Project no. 027391

Project acronym: LT4eL
Project title: Language Technology for eLearning

Instrument Specific Targeted Research Project
Thematic Priority Information Society Technology

D4.1c ILIAS LMS with integrated functionalities and documentation – second cycle

Due date of deliverable: 31-05-2008
Actual submission date: 08-07-2008

Start date of project: 1-12-2005
Duration: 30 Months

Organisation name of lead contractor for this deliverable: Swiss Federal Institute of Technology (ETHZ)

Revision [1]

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
</tr>
<tr>
<td>PP</td>
</tr>
<tr>
<td>RE</td>
</tr>
<tr>
<td>CO</td>
</tr>
</tbody>
</table>
LT4eL – Work Package 4
D4.1 Integration Report
Overview

LT4eL is a project that offers a set of language technology based services for e-Learning platforms. The acronym stands for Language Technology for e-Learning. Basic services are keyword generation, definition detection and enhanced semantic and multilingual search capabilities. The tools support a more effective metadata annotation of learning objects, authoring of glossaries and better search functionalities in learning management systems.

The LT4eL tools are implemented as web services that are deployed in a Java web server architecture and can be used via a SOAP interface by other applications. This makes the LT4eL tools independent from the technology that is used in the e-Learning platform, since SOAP libraries are available for all major programming languages like Java, C++, C#, PHP, Python or Perl.

This document is part one of the deliverable 4.1 as described in the technical annex of the project. It is targeted at developers of other learning managements systems and provides a step-by-step procedure that explains how to download, install, configure the LT4eL tools. It also documents the web services offered by the tools and explains how to integrate them into another LMS. To see how these functionalities can be used and affect a user interface within a learning management system, refer to the User Documentation document of the LT4eL project. For possible future improvements please refer to the Changes and Future Plans Annex provided with the 4.1 deliverables.

All source code of the project is available at https://sourceforge.net/projects/lt4el.

Change Log

Version 0.1

- Basic implementation of sendNewLO(), getStatus(), deleteLO(), findKeywordCandidates(), getDefinitionCandidates(), search() and getConceptNeighbourhood()

Version 0.2

- Changes in properties file
- Search returns concepts related to search results
- Revised search implementation
- Increased stability of tools (especially for concurrent requests)
- getDomain() and domain parameters abandoned
- sendNewLo() input parameters corrected
- removed restriction of deleteLO() (language model is updated, when deleteLO is called now)
- removed restriction of getDefinitionCandidates() (local grammars are included within the package now)
- removed restriction for search
- search result includes related concepts for each LO now
- sendApprovedDefinitions() abandoned
• For more information see project’s M24 Changes Report

Version 1.0

• Revised lt4el.props.template section (new, reduced number of settings).
• Added ant targets for building command line utility and stand-alone search GUI.
• Renamed sendNewLo webservice implementation to SendNewLoAnnotated. This service can be used to upload pre-annotated learning objects.
• Added new sendNewLo webservice implementation. This service is now used to send original raw learning objects that are not pre-annotated.
• getStatus webservice now returns additional information on processing status in a message string.
• search webservice has been completely revised. Now it allows a combination of the different search methods (semantic, keywords, fulltext) and conjunctive and disjunctive combination of the search terms. Each search result provides a score based on its relevance and a text snippet of search terms surrounding text.
• Added getRankedConceptsForDocs webservice.

Version 1.1

• Added LPC related documentation
# Table of Contents

OVERVIEW ................................................................................................................................. 2  
CHANGE LOG ............................................................................................................................... 2  
ARCHITECTURE ........................................................................................................................... 5  
INSTALLATION ........................................................................................................................... 5  
  PREREQUISITES ....................................................................................................................... 5  
  DOWNLOAD .............................................................................................................................. 6  
  CONFIGURATION ..................................................................................................................... 6  
  BUILDING AND DEPLOYING ...................................................................................................... 7  
LINGUISTIC PROCESSING CHAIN ............................................................................................... 8  
  OVERVIEW .............................................................................................................................. 8  
  CONFIGURATION ..................................................................................................................... 8  
WEB SERVICES ........................................................................................................................... 9  
  SENDNEWLO ........................................................................................................................... 9  
  SENDNEWLOANNOTATED ........................................................................................................ 10  
  GETSTATUS ............................................................................................................................ 10  
  DELETELO ................................................................................................................................ 11  
  FINDKEYWORDCANDIDATES ................................................................................................. 11  
  SENDAPPROVEDKEYWORDS .................................................................................................... 12  
  GETDEFINITIONCANDIDATES ................................................................................................. 12  
  SEARCH ................................................................................................................................... 12  
  GETCONCEPTNEIGHBOURHOOD .............................................................................................. 13  
  GETRANKEDCONCEPTSFORDOCS .......................................................................................... 14  
CLIENT EXAMPLE CODE ........................................................................................................... 15  
  PHP ......................................................................................................................................... 15  
  JAVA ...................................................................................................................................... 18  
ILIAS LMS PROTOTYPE IMPLEMENTATION ............................................................................. 21
Architecture

The below picture shows the major components of the integration setup. The language technology server on the left provides the keyword extractor, definitory context finder and ontology management system functionalities.

The tools themselves are developed using the Java programming language and are hosted on a Java web server. The functionalities can be accessed by a learning management system through the web service interface. The web service interface separates the technologies used on both sides.

Installation

This section explains which prerequisites are required to use the LT4eL tools and how to build and deploy them. Please note that these are requirements for running the LT4eL tools server. Your learning management system may run on any other technology.

Prerequisites

To use the LT4eL tools you need:

- Apache Tomcat 5.5 or later
- Apache Ant 1.6.x or later

Apache Tomcat is a web container developed at the Apache Software Foundation (ASF). Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. You can get Apache Tomcat and instructions about installation at http://tomcat.apache.org/.

Apache Ant is a Java-based build tool, like Make, but without Make’s wrinkles and with the full portability of pure Java code. The LT4eL tools use an Ant script to build and deploy the tools in the Apache Tomcat web container. You can get Apache Ant and instructions about
the installation at \url{http://ant.apache.org/}.

