



PROLEARN

European Commission Sixth Framework Project (IST-507310)

Deliverable	D4.6 PLQL - The ProLearn Query Language
--------------------	--

Editor *A. Campi, S. Ceri, E. Duval, S. Guinea, D. Massart, S. Ternier*

Work Package *WP 4*

Status *Draft*

Date *27/04/2007*

The PROLEARN Consortium

1. Universität Hannover, Learning Lab Lower Saxony (L3S), Germany
2. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Germany
3. Open University (OU), UK
4. Katholieke Universiteit Leuven (K.U.Leuven) / ARIADNE Foundation, Belgium
5. Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (FHG), Germany
6. Wirtschaftsuniversität Wien (WUW), Austria
7. Universität für Bodenkultur, Zentrum für Soziale Innovation (CSI), Austria
8. École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
9. Eigenössische Technische Hochschule Zürich (ETHZ), Switzerland
10. Politecnico di Milano (POLIMI), Italy
11. Jožef Stefan Institute (JSI), Slovenia
12. Universidad Politécnica de Madrid (UPM), Spain
13. Kungl. Tekniska Högskolan (KTH), Sweden
14. National Centre for Scientific Research "Demokritos" (NCSR), Greece
15. Institut National des Télécommunications (INT), France
16. Hautes Etudes Commerciales (HEC), France
17. Technische Universiteit Eindhoven (TU/e), Netherlands
18. Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Germany
19. Helsinki University of Technology (HUT), Finland
20. imc information multimedia communication AG (IMC), Germany
21. Open Universiteit Nederland (OU NL), Netherlands
22. University of Warwick

Document Control

Title: The ProLearn Query Language
Author/Editor: A. Campi, S. Ceri, E. Duval, S. Guinea, D. Massart, S. Ternier
E-mail: Campi/Ceri@elet.polimi.it

Amendment History

Version	Date	Author/Editor	Description/Comments
1	27/04/2007	Sam Guinea	The document is opened and organized
2	17/05/2007	Alex Campi	Specification of PLQL
3	4/06/2007	Stefaan Ternier	PLQL implementation and experimentation
4	6/06/2007	Stefano Ceri	Consolidation
5	6/06/2007	Alex Campi	Format fixing
6	11/06/2007	David Massart	Addition of SQL mapping and EUN experience
7	11/06/2007	Stefano Ceri	Final pass prior to peer review submission
8	10/07/2007	Alex Campi	First revision due to first reviewer's comments
9	15/07/2007	Alex Campi	Second revision due to second reviewer's comments
10	18/07/2007	Stefano Ceri	Further revision of the document
11	22/07/2007	Stefaan Ternier	Final fix of examples
12	24/07/2007	Erik Duval	Final iteration over conclusion

Legal Notices

The information in this document is subject to change without notice.

The Members of the PROLEARN Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the PROLEARN Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

1. Introduction

This deliverable provides the definition of the ProLearn Query Language (PLQL), a “query language for repositories of learning objects” defined in the context of the ProLearn Network of Excellence (WP4).

Previous to PLQL, ProLearn has produced the definition of SQL, "Simple Query Interface", a query transport standard that is becoming widely used within the e-learning community. One of SQL's distinguishing features is to be (on purpose) agnostic about query languages; as such, it can be used together with any query language, but does not include any provision for expressing the query semantics. Going beyond SQL requires adding support to a specific query language, designed with the specific objective of retrieving learning objects (LOs) from collections of possibly heterogeneous learning object repositories.

PLQL aims at covering such gap. It is primarily a “query interchange format”, used by source e-learning applications (or PLQL-clients) for querying LOs repositories (or PLQL-servers); each LO can be described by means of metadata, which may be compliant to the most popular standards, such as Dublin Core, LOM, or Mpeg. For submitting the query and retrieving the results, the application can use the SQL protocol as well as other interoperability standards. It is up to the PLQL-client to define user interfaces; these can range from highly sophisticated interfaces to simple keyword-based forms. It is up to the PLQL-server to build PLQL-adapters to the local repository engines. The PLQL-client addresses its queries to one PLQL-server at a time; the application logics, including data integration if needed, should be programmed within the system hosting the PLQL-client.

In defining PLQL, we aim at combining exact search, used for selecting LOs by using their metadata, and approximate search, used for extracting LOs by means of approximate descriptions of their content (e.g., obtained by indexing text of the learning object itself). Thus, a query in PLQL contains both "exact clauses" and "approximate clauses", where each clause is syntactically well-defined; typically, exact clauses query metadata of known structure, while approximate clauses perform ranked retrieval, typically by using document indexing based upon their content.

The result of a PLQL query is normally a set of URI pointing to documents, possibly augmented with meta-information describing them; the actual retrieval of each document is a task that should be performed by the PLQL-client. If the query includes approximate clauses, then the result is an ordered set; ordering of result URIs is determined by the PLQL-server and takes into account their matching against approximate clauses. We give a precise description of the language's semantics concerning both kinds of clauses and their mutual relationship; such semantics is classic when a query contains only clauses of one kind. The most challenging aspect of the language's semantics was to define the meaning of a query with both query aspects. We anticipate that exact search prevails, and that ranking occurs among all result elements which satisfy the exact search criteria..

PLQL is based upon existing language paradigms, and actually aims at minimizing the need of introducing new concepts. Specifically, we have borrowed approximate search

concepts from CQL [CQL], a well-established language used for library search. Given that an XML description is available for all dominant metadata standards of learning objects, we have next decided to express exact search by using object-oriented paths to navigate hierarchies, easily translated to Xpath when needed.

