately, in the first experiments with a JCR/JCROM-based repository, it turned

² Java Content Repository: <http://jcp.org/en/jsr/detail?id=170>

³ Apache Lucene: <http://lucene.apache.org/>

⁴ JCR object mapper JCROM: <http://code.google.com/p/jcrom/>

out, that the performance of JCROM for complex and deeply nested objects like ELOs is not sufficient for our purpose. The basic technical structure of ELOs is presented in Figure 3. Especially the type-specific parts of the content and the metadata caused a lot trouble with JCR/JCROM in relation to performance. While the upper level structure is the same for all ELOs, the type-specific parts on the bottom level (Binary/XML) vary from ELO to ELO, which prevents from solving the problem with a generic approach.

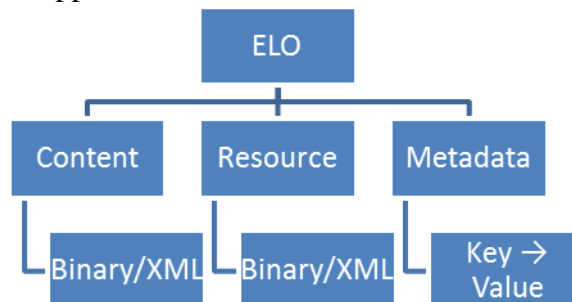


Figure 3: The basic structure of an ELO (Emerging Learning Object)

As the concept of reuse is a main target of the ELO idea, there is a need for a fast and scalable searching and retrieving mechanism. In the production stage of SCY we think of having thousands of ELOs inside the repository. The learners must be able to access the stored ELOs simultaneously. Combining the JCR implementation JackRabbit⁵ (the implementation of the JCR API that we used) and the JCROM object mapper with the concept of ELOs was far too slow for our requirements. The second idea of mapping the ELOs manually is contradictory to the approach of having a generic repository. As we do not know the type of the content or the metadata, the only way to store the objects in this case would be to serialise the ELOs as binary data, which is obviously not what we wanted.

JPA as the storage platform

Due to the disadvantages described above, we decided to implement a prototype of RoOLO using the JPA (Java Persistence API) specification. JPA is an interface to map and persist Java objects inside a relational database like MySQL. There are several implementations of JPA like EclipseLink⁶ and Hibernate⁷. We decided to use Hibernate, because there have been prior knowledge and good experiences in the SCY project with this technology.

JPA has some advantages compared to JCR and the most important for the SCY project will be explained here.

In JPA every object that should be persisted is annotated in the source code as an *entity*. This is very helpful as we were able to reuse our existing ELO implementation and enrich it with those annotations without having to re-implement parts of them from scratch. The same was true for the ELO API, which is very important, because this API is used heavily in SCY. The problem with complex and nested structures as we experienced with the JCR approach did not appear with JPA anymore. Moreover,

⁵ <http://jackrabbit.apache.org/>

⁶ <http://www.eclipse.org/eclipselink/>

⁷ <http://www.hibernate.org/>

JPA can be used to store multi-level structures like ELOs in a very straightforward way. The most important advantage for the SCY project of JPA over JCR is the performance. Compared to our RoOLO prototype with JCR as the core technology, the JPA-based prototype was 10-15 times faster.

Of course, there are some disadvantages as well. The most important is that JPA does not have a built-in versioning management as JCR has. During the investigation for versioning, we found a promising project called Envers⁸, which is built on top of JPA. Envers is able to handle the versioning, but as described in Section 2.3, we have a concept of minor changes, which was not supported by Envers, yet. Nevertheless, we decided to use JPA for the implementation of RoOLO. We created our own versioning system inside RoOLO, which was able to handle all the requirements of the SCY ideas.

Another disadvantage compared to the JCR prototype of the SCY repository was the search facility. Hibernate (as the implementation of JPA that we are using) does provide a built-in Lucene support, but as we want to add additional information like weights in the index (see Section 5), we have chosen to implement our own Lucene wrapper around the JPA-based repository. It turned out that we could reuse most parts of the existing Lucene interface of the JCR prototype, which cares about the indexing and searching facility for RoOLO.