**Download**

You can find the latest release of the LT4eL tools at the SourceForge.net project page at \url{http://sourceforge.net/projects/lt4el/}. The package will be named something like lt4el.1.m30.beta.tar.gz. Please use the tar command to extract the package to a local folder.

```sh
> tar -xzvf lt4el.1.m30.beta.tar.gz
```

If you want to use the latest development snapshot of the tools, you can check them out from the subversion server on Sourceforge.net.

```sh
> svn co https://lt4el.svn.sourceforge.net/svnroot/lt4el/trunk/ lt4el
```

**Configuration**

Before building the tools please make sure, that the CATALINA_HOME environment variable is set to your Tomcat directory. For example if you are using the bash shell, you could add the following line to your ~/.bash_profile file.

```bash
export CATALINA_HOME=/<your_path>/apache-tomcat-5.5.20
```

To configure the LT4eL tools make a copy of the file lt4el.props.template int the LT4eL main directory and name this copy lt4el.props. This properties file contains a number of paths to directories that store various data of the LT4eL tool set.

```
# LT4eL properties file
#

# 1. Save a copy of this file as 'lt4el.props':
#   lt4el.props.template -> lt4el.props
# 2. Change the values according to your environment. Save the file.

# LO files directory, must include the necessary .dtd files
Docs=/path/to/lo/directory

# Internal DB index directory (subdirectories are created)
Index=/path/to/database/directory

# Ontology lexicons path
Lexicons=/path/to/directory/of/lexicons

# Ontology OWL file
Owl=/path/to/owl/file

# Linguistic processing chain config file path
LpcConfig=/path/to/lpc/config/file
```

You also need to configure the Linguistic processing chain in a similar fashion. In the lpc/
directory, rename lpc.xml.template to lpc.xml and adjust configuration according to your needs (mainly environment variables). Then run lpc/check-deps.sh script to see whether all required binaries and modules are correctly installed. For further details see Linguistic Processing Chain section.

**Building and Deploying**

After configuration you just need to call the ant command in the main directory to build and deploy the LT4eL tools. The default target of the ant script builds and deploys everything into two directories in the Tomcat webapps/ folder. The directory „lt4el“ contains all tools except the web services, whereas the web services are deployed into a directory called „lt4elservices“.

Beside the default target, the ant script offers a number of other targets for the different build and deployment steps. If you type „ant help“ you will get a list of all targets.

- **ant deploy.all**: Compiles and deploys everything into Tomcat webapp folder. (default target)
- **ant deploy.restart**: Same as deploy.all, tries to restart Tomcat (Unix only).
- **ant help**: This help message.
- **ant clean**: Cleans build and deploy directory.
- **ant compile.kwext**: Compiles keyword extractor component.
- **ant deploy.kwext**: Compiles and deploys keyword extractor component.
- **ant compile.defcon**: Compiles defcontext component.
- **ant deploy.defcon**: Compiles and deploys definition finder component.
- **ant compile.gui**: Compiles GUI component.
- **ant deploy.gui**: Compiles and deploys GUI component.
- **ant build.war**: build war-file for tomcat deployment
- **ant compile.webservices**: Compiles webservices component.
- **ant deploy.webservices**: Deploy webservices component.
- **ant lt4el-cmd.jar**: Creates command-line utility.
- **ant lt4el-searchgui.jar**: Creates stand-alone search GUI.
Linguistic Processing Chain

The Language Technology server (LTserver) needs to call several external tools for learning objects at various points of processing (conversion programs and scripts, tokenizers, taggers, tools for definition extraction, etc.). Thus it was necessary to establish a framework for external tools execution – Linguistic Processing Chain (LPC).

Overview

LPC is a general framework for executing different programs and scripts in a unified way. Every tool in the LPC is described by five parameters: input format, output format, allowed languages, cost and command to be executed. Input and output formats are defined using public or private MIME type (e.g. text/html or text/x-ana). Languages parameter may be used to specify languages accepted by the tool (for example tagger is usually language-specific) or asterisk (*) can be used to describe language-independent tools (e.g. conversion from text/html to text/plain format). By using cost parameter it is possible to express preference of one tool over another. The value is an arbitrary positive integer, it may represent any property of the tool – slowness, quality, memory requirements, etc. For example there may be generic tokenizer usable for all languages with high cost and English-specific one with lower cost, so that for processing English documents, better (lower-cost) tokenizer is used. Finally command parameter describes the script or program to be executed (and its arguments). The command must accept input format on the standard input and produce output format on its standard output.

We can describe the whole processing system as a general directed graph, where every vertex is assigned one file format and a set of edges represent various conversion tools. In fact we use a hypergraph: we parametrize every edge by two additional labels: the language and the cost (as described above), so that several edges (tools) can be used for conversion from one format into another.

When LTserver requires performing a conversion, a Dijkstra's algorithm for finding shortest path in the graph is called. Result of the algorithm is a list of conversion steps that must be followed to perform the conversion.

Configuration

For successful file processing, LPC must be configured properly during the LTserver installation. The main configuration file is specified in lt4el.props, default it lpc/lpc.xml. The structure of lpc.xml config file is quite simple, there are two main sections:

- environment variables definition, using <env></env> tags. This can be used for PATH, PERL5LIB, JAVA_HOME, and similar environment variables setup. `${basedir}` string will be substituted by a base dir of the LPC (by default lpc/).

- tool description, using <tool/> tag. Attributes used are:
  
  o src: source format, e.g. text/html
  o dst: destination format, e.g text/plain
  o lang: list of languages supported by given tool, or "*" for any language
  o cost: cost for this tool, this is used by Dijkstra's algorithm, enables introduction of preference of some tool over another one.
  o command: command to be executed. `${lang}` string will be substituted by
actual language used.

List of MIME types currently used in LPC:

- text/plain, text/html, application/pdf, application/msword: common document formats
- text/x-base: simplified version of XHTML, defined by LT4eLBase.dtd, starting point for language processing
- text/x-ana: basic XML format (so-called Ana format): segmented, tokenized, lemmatized/tagged text, used for further processing
- text/x-ana-base: intermediate format, the same as text/x-ana, but before lemmatization/tagging was performed
- text/x-ana-def: Ana format with marked definitions (output of lxtransduce tool)
- text/x-ana-ont: Ana format with marked concepts (semantic annotation)

All conversion scripts are executed with current working directory set to the base dir of the LPC (default is lpc/), tools expect input on the standard input and produce output on the standard output.

Many conversion scripts require installation of additional software (taggers for example), exact requirements are described in the INSTALL files in corresponding directories. Because proper setup of all tools is very error-prone and time-consuming task, you can use a script (lpc/check-deps.sh) which performs basic checking of availability of required tools and their dependencies.

Currently, LPC can be used (after installation of all dependencies) for processing text, HTML, PDF and DOC files in Czech, Dutch, English, Polish, Portuguese and Romanian. German is not possible as they use some shallow parsing, which is very slow. Maltese and Bulgarian have not been implemented. In addition, LPC is used for extracting definitions from linguistically annotated files (all languages) and for automatic semantic annotation.

For some languages (English and Romanian), several tasks are executed at once (such as tokenization+tagging+conversion into the Ana file format) using the ALPE processing tool.

**Web Services**

All functionalities of the LT4eL provide a web service interface. If you have configured the tools, you can use them with standard technologies like WSDL and SOAP through this interface. After successfully installing the tools you can access the web service description at:

http://<your_tomcat_url>/lt4elservice/services/Lt4elService?wsdl

The current specification lists eight implemented methods.

**sendNewLo**

Sending a new learning object to the language technology server. This is the first method that should be invoked right after a new learning object is created in or uploaded to a learning management system. This services passes the URL of a learning object in its original format (HTML, PDF, ...) to the server. To make this work, the linguistic processing chain for the given language and format must be configured and running.

**Input Parameters**

- loid (xsd:string): Learning Object ID. This ID is used to identify the learning object. It is assumed, that this ID is generated in the learning management system, when new
learning objects are created. This ID is used as an input/output parameter in most if the other functions.

- **language (xsd:string)**: The (main) language of the learning object. The language must be represented by a two-letter code as defined in ISO 639-1. See [http://www.oasis-open.org/cover/iso639a.html](http://www.oasis-open.org/cover/iso639a.html) for details.
- **url (xsd:string)**: URL of the learning object file in its original format.

**sendNewLoAnnotated**

Sending a pre-annotated learning object to the language technology server. This service does basically the same as sendNewLo. The difference is that instead of a raw learning object, file paths to local pre-annotated versions of the learning object are passed as parameters. The files must respect the LT4eLAna DTD. The service should be used, when a language processing chain is not available for a given language.

**Input Parameters**

- **loid (xsd:string)**: Learning Object ID. This ID is used to identify the learning object. It is assumed, that this ID is generated in the learning management system, when new learning objects are created. This ID is used as an input/output parameter in most if the other functions.
- **language (xsd:string)**: The (main) language of the learning object. The language must be represented by a two-letter code as defined in ISO 639-1. See [http://www.oasis-open.org/cover/iso639a.html](http://www.oasis-open.org/cover/iso639a.html) for details.
- **filename (xsd:string)**: Local path to the pre-annotated learning object file (LT4eLAna DTD).
- **attach (xsd:boolean)**: not used.
- **filename2 (xsd:string)**: Local path to ontological annotated file. If no path is given, semantic search will not be available for this learning object.

**Output Parameters**

- **accepted (xsd:boolean)**: True, if the language technology server successfully received the learning object, false otherwise.

**getStatus**

Get processing status of a learning object that has been sent to the language server. This function can be used after *sendNewLO* or *sendNewLOAnnotated* has been invoked for a learning object. Since the processing and conversion of a new learning object may take several minutes, this function tells the learning management system the status of the processing. It can be used to be displayed for the user and to deactivate certain functions, that cannot be used until the processing status is *finished*.

**Input Parameters**

- **loid (xsd:string)**: Learning Object ID. (see *sendNewLo*).

**Output Parameters**

- **status (DocumentStatus)**: Two parameters are returned. Status contains the current processing status of the LO as string with the following possible values:
  
  **UNKNOWN**
The second parameter StatusStr contains a longer status message, containing additional information e.g. about a processing failure.

Types
- DocumentStatus: sequence of
  - Status (xsd:string)
  - StatusStr (xsd:string)

**deleteLO**
Delete a learning objects representation on the language technology server. This function should be called when a learning object is deleted in the learning management system. After successfully invoking deleteLO, subsequent calls to getStatus will return UNKNOWN again.

**Input Parameters**
- loid (xsd:string): Learning Object ID. (see sendNewLo).

**Output Parameters**
- success (xsd:boolean): True, if the language technology server successfully removed the the learning object, false otherwise.

**findKeywordCandidates**
Find candidate terms for keyword annotation of a learning object. This method should be used by a learning management system, when a learning object is annotated with keywords. A lot of learning management systems come with support for LOM or Dublin Core meta data. Both of these standards allow the annotation with keywords. However, simple tagging systems work the same way and could use this function to propose keywords to an annotator.

**Current Restrictions**
- In general this function works better, when the internal language model gets larger. This means if only a small number of learning objects have been sent to the language technology server by using the function sendNewLo, the quality of the results is suboptimal. Good quality can be expected after 30-50 mid-size learning objects have been sent to the language technology server.

**Input Parameters**
- loid (xsd:string): Learning Object ID. (see sendNewLo).
- maxnum (xsd:int): Maximum number of keywords that should be returned by the function.
  - tfidf: TF-IDF
  - ridf: R-IDF
  - adridf: ADR-IDF (currently best performing)

**Output Parameters**
Types
- ArrayOfString
  - minOccurs 0
  - maxOccurs unbounded
  - type xsd:string

sendApprovedKeywords
Send all keywords related to a learning object, that have been approved by a human annotator back to the learning technology server. The language technology server could later use this information, e.g. during the search.

Input Parameters
- loid (xsd:string): Learning Object ID
- keywords (tns:ArrayOfString): Keywords approved by an author.

Output Parameters
- success (xsd:boolean): Success true/false

getDefinitionCandidates
Get a set of terms and candidate definitions for a learning object. This method can be used by learning managements systems to support semi-automatic generation of glossaries with terms and definitions found in a learning object. The returned value context includes the surrounding text of the definition (usually around three sentences).