All in all, we found with JPA a decent alternative for JCR that help us to overcome the performance issues that we had with the combination of JCR and JCROM. Figure 4 shows the current, multi-layered structure of RoOLO with the JPA approach.

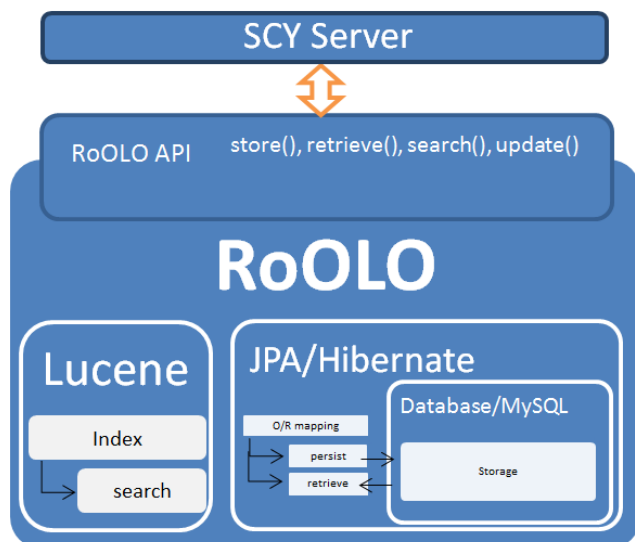


Figure 4: The multilayered RoOLO approach

3.2 Search

As described in 3.1, we are using Lucene as the underlying technology for the search mechanism in the SCY repository. Apache Lucene is a free software library facilitating the basic components of information retrieval systems consisting of an

⁸ <http://www.jboss.org/envers>

indexer and the query engine. As the index written by Lucene is optimised, it needs software like e.g., Luke⁹ for investigation. *Luke* is not only able to show the documents that are stored in the index; it also provides access to the Lucene API and a sandbox for testing different analysers, query formulation and parsing, etc. See Figure 5 for a screenshot of *Luke*.

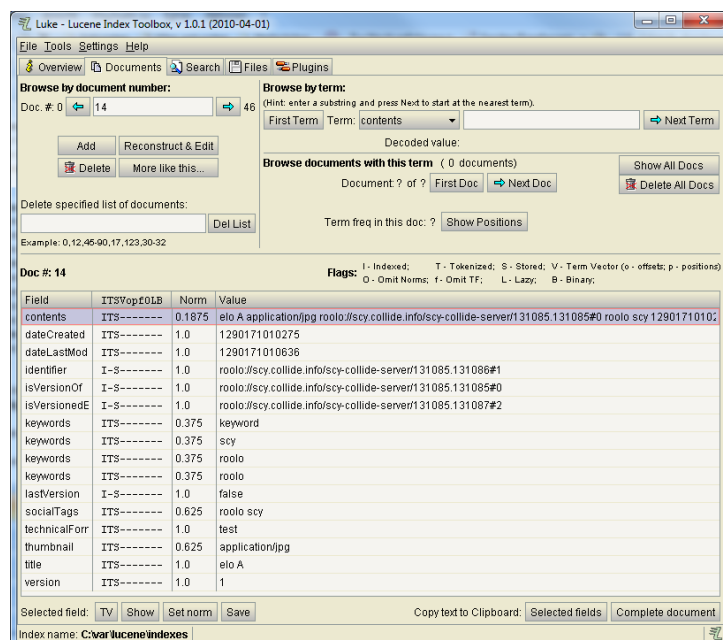


Figure 5: Luke displaying a document representation of an ELO in RoOLO

As Lucene is a state-of-the-art information retrieval library, it follows the concepts of modern information retrieval systems. In Section 5 we will give more details on the SCY approach of using Lucene.

4 Accessing RoOLO

As RoOLO is a core component in the SCY server, it must be accessible in a convenient way by all the components inside and outside SCY-Lab that are used in the SCY project. Due to the heterogeneous structure of the different parts like web tools, SCY tools, agents and mobile devices, there must be several ways to communicate with the repository. In this section these ways are explained.