Input Parameters
- loid (xsd:string): Learning Object ID. (see sendNewLo).

Output Parameters
- definitions (ArrayOfDefinition): Array of terms and related defining texts.

Types
- ArrayOfDefinition
  - minOccurs 0
  - maxOccurs unbounded
  - type Definition
- Definition
  - type sequence of
    - context (xsd:string)
    - definedTerm (xsd:string)
    - definingText (xsd:string)

search
Search for learning objects. This function supports extended search capabilities based on fulltext search, keyword based search and semantic search. Semantic search supports multilingual retrieval of learning objects by using lexicons and an ontology.
**Input Parameters**

- `searchTerms (tns:ArrayOfString)`: The search terms
- `semantic (xsd:boolean)`: Semantic search.
- `keywords (xsd:boolean)`: Keyword search.
- `fulltext (xsd:boolean)`: Fulltext search.
- `conjunctive (xs:boolean)`: Conjuctive combination of search terms (otherwise disjunctive)
- `searchLangs (tns:ArrayOfStrings)`: Search term languages
- `retrievalLangs (tns:ArrayOfStrings)`: Languages of target learning objects
- `method (xsd:string)`: Search Method ("SEMANTIC", "KEYWORD", "FULLTEXT")
- `searchConcepts (tns:ArrayOfString)`: Concepts, if learning objects related to concepts are searched
- `systemLang (xsd:string)`: System Language
- `maxSnippets (xsd:int)`: Maximum number of search context snippets that should be returned. Use lower values to increase performance.

**Output Parameters**

- `searchresult (WSSearchResult)`: Two arrays. The first one includes a result list with all found LOs, including information on score, text context, related concepts. The second array holds a list of ontology concepts that are related to the search terms.

**Types**

- `WSSearchResult`
  - `resultList` `ArrayOfSearchResult`
  - `termConcepts` `ArrayOfString`

- `Search Result`
  - `docid` `xsd:string` (Learning Object ID)
  - `fulltext` `xsd:boolean` (Found by fulltext search)
  - `semantic` `xsd:boolean` (Found by semantic search)
  - `keyword` `xsd:boolean` (Found by keyword search)
  - `matchingConcepts` `ArrayOfString` (Concepts related to search terms and LO)
  - `rankedConcepts` `ArrayOfString` (Concepts annotated to the LO)
  - `score` `xsd:double` (Relevance Score)
  - `snippet` `xsd:string` (Contextual text snippet)

**getConceptNeighbourhood**

Get relations and related concepts of a ontology concept. This function can be used to support browsing through the ontology in the learning management system’s interface.

**Input Parameters**

- `concepts (tns:ArrayOfString)`: Concepts
- `languages (tns:ArrayOfString)`: Languages (the entries in the lexicons for these
languages and the concepts included in the fragments will be returned)

**Output Parameters**

- `fragments (tns:ArrayOfString)`: Ontology fragments for concepts.

**getRankedConceptsForDocs**

This service returns all concepts that are related to one (or multiple) learning object(s). The concepts are ranked according to their number of occurrences in the learning module.

**Input Parameters**

- `loids (tns:ArrayOfString)`: Learning Object IDs.

**Output Parameters**

- `concepts (tns:ArrayOfConceptItem)`: List of ranked concepts.

**Types**

- `ConceptItem`:
  - `concept` (ArrayOfSearchResult)
  - `docId` (xsd:string)
Client Example Code

SOAP libraries are available for all major programming languages. In this section we want to provide some example implementations that show how to use the LT4eL web services. If you write a similar code in a programming language that is not listed here, please do not hesitate to send us your code. We would like to extend this chapter with additional examples.

To write an adapter to the LT4eL webservices is the major step to use the functionalities within your application. You can then call the adapter’s functions to extend your application with the functionalities of the LT4eL tools.

PHP

For PHP a popular SOAP library is nuSOAP (see http://sourceforge.net/projects/nusoap/). The following example code uses this library. It implements a wrapper class for all functions of the language technology server.

```php
<?php

// LT4eL Example Class

// You must install nuSoap to us this example.
include_once ("./nusoap.php");

/**
 * LT Server Adapter
 */
class LTServerAdapter
{
    /**
     * Constructor
     */
    function __construct()
    {
        // replace the following URL with your LT server address
        $this->setServer(
            "http://localhost:8080/lt4elservice/services/Lt4elService";

        // initialize soap client
        $this->client = new soap_client($this->getServer());
        $this->client->soap_defencoding = 'UTF-8';
        $this->client->decode_utf8 = false;
        $this->client->response_timeout = 240;
        if ($err = $this->client->getError())
        {
            die('SOAP Client constructor error: ' . $err);
        }
    }

    /**
     * Set Server.
     *
     * @param string $a_server Server
     */
    function setServer($a_server)
    {
        $this->server = $a_server;
    }

    /**
     * Get Server.
     * @return string Server
     */
    function getServer()
    {
        return $this->server;
    }

    /**
     * Send new learning object
     *
     * @param string $a_lo_id learning object id
     */
```
function sendNewLO($a_lo_id, $a_language, $a_url)
{
    // sendNewLO call
    $accepted = $this->client->call('sendNewLO',
        array('loid' => (string) $a_lo_id,
              'language' => $a_language,
              'url' => $a_url));
    
    return $accepted;
}

function sendNewLOAnnotated($a_lo_id, $a_language, $a_filename, $a_attach,
                             $a_ont_filename)
{
    // sendNewLO call
    $accepted = $this->client->call('sendNewLO',
        array('loid' => (string) $a_lo_id,
              'language' => $a_language,
              'filename' => $a_filename,
              'attach' => $a_attach,
              'filename2' => $a_ont_filename));
    
    return $accepted;
}

function getStatus($a_lo_id)
{
    // getStatus call
    $ret = $this->client->call('getStatus',
        array('loid' => (string) $a_lo_id));
    
    return $ret;
}

function deleteLO($a_lo_id)
{
    // deleteLO call
    $success = $this->client->call('deleteLO',
        array('loid' => (string) $a_lo_id));
    
    return $success;
}

function findKeywordCandidates($a_lo_id, $a_number, $a_method = "adridf")
{
    // findKeywordCandidates call
$keywords = $this->client->call('findKeywordCandidates',
    array('loid' => (string) $a_lo_id,
          'maxnum' => (int) $a_number,
          'method' => $a_method));

return $keywords;
}

/**
* Send approved keywords
*
* @param string  $a_lo_id learning object id
* @param string  $a_keywords approved keywords
* @return boolean     success
*/
function sendApprovedKeywords($a_lo_id, $a_keywords)
{
    if (!is_array($a_keywords))
    {
        $a_keywords = array($a_keywords);
    }

    // sendApprovedKeywords call
    $success = $this->client->call('sendApprovedKeywords',
        array('loid' => (string) $a_lo_id,
              'keywords' => $a_keywords));

    return $success;
}

/**
* Get Definition Candidates
*
* @param string  $a_lo_id learning object id
* @return array definition candidates array of array ("definedTerm" =>..., 
*      "definingText" => ..., "context" => ...)
*/
function getDefinitionCandidates($a_lo_id)
{
    // findKeywordsCandidates call
    $definitions = $this->client->call('getDefinitionCandidates',
        array('loid' => (string) $a_lo_id));

    return $ret;
}

/**
* Get concepts and relations of a concept
*
* @param string  $a_concept  concept (in english)
* @param string  $a_preferred_lang preferred languages
* @return string   ontology fragments (for XML see WP3)
*/
function getConceptNeighbourhood($a_concepts, $a_preferred_langs = "en")
{
    if (!is_array($a_concepts))
    {
        $a_concepts = array($a_concepts);
    }
    if (!is_array($a_preferred_langs))
    {
        $a_preferred_langs = array($a_preferred_langs);
    }

    $frags = $this->client->call('getConceptNeighbourhood',
        array('concept' => $a_concepts,
              'preferred_languages' => $a_preferred_langs));

    return $frags;
}

/**
* Get concepts annotated to a list of learning objects
*
* @param array  $a_lo_ids array of learning object ids
* @return array    array of concepts per learning object
*/
function getRankedConceptsForDocs($a_lo_ids)
{
    // getRankedConceptsForDocs call
    $concepts = $this->client->call('getRankedConceptsForDocs',
        array('loids' => $a_lo_ids));
}
return $concepts;

/**
 * Search for learning objects
 *
 * @param string  $a_search_terms  array of search string
 * @param array  $a_languages  search terms languages (only needed
 *        for semantic)
 * @param boolean  $a_semantic  use semantic search
 * @param boolean  $a_keyword  use keyword search
 * @param boolean  $a_fulltext  use fulltext search
 * @param string  $a_system_language  system language used by the user
 * @param array  $a_retrieval_languages  retrieval languages
 * @param array  $a_search_concepts  look for LOs that are related to
 *        concepts
 * @param boolean $a_conjunctive  conjunctive combination of search terms
 * (otherwise disjunctive)
 * @param int  $a_max_snippets  maximum number of snippets
 *
 * @return array  $result["resultList"] contains found LOs as array:
 *      "docid" => LO ID
 *      "matchingConcepts" => concepts found in LO
 *      "rankedConcepts" => concepts of LO
 *      "score" => relevance score
 *      "snippet" => context text snippet
 *      "fulltext" => found by fulltext search
 *      "semantic" => found by semantic search
 *      "keyword" => found by keyword search
 *
 * $result["term Concepts"] contains found concepts
 */
function search($a_search_terms, $a_languages,
                $a_semantic, $a_keyword, $a_fulltext, $a_system_language,
                $a_retrieval_languages, $a_search_concepts = "",
                $a_conjunctive = false, $a_max_snippets = 100)
{
    if ($a_search_concepts == "")
    {
        $a_search_concepts = array();
    }

    $result = $this->client->call('search',
                                array(
                                    "semantic" => (boolean) $a_semantic,
                                    "keywords" => (boolean) $a_keyword,
                                    "fulltext" => (boolean) $a_fulltext,
                                    "searchTerms" => $a_search_terms,
                                    "conjunctive" => $a_conjunctive,
                                    "retrievalLangs" => $a_retrieval_languages,
                                    "searchConcepts" => $a_search_concepts,
                                    "systemLang" => $a_system_language,
                                    "maxSnippets" => $a_max_snippets
                                )
                                );

    return $result;
}

Java

For Java a popular SOAP library is Axis (see http://ws.apache.org/axis/). This example uses
Axis Version 1.4. The first thing you have to do is to generate the Mapping-Classes,
WebService-Stubs and the Definition-Classes. To do this download the axis-distribution,
extract it, change to the axis-directory/lib and then run the following command:

java -cp commons-logging-1.0.4.jar:axis.jar:commons-discovery-
0.2.jar:saaj.jar:jaxrpc.jar:log4j-1.2.8.jar:wsdl4j-1.5.1.jar
org.apache.axis.wsdl.WSDL2Java

http://localhost:8080//lt4elservice/services/Lt4elService?wsdl

The classes are generated in the eu-subfolder. The following example code uses axis and the
autogenerated classes. It implements a wrapper class for all functions of the language
technology server.

package eu.lt4el.webservices;
import java.rmi.RemoteException;
import org.apache.axis.AxisFault;
import eu.lt4el.defcontext.Definition;
import eu.lt4el.webservices.service.Lt4ElServiceHttpBindingStub;
import eu.lt4el.webservices.service.Lt4ElServiceLocator;
import eu.lt4el.webservices.service.Lt4ElServicePortType;

public class LTServerAdapter {
    private Lt4ElServiceLocator serviceLocator;
    private Lt4ElServicePortType serviceBinding;

    public LTServerAdapter(String address) throws AxisFault {
        serviceLocator = new Lt4ElServiceLocator();
        serviceLocator.setLt4elServiceHttpPortEndpointAddress(address);
        serviceBinding = new Lt4ElServiceHttpBindingStub(serviceLocator);
    }

    public boolean deleteLO(String loid) throws Exception {
        try {
            return serviceBinding.deleteLO(loid);
        } catch (RemoteException e) {
            throw new Exception("Can't delete LO. with id = " + loid, e);
        }
    }