4.1 Internal Access

To provide the SCY-Lab (and hence the SCY tools) with a connection to the SCY repository, we are using Spring¹⁰ to integrate the repository. Spring is an open-source framework for Java, which allows making use of a technique called “dependency

⁹Luke: <http://code.google.com/p/luke/>

¹⁰Spring: <http://www.springsource.org/>

injection”. Usually, in the world of object-oriented programming, a system will create and initialise its dependencies on its own. Therefore, the system must know about the concrete implementation of these objects. By using the Spring Framework, the dependencies can be configured independently of the system, which only knows about the interfaces of the objects. Especially in the case of multiple implementations of an interface (like for RoOLO, see Section 6), this is very handy as you can easily exchange the concrete implementation.

SCY-Lab gets the repository reference through a module called tool-broker (as described in DVI.1), which is the target for the injection of all remote dependencies like the repository or the action-logging facility. The remote communication is handled by another component of the Spring Framework, the HTTP invoker. This component provides a remoting strategy for objects serialisation and deserialisation via the HTTP protocol.

Using these techniques leads to an easy-to-use and robust communication between the server-side, where the repository resides, and the SCY-Lab and the corresponding tools.

4.2 External Access

As seen in Section 4.1, the internal access to RoOLO benefits from getting references to Java objects. For external tools, e.g., tools written in JavaScript, this is not possible. These components need another, more loosely coupled way to access RoOLO. A commonly used way to connect external applications to a server-based infrastructure is using a webservice.

Webservice

A webservice is a common way to provide services via the web. For this kind of access, we implanted a component namely SCY-RoOLO-WS. This service follows the REST (**R**epresentational **S**tate **T**ransfer) approach. This architecture style is a standard for the communication between a server and a client and is used to address and transfer remote objects.

For the data interchange with the SCY webservice, the JSON format is used. This format is a common alternative to XML for communication between web server and client, being more compact.

For accessing RoOLO, the service wraps an instance of the repository passed via Springremoting (which is a RPC-like technology that is also provided by the Spring Framework), so it is loosely coupled to RoOLO itself.

The webservice consists of three services for the basic operations storing, retrieving and searching. They differ in required and optional fields for the JSON data that is submitted when calling a service via HTTP POST.

The following enumeration shows the different services with their mandatory and optional fields.

1. /saveELO

a. Required parameters:

content: the content of the ELO

username: username for authentication

password: password of the user for authentication

language: language of the ELO (ISO-639)

title: title of the ELO

type: technical format of the ELO, e.g., "scy/webresourcer" for SCYLighter/Webresourcer ELOs

b. Optional parameters:

uri: the URI of the ELO to update. If specified, not a new ELO will be created, but an existing one will be updated

country: country code (ISO-3166)

description: a short description of the ELO

dateCreated: The date of ELO creation - if this is not specified, it will be created.

c. Result:

A plain text containing the URI of the saved ELO.

2. /getELO

a. Required parameters:

As the URI is the only required information for retrieving an ELO, JSON is not needed. This service consumes just the URI as a plain text.

b. Result

The service returns a JSON object containing all relevant data, which is named like the ELO. Metadata values are in fields that are named after the corresponding metadata keys. The content of the ELO is stored in a field "content", the resources in a field "resources".

3. /search

a. Required parameters:

Searching for ELOs requires a query formulation. As the external tools are inhomogeneous, a simple solution for that query formulation is the use of a string-based query language

b. Result

The search service consumes the query string as plain text and produces a JSON object wrapping all attributes of the search result from the Java object, e.g., the URI, author and title.

As an example for the usage of the webservice for external tools, SCYLighter (see DII.1, Web browsing Tool) is presented. SCYLighter is an extension for the Mozilla Firefox¹¹ browser, written in JavaScript and XUL¹², which can be used to highlight sections on a webpage. These sections can be used to create an ELO with the text or images as its content. That ELO can be displayed and edited inside SCY-Lab later on. When SCYLighter is saving the ELO, the webservice is called to store the ELO inside RoOLO and returns the URI. This URI can then be used to update the ELO on the next save event of SCYLighter.

4.3 Access for Agents

As described in Sections 4.1 and 4.2, there are several ways to access to repository through the TBI or using the webservice. Other parts of the SCY server are the software agents. Some of these components have to access RoOLO as well. One example for the necessity for the RoOLO accessing agents is the keyword enrichment agent. This agent recognises that a learner saved an ELO and enriches that ELO with extracted keyword and updates¹³ this ELO in RoOLO. To get access to the repository, an agent has to implement an interface called *IRepositoryAgent*. This agents will get the repository injected using the Spring Framework during the startup of the SCY server.