    public String[] findKeywordCandidates(String loid, int maxnum, String method) throws Exception {
        try {
            return serviceBinding.findKeywordCandidates(loid, maxnum, method);
        } catch (RemoteException e) {
            throw new Exception("Error! Write your message here!", e);
        }
    }

    public String[] getConceptNeighbourhood(String[] concepts, String[] languages) throws Exception {
        try {
            return serviceBinding.getConceptNeighbourhood(concepts, languages);
        } catch (RemoteException e) {
            throw new Exception("Error! Write your message here!", e);
        }
    }

    public ClientImpl() throws WebserviceException {
        serviceLocator = new Lt4ElServiceLocator();
        try {
            serviceBinding = new Lt4ElServiceHttpBindingStub(serviceLocator);
        } catch (AxisFault e) {
            throw new Exception("Can't instaniate ClientImpl! ", e);
        }
    }

    public ClientImpl(String loid) throws WebserviceException {
        serviceLocator = new Lt4ElServiceLocator();
        try {
            serviceBinding = new Lt4ElServiceHttpBindingStub(serviceLocator);
        } catch (AxisFault e) {
            throw new Exception("Can't instaniate ClientImpl! ", e);
        }
    }

    /**
     * Constructor
     */
    public LTServerAdapter() throws AxisFault {
        // Constructor code
    }

    /** Delete learning object */
    * @param string loid learning object id
    * @return boolean success
    */
    public boolean deleteLO(String loid) throws Exception {
        try {
            return serviceBinding.deleteLO(loid);
        } catch (RemoteException e) {
            throw new Exception("Can't delete LO. with id = " + loid, e);
        }
    }

    /** Find Keyword Candidates */
    * @param string loid learning object id
    * @param int maxnum number of keywords
    * @param string method method
    * @return string keyword candidates
    */
    public String[] findKeywordCandidates(String loid, int maxnum, String method) throws Exception {
        try {
            return serviceBinding.findKeywordCandidates(loid, maxnum, method);
        } catch (RemoteException e) {
            throw new Exception("Error! Write your message here!", e);
        }
    }

    /** Get concepts and relations of a concept */
    * @param string concepts concept (in english)
    * @param string languages preferred languages
    * @return string ontology fragments (for XML see WP3)
    */
    public String[] getConceptNeighbourhood(String[] concepts, String[] languages) throws Exception {
        try {
            return serviceBinding.getConceptNeighbourhood(concepts, languages);
        } catch (RemoteException e) {
            throw new Exception("Error! Write your message here!", e);
        }
    }

    /** Get Definition Candidates */
    * @param string loid learning object id
    *
public Definition[] getDefinitionCandidates(String loid) throws Exception {
    try {
        return serviceBinding.getDefinitionCandidates(loid);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

public DocumentStatus getStatus(String loid) throws Exception {
    try {
        return serviceBinding.getStatus(loid);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

public WSSearchResult search(boolean semantic, boolean keywords, boolean fulltext,
                             String[] searchTerms, boolean conjunctive, String[] searchLangs,
                             String[] retrievalLangs, String method,
                             String[] searchConcepts, String systemLang, int maxSnippets)
        throws Exception {
    try {
        return serviceBinding.search(semantic, keywords, fulltext, searchTerms,
                                   conjunctive, searchLangs, retrievalLangs, method, searchConcepts,
                                   systemLang, maxSnippets);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

public boolean sendApprovedKeywords(String loid, String[] keywords) throws Exception {
    try {
        return serviceBinding.sendApprovedKeywords(loid, keywords);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

public boolean sendNewLO(String loid, String language, String url) throws Exception {
    try {
        return serviceBinding.sendNewLO(loid, language, url);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}
try {
    return serviceBinding.sendNewLO(loid, language, url);
} catch (RemoteException e) {
    throw new Exception("Error! Write your message here!", e);
}

/**
 * Send new learning object (pre-annotated version)
 *
 * @param string loid  learning object id
 * @param string language language
 * @param string filename  local filename for LT4eLAna annotated file
 * @param string filename2  ontology annotated file
 * @param boolean attach  file content (not used yet)
 *
 * @return boolean   accepted
 */
public boolean sendNewLOAnnotated(String loid, String language, String filename, boolean attach, String filename2) throws Exception {
    try {
        return serviceBinding.sendNewLOAnnotated(loid, language, filename, attach, filename2);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

/**
 * Get ranked concepts for learning objects
 *
 * @param array  loid array of learning object ids
 *
 * @return array  ranked concepts
 */
public ConceptItem[] getRankedConceptsForDocs(String[] loid,) throws Exception {
    try {
        return serviceBinding.getRankedConceptsForDocs(loid);
    } catch (RemoteException e) {
        throw new Exception("Error! Write your message here!", e);
    }
}

**ILIAS LMS Prototype Implementation**

The LT4EL tools have been integrated into the ILIAS open source learning management system. You can download the prototype implementation of ILIAS at [http://sourceforge.net/projects/lt4el/](http://sourceforge.net/projects/lt4el/). The package will be named something like ilias-lt4el.m30.tar.gz. Please use the tar command to extract the package to a folder within your webspace.

```
> tar -xzvf ilias-lt4el.m30.tar.gz
```

The ILIAS project provides installation instructions for different kinds of operating systems at [http://www.ilias.de/docu/](http://www.ilias.de/docu/). You can use one of these or the instructions in the docs/INSTALL file for installing the prototype as well.

After the usual ILIAS setup, login into ILIAS and open the ILIAS administration at [http://<your.domain>/<your_path>/ilias.php?baseClass=ilAdministrationGUI](http://<your.domain>/<your_path>/ilias.php?baseClass=ilAdministrationGUI).

Enter the URL of the LT SOAP server and the paths of the lexicons and the ontology in the administration panel. Select „Use Processing Chain“ for all languages that provide a configured linguistic processing chain. If you want to use the tools for other languages, you need to put pre-annotated files into the directory specified under „LO Source data directory“. After entering all values, press the „Save“ button at the bottom of the administration screen. After that click on "Create additional fields and tables“ and "Load Ontology and Lexicons into DB“. Now all functionalities described in the „LT4EL WP4 D4.1 User Documentation“ document should be available.
Overview

LT4eL is a project that offers a set of language technology based services for e-Learning platforms. The acronym stands for Language Technology for e-Learning. Basic services are keyword generation, definition detection and enhanced semantic and multilingual search capabilities. The tools support a more effective metadata annotation of learning objects, authoring of glossaries and better search functionalities in learning management systems.

This document is part two of the deliverable 4.1 as described in the technical annex of the project. This document is an extension to the documentation of the ILIAS learning management system (Version 3.7.6). It addresses all functionalities that have been changed, extended or added due to the developments of the LT4eL project:

- Glossary Generation: Semi-automatic extraction of glossary terms and definitions from existing learning objects to facilitate the generation of glossaries.
- Extended Search Engine: The search engine makes use of additional ontology-based and metadata-based (keywords) information. Users can browse the ontology and perform subsequent searches to find learning objects related to concepts in the ontology.
- Personal Desktop: Users can create references to learning objects on their personal desktop on the basis of search results. They can rearrange learning objects according to their individual needs.

All source code of the project is available at https://sourceforge.net/projects/lt4el.
Change Log

Version 0.1

- Basic implementation LT4eL functionalities.

Version 0.2

- Show 10 instead of 5 keywords per block. Show max. 50 in total.
- Added context of definitions in glossary generation screen.
- „Generate Glossary“ button renamed to „Finish and Save Glossary“.
- Glossaries are directly attached to learning object. They are listed together with the learning objects within the repository.
- Attached glossaries can be viewed and deleted from the info screen of the parent learning object.
- Moved chapter on learning object deletion before chapter on glossary generation.
- Search terms languages are selective immediately on the search screen.
- Topics related to learning objects are listed within semantic search results.
- Added lexicon definitions to ontology browser.
- Added learning path reordering on personal desktop section.

Version 1.0

- Revised chapter on search completely (new features: relevance, text snippets, combination of search methods, conjunctive/disjunctive search).
- Revised chapter on ontology browsing (complete new screens, separated from search).
- Revised chapter of personal desktop (hierarchical organisation of the resources, arrows instead of numbers for reordering).
Table of Contents

OVERVIEW .......................................................................................................................... 2
CHANGE LOG .......................................................................................................................... 3
TABLE OF CONTENTS.......................................................................................................... 4
USING LT4EL FUNCTIONALITIES IN ILIAS ........................................................................... 5
  ADDING A NEW LEARNING OBJECT .................................................................................. 5
  DELETING A LEARNING OBJECT ..................................................................................... 6
  GENERATE KEYWORDS FOR A LEARNING OBJECT ......................................................... 7
  GENERATE A GLOSSARY ................................................................................................... 8
  SEARCH ............................................................................................................................... 10
  ONTOLOGY BROWSER ..................................................................................................... 11
  MANAGING LEARNING PATH ON PERSONAL DESKTOP ................................................. 12
Using LT4eL Functionalities in ILIAS

The changes done in ILIAS within the LT4eL project are related to the handling of learning objects, their metadata and their retrieval. Other major functionalities of ILIAS like course management, communication tools, test and assessment are not affected.

Adding a New Learning Object

The creation process of learning objects has been adopted to enable the use of a repository of pre-annotated learning objects (linguistically and ontologically). For some languages an automatic annotation process (LPC) has been implemented. If this process is not configured, ILIAS will present a list of pre-annotated learning objects of the repository that is connected with the system.

After logging into ILIAS click "Repository" in the main menu. The repository contains all learning material and tools organised in categories. If you start with an empty installation, please add at least one category to the repository first. To enter a category, simply click on the name of the category. To add a learning object into the current category, select "Learning Module" in the selection list at the top right, then click on "Add".

A dialog opens, asking for the language and the learning object. To list learning objects of another language, select the language and click "Change Language". After that select the Learning Object you want to add and click "Add". The processing of the learning object may take several seconds (up to a minute, if multiple users work on the system).
After this step, the "Information Screen" of the learning object opens. It displays the processing status of the LT4eL server, which should be "FINISHED" if everything has worked correctly. When you create a learning object, you become the owner of the object. The owner has all permissions (view, edit, delete) related to the learning object. You can click "View" and the original learning object will be shown.

Deleting a Learning Object

To delete a learning object, navigate to the corresponding repository category first (e.g. "English Learning Objects"). Now click on "Switch Administration Commands On" at the bottom of the screen. This will add some additional functionalities for users with administrative rights. Click on the "Delete" button of the learning object that should be deleted.
A confirmation screen is displayed. Click on "Confirm" and the learning object will be deleted finally. Please note, that "Delete" buttons will not appear for learning objects, if you do not have the necessary permission.

**Generate Keywords for a Learning Object**

After adding a learning object, keywords can be entered in the "Metadata" tab. To open it, simply click on "Metadata". The screen offers a form to enter title, language, description, keywords, authors, copyright and typical learning time information. It also offers a button called "Generate Keywords".
Clicking on "Generate Keywords" lists the first ten keywords delivered by the keyword extractor. The link "Show More" will display the next ten keywords. The link "Show Less" will hide the last added keywords from the list. To select keywords, simply check the corresponding checkboxes. You can add additional keywords in the text area next to the list. The list will show a maximum of 50 keywords.

If you are satisfied with your selected and entered keywords, click "Save" at the bottom of the form. All selected keywords will be put into the text area as a comma separated list. Switch back to the "Information Screen" by clicking "Info". The selected keywords will be displayed on the information screen, too. This screen and the view button is accessible by students.

**Generate a Glossary**

The LT4eL tools support the semi-automatic generation of glossaries. After adding a learning object, a glossary related to the learning object can be generated in the "Properties" tab of the learning object. To open it, simply click on "Properties".
You will find a button called "Generate Glossary". After clicking on this button ILIAS will list candidates for terms and their definition. You can choose which terms/definitions should be included in the glossary by checking the corresponding "Include in Glossary" checkbox. You may also make changes to the text. If you want to add new term/definition pairs, click on "Add Term/Definition" at the bottom of the page.

To generate the glossary, click the "Finish and Save Glossary" button at the bottom of the page. You will return to the initial "Properties" screen where two new buttons will appear, one for viewing the glossary another for deleting it.