Another way to access RoOLO is the usage of the *RooloAccessorAgent*. This agent is listening to the TupleSpace (see DVI.3) and processes request tuples from other agents that are not able to access the repository directly but want to retrieve an ELO. Once the *RooloAccessorAgent* is triggered, it will retrieve the specific ELO and send it back to the querying component as serialised XML. This form of agent access is especially important for those agents, which are not using RoOLO heavily.

4.4 SCY-specific and generic Access

During the first planning phases for the SCY repository, we decided that RoOLO should not be fixed to the SCY design but should be a repository for any ELOs that could also be used outside of SCY.

It turned out that the generic approach of RoOLO is very good for developing different ELOs with different metadata. One can easily add new types of metadata and register them at runtime. On the other hand, this generic approach has one drawback: For the tool developers in SCY it is not very handy to retrieve the values of some metadata-keys. One example is the access to the author of an ELO. To get this information a developer has to use the *MetadataTypeManager* (as described in Section 2.4) and has to know about the name of the key that is used for the author. As this is not very convenient, we developed a layer on top of the generic ELO called *ScyElo*. This layer allows retrieving commonly used metadata values by using get-methods. The *ScyElo* wrapper provides all core metadata keys that can be accessed very easily. To get ELO specific metadata that are not defined in the core metadata keys, a developer can of course use the standard mechanism using the *MetaDataTypeManager*.

¹¹ Mozilla Firefox: <http://www.mozilla.com>

¹² XUL is an acronym for XML User Interface Language. XUL is a declarative markup language to develop graphical user interfaces especially for Mozilla products.

¹³ As described in Section 2.3 this agents uses the update with minor change functionality.

A similar wrapper was developed for the repository itself. As the generic approach of the repository is very flexible and extendable, a layer on top of it helps the SCY developers to access the RoOLO more conveniently. According to the ScyElo this RoOLO wrapper is called *ScyRoolo* and provides besides the standard functions like store, update or search, a built-in cache and the automatically executed fingerprinting to detect minor changes in the repository as described in Section 2.3.

5 Search

Reusability is a core requirement of ELOs, which can only be achieved if the learner can access own or other ELOs easily. This might be done passively by recommendations generated by agents or more actively by searching in the repository for specific ELOs. This section will present the conceptual issues and the technical aspects of a searching mechanism inside RoOLO.

5.1 Conceptual Issues

To satisfy the learners need for information and to enforce the concept of reuse, he or she must be able to query the system for ELOs in a proper and convenient way. The formulation of the query is one conceptual issue of the process of searching. The Berrypicking model described in [Bates89] understands this process as a sequence of queries which involve an evaluation of the ELOs' relevance by the learner and a reformulation of the query to improve the subjective relevance (pertinence). As the amount of ELOs stored in the repository might be huge, it is important to pre-sort the results by a system-oriented relevance.

Being a time-critical operation, searching should happen in an optimised infrastructure, which is provided by the Lucene Framework (described in Section 3.2).

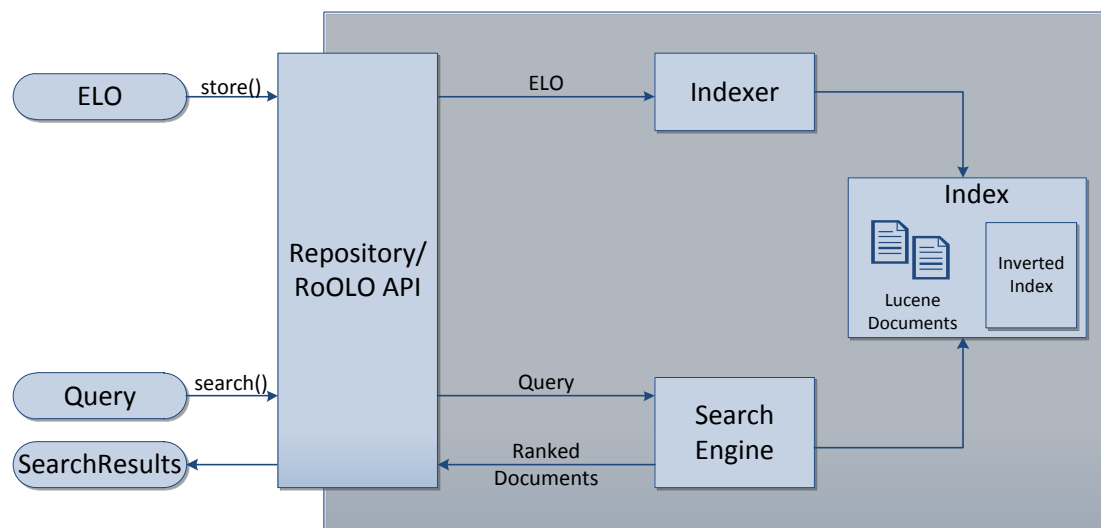


Figure 6: the conceptual design of indexing and searching in RoOLO

Indexing

When RoOLO is triggered to perform a search on the repository, the search is not performed on the ELO entities itself. For faster access to the data, an index is queried for the given terms. If one would search the terms “CO2” and “mission”, querying the

index is much faster than directly working on the ELO entities. The results of the query are the conjunction of both document sets from this inverted index. To obtain all relevant information the index does not only contain the inverted index but also all document representations of the ELOs. This will lead to a bigger index, but we think that performance is more important than the index size, because performance is perceived by the learner and is an important factor regarding the acceptance of the software. On the other hand, memory is not that expensive anymore.

To build up the index, every store and update operation must be intercepted and the ELO must be transferred into a Lucene document. The steps for that are:

1. Storing a document representing the entity and
2. Inserting all terms of the document into the inverted index

Figure 6 shows the complete process of storing ELOs in the repository and adding them to the index. The process of querying and retrieving the results is presented here.

Searching

The process of performing a search with a specified query triggers the search engine to calculate a score of documents, which are basically similarities of each document representation to the query representation, which is a common approach in the Vector Space Model of information retrieval.

Query formulation

For query formulation there are two types of query language: one object based query language providing typesafety and a query language for natural string queries as used in Lucene¹⁴ Both are parsed by the RoOLO query parser and are translated in native Lucene queries. The usage of two query languages solves the trade-off between precise formulation and simplicity. Another advantage of having both approaches is that an object based query formulation is much easier to parse for search components like a query expansion agent as described in DIV.3.

5.2 Technical Aspects

In Section 5.1 we discussed the conceptual ideas of a search and the requirement to the facility. In this section we explain the basic ideas of the underlying technology.

Indexing

When storing an ELO in RoOLO, the indexer will be triggered to analyse these entities to create a Lucene document as described in Section 5.1. Every document is a Lucene-based representation of one ELO. It consists of fields corresponding to the metadata keys, e.g., a field “author” containing the author string from the metadata.

Lucene specifies a default field, which is used when querying without naming a field to search at.

For enabling a “Google-like” search, it is not feasible to specify fields for search. A good approach would have been to search in all fields, but as Lucene is only capable of this feature with a lot effort and the size of the index is not that important in our case, every metadata value of an ELO is stored in a special field called “contents”. This is also the default field used for search without field specification, so that a

¹⁴ Lucene Query Language:

http://lucene.apache.org/java/2_0_0/queryparsersyntax.html

search string “co2 scy” will force the search engine to take a look at the default field called “contents” and calculate a scoring for every document fitting to this query. The complete index is a conjunction of all documents. This index can be investigated with special tools. A perfect suited tool for our purpose is the already mentioned Luke application.