Within the repository the glossary will be listed beneath the parent learning object. All glossaries are additionally listed at the bottom of each repository category.
Search

The search function of ILIAS has been extended to enable ontology based learning object retrieval. The extended ILIAS search also offers a keyword based search that uses the keywords as stored by the content provider during the learning object creation and metadata annotation process. Additionally a search for definitions within the generated glossaries and a standard fulltext search using a standard implementation of the Apache Lucene search engine are available.

To search for learning objects and definitions click on "Search" in the main menu. This will open the following form.

After entering the search term, you can select the languages of the learning objects you are interested in under „Retrieval Languages“. If you use semantic search, you also should ensure, that the list of „Search Terms Language(s)“ reflect the languages you used in the search terms input field.

The „Or“/“And“ option determines whether the search results should contain all of the search terms („And“) or just any of the search terms („Or“). After entering all necessary values, hit the „Search“ button. ILIAS will display the search result as a list of learning objects. Learning
objects can be put on the personal desktop of ILIAS for further management of your personal learning path by clicking on „Add to desktop“.

All search result items are ordered by relevance and show a small context textual snippet of the learning object, where the search terms occur. If semantic search is used, the search results will also include matching topics, which are concepts of the ontology. By clicking on a concept name, the topics browser opens and you may invoke subsequent searches for learning objects related to similar topics.

**Ontology Browser**

The ontology browser is opened, either by clicking on a topic listed with a search result or by clicking on „Topics“ in the main menu of ILIAS. Concepts of the ontology are called topics in the ILIAS user interface. The ontology browser features two main screens. The first one lists all topics in a hierarchical tree, as defined by the ontology. The second one lists all topics in an alphabetical order.

Both screens offer the possibility to search for learning objects, that are related to selected topics of the ontology. To do this, select the checkboxes of the corresponding topics and click on „Search“. Conjunctive or disjunctive combination of the topics can be determined by the „Or“/„And“ option.
The hierarchical view offers a search for topics at the top right of the screen. Entering search terms and clicking on „Search Topic“ will unfold and highlight all related topics in the hierarchy.

Both screens offer a selection of languages. Selecting additional languages will show the translations of the topics and search for learning objects in the respective languages, when the search is performed. Instead of a topic search the alphabetic view shows all characters of the specific alphabet. Clicking on a character will show all topics that start with the selected character.

**Managing Learning Path on Personal Desktop**

Within the repository and on search results screens you can put single listed learning objects on your personal desktop. This function adds a reference to the learning object on your
personal desktop and makes it easy to quickly access learning objects you are interested in next time without searching or browsing the repository again.

You can switch to your personal desktop anytime by clicking on „Personal Desktop“ in the main menu. You will get a list of all the learning objects, that you have selected before. A button on the top right will allow you to switch detailed for every learning object on or off. In the detailed view all annotated keywords and related topics are listed. A click on a topic will open the topic browser screen.

If you want to reconfigure the list, you can click on the „Reorder“ button in the upper right corner of the list. This will show additional functionalities that allow you to categorize the learning objects in sections and subsections and to reorder their sequence in each section.
To add a new section enter its name in the bottom left input field and click on „Add Section“. Afterwards all learning objects and existing sections will display a selection list on the right that allow you to put learning objects and sections into other sections. Simply select the desired section in the list and click on „Save Titles/Assignment“. This will also save modified section names in the corresponding input fields. To change the order of elements within one section, simply click on the up and down arrow icons. If you are satisfied with your new organization, click on „Finish“ to return to the standard desktop view.
LT4eL – Work Package 4
D4.1 Annex. M30 Changes
Change Log

Version 0.1
- Changes done due to testing and validation since initial integration up to M24.

Version 1.0
- Changes done due to M24 validation.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE LOG</td>
<td>2</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>3</td>
</tr>
<tr>
<td>OVERVIEW</td>
<td>4</td>
</tr>
<tr>
<td>MAJOR CHANGES M24 TO M30</td>
<td>4</td>
</tr>
</tbody>
</table>
Overview

This document gives an overview about the changes up to M30 that have been made to the integrated LT4eL tools within the ILIAS prototype due to M24 validation. The project has continued the use of a bug tracker at the sourceforge.net’s project page. The tracker includes detailed information on some issues including detailed description and correspondence between reporters and developers. The tracker is openly accessible at: https://sourceforge.net/tracker/?atid=881658&group_id=177544&func=browse

Major Changes M24 to M30

The work being done in this last period has been based on the outcome of M24 validation activities. A plan has been already outlined in the approved „D4.1 Annex. M24 Changes and Future Plans“ document provided with the D4.1 M24 deliverables.

The latest version of the keyword extractor and the glossary finder have been integrated into ILIAS. The web services of these functionalities have been proved to be already stable, so that this integration did not cost much time.

Major work has been done on the integration of the search and ontology browsing capabilities. The search results are now ordered by relevance, a relevance value from 0% to 100% is displayed to the user. In the old version users complained about the fact that it was unclear how search results are ordered. Text snippets now show parts of the learning objects where the search terms or related concepts occur.

Users can now combine the different search modes: semantic search, fulltext search, keyword search and definition search. The results of the different search types will be merged in the user interface.

An option for conjunctive or disjunctive combination of the search terms has been introduced, which was requested by more experienced users.

Ontology browsing has been moved to a separate view, since it confused some users, that did not expect it on a search results screen. Each result item lists now matching topics, which are related to the search terms. Clicking the learning objects opens the ontology view.

The ontology is now accessible from the search results or the main menu in ILIAS. The whole ontology is displayed in a hierarchical view, based on is-a-relations. Additional relations are listed as hyperlinks with each concept. Users complained about the fact, that the old version showed too small fragments of the ontology and that navigation into upper ontology concepts was not possible. Now they can unfold each part of the ontology, which gives them an overview of all parts. They can select multiple concepts and search for learning objects that are related to these concepts. An additional view lists all concepts in an alphabetical order. Like in the hierarchical view, users can select multiple concepts and search for related learning objects. Both screens allow to select multiple languages of interest. Translations of the concepts in the corresponding languages are displayed and search request look for learning objects in all selected languages.

The personal desktop has been revised to allow users to organize selected learning objects in sections and subsections. This should give them the opportunity to structure them in a hierarchical way like in the ontology.