Ranking

When performing a search, the user wants to retrieve relevant results rather than non-relevant. A common approach is a ranking of search results based on the system relevance to approximate the subjective relevance of the user. The ranking is a list of documents ordered by the score of each pair (d,q) of document and query computed by the Lucene Scoring Formula:

$$\text{score}(q,d) = \text{coord}(q,d) \cdot q\text{Norm}(q) \cdot \sum_{t \text{ in } q} (\text{tf}_{t,d} \cdot \text{idf}_t^2 \cdot \text{boost}(t) \cdot \text{norm}(t,d))$$

- $\text{coord}(q,d)$ is a score factory reflecting the fraction of query terms found in the document.
- $q\text{Norm}(q)$ normalises queries to make them comparable. This does not affect the ranking of documents.
- $\text{tf}_{t,d}$: The term frequency is the count of a term t in the document d . $\text{tf}_{t,d}$ correlates to the value and is calculated by default: $\text{tf}_{t,d} = \sqrt{\text{frequency}}$.
- idf_t : The Inverse Document Frequency lets rarer terms contribute in a more impacting way to the score and correlates to the inverse of the document frequency describing the number of documents, in which the term t occurs.
- $\text{boost}(t)$ describes a boost factor specifying the importance of the query term t , by default 1.
- $\text{norm}(t,d)$ is a normalisation factor containing length normalisation, document and field boost, which are all calculated at indexing time in contrast to the preceding factors.

This calculation of score is similar to the cosine-distance between the document and query vectors in a Vector Space Model containing some specific normalisations and boosting of terms and documents. This can be useful for implementing a more efficient ELO search, where the “author” field is in most cases more important than the “*versioned_by*” metadata field.

6 Implementations

During the design of RoOLO, multiple implementations were developed. As described in Section 2 our productive implementation is JPA-based and we are using Lucene as a framework for searching. Another implementation, which is mainly used for local installations and for testing purposes, is the filesystem-based RoOLO-FS.

In this section an overview of these two implementations is presented.

6.1 RoOLO-FS

In the beginning of the SCY project, there was a need for a working repository for ELOs. Therefore we decided to implement a “Mock” version of the repository API. Starting with the major methods like store, update and retrieve, the extended functionality like searching was not implemented. During the implementation of RoOLO-JPA we decided that we could need a simple and easy to configure RoOLO as an addition for RoOLO-JPA. We integrated a fully working search support (also based on our Lucene implementation for RoOLO-JPA) and renamed this repository to RoOLO-FS. It is a basic but full featured implementation of the repository API. It is not relying on a database as the ELOs are stored directly as XML data on the local filesystem of the server. While the core implementation needs a configured SQL server installed, RoOLO-FS runs autonomously. RoOLO-FS is not optimised for performance but is fast enough especially for testing and authoring purposes. During the process of authoring a mission, a designer needs a storage for the ELOs that he or she is creating. These created ELOs can later be imported into the productive JPA-based RoOLO.

6.2 RoOLO-JPA

In contrast to RoOLO-FS the core implementation of RoOLO is dependent on a configured SQL server on the machine RoOLO is running on. Therefore, it is running much faster as the database is optimised for fast access and even more important, it is scaling very well. The internal structure and the workflow inside the JPA-based RoOLO are explained in detail in Section 3.1.

7 Conclusion

The implementation of the SCY repository has been described in this deliverable. Especially the requirements for the repository that came from the ELO idea in the SCY project have been presented. The different ways of accessing the repository and the flexible search API shows different usage scenarios of RoOLO.

During the development of the SCY repository we evaluated several different technologies that could be a basis for a later implementation. Based on those technologies, we implemented three major prototypes: JCR-based RoOLO, RoOLO-FS and JPA-based RoOLO. We discarded the promising approach of using JCR as the core technology for the repository as it was not sufficient in terms of performance. The JPA-based repository solved this problem and will be the final RoOLO in the SCY project.

Some conceptional problems arose in the development. Especially the versioning / updating issues are worth to mention here. These problems are solved in a (in our opinion) good and sensible way described in this document.

Another challenge was the search support. This is a complex but crucial component of the repository and the decision to use the Lucene engine in RoOLO seems to be well suited for our purpose in SCY.

The first trials have been made using the RoOLO-FS as JPA-RoOLO was not finished at that time. We made good experiences with JPA-RoOLO on our test and development environments so that we will conduct the next trials based on that technology.

References

- LOM (2002) Draft *standard for learning object metadata*. New York: Institute of Electrical and Electronics Engineers. Retrieved on February 23, 2009, from http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf
- BATES89 Bates, Marcia J., (1989) The design of browsing and berrypicking techniques for the online search interface. Los Angeles: Graduate School of Library and Information Science, University of California at Los Angeles