The implementation of PLQL can therefore use existing technology for both exact search (using, e.g., XML-based or relational technology) and approximate search (using, e.g., information retrieval engines). We envision language implementations that consist primarily in syntax-directed translators of the two kinds of clauses to the corresponding engines, while the adopters of PLQL will not be concerned with "query optimization". Sophisticated query processing capability is instead supported by existing products; for instance, database engines may get high performances (e.g. through parallelism), while search engines may go much beyond keyword matching, (e.g. by measuring semantic distance between keywords or using fuzzy word matching).

2. PLQL Levels

In designing PLQL, we aim at supporting also very simple repositories. Thus, one of the main concerns of the language design is providing progressive levels, supporting increasingly expressive power, so that even simple repositories can support the lower levels. The structure of the result returned by a PLQL query is also defined by levels, and it will be described in a dedicated section of this document.

Level 0

Level zero is very basic and corresponds to simple approximate search. Level two is the richest level; future versions of the specifications may address levels beyond two, if additional complexity will be required by the applications. For each level we indicate: 1. expressive power, 2. syntax, 3. Examples

Expressive power

This layer enables the expression of conjunctive approximate queries. The target must contain all the search terms specified in the query, which can be present either in the metadata descriptions, or in the LO as represented through suitable information retrieval structures (e.g., indexes). When several approximate clauses are presented in the same query, they are considered in conjunction, therefore LOs must have all the keywords presented within an approximate clause in order to be selected; selected LOs are ranked according to the cumulative relevance of the keywords, the ranking is performed by the search engine supported on the server.

Layer zero offers the same expressive power as VSQL [VSQL], a very simple (and limited) query language commonly supported by many SQL targets. VSQL belongs to some of the authors' body of background work. As a language, it has achieved in being a common denominator for SQL targets, and can be considered a motivation for the introduction of PLQL. As an example, a VSQL query looks like:

```
<simpleQuery>
  <term>learning object</term>
  <term>dog</term>
</simpleQuery>
```

Syntax

Following is the Backus Naur Form (BNF) definition [BNF] for level 0.

0-1: PLQLQuery ::= approximateClause

0-2: approximateClause ::= operand | '(' approximateClause ')' | approximateClause 'and' approximateClause

0-3: operand ::= term1 | term2 | integer | real

0-4: term1 ::= charString1

0-5: term2 ::= charString2

0-6: charString1 ::= Any sequence of characters that does not include any of the following:

- * whitespace
- * tab
- * ((open parenthesis)
- *) (close parenthesis)
- * =
- * <
- * >
- * "" (double quote)
- * /
- * \
- * .

If the final sequence is the reserved word 'OR' (case insensitive), its token is returned instead (in order to avoid the improper use of this logical connector).

0-7 charString2 ::= Double quotes enclosing a sequence of any characters except double quote (unless preceded by backslash (\)). Backslash escapes the character following it. The resultant value includes all backslash characters except those releasing a double quote (this allows other systems to interpret the backslash character). The surrounding double quotes are not included.

0-8: integer ::= [0-9]+

0-9: real ::= [0-9]*\.[0-9]+

Each PLQL level is identified by a URI, allowing a SQI target to specify its degree of PLQL support. In this case, level 0 is identified with the following URI: <http://www.prolearn-project.org/PLQL/0>.

Examples

1. Correct queries

The following queries are correct PLQL level zero expressions:

```
"dog"  
"learning object" and dog  
dog and cat and jaguar  
(dog and cat) and jaguar  
"Lom.general.title" and "my dog"  
1.2 and dog  
test and 1024  
"12.25 dog"  
"hello" he said"
```

2. Incorrect queries

These examples are incorrect expressions (that cannot be submitted to a repository using PLQL level 0):

```
"learning object" or "dog" // or is not allowed here  
"learning object" dog // connector missing  
Lom.general.title = "dog" // paths not allowed here  
Lom.general.title or dog // paths and or not allowed here  
wrong"
```

Level 1

Expressive Power

In level one, in addition to the approximate searches supported by level 0, PLQL queries can express exact searches on metadata fields. The latter are denoted by means of paths. Level 1 supports paths as simple concatenations of elements (separated by dots), starting from the root, with no omission; expressions and parentheses are not allowed.

Level 1 only supports the following roots (lowercase): 'dc' (Dublin Core Metadata Element Set [3]), 'lom' (Leaning Object Metadata [4]), and 'mpeg' (Moving Picture Experts Group [5]). Generic namespaces are not supported.

This level is unaware of “types”, and attribute values cannot be composed. However, encoded strings that represent diverse datatypes are allowed, given that their meanings can be clarified by referencing URI-identified meta-schemas. Similarly, we allow for the meaning of certain expressions to be clarified by referring to these meta-schemas.

When several exact clauses are presented in the same query, they are considered in conjunction. When both exact and approximate clauses are present in a single query, it is assumed that the exact search has a higher priority than the approximate search. The semantics of PLQL when both exact and approximate clauses are present is to apply the exact clauses first to build an initial result set, then to apply the approximate clauses to the initial result set. This produces a final result set.

However, exact queries might not parse correctly against the metadata available at the storage server. When some exact clauses cannot be parsed by the server, a return code should indicate each of them as "not executed". In particular, if no exact clauses can be parsed in the repository metadata, then the effect of the exact search is null; the repository should operate on the entire set of LOs as if no exact search had been performed.

As a variant to this semantics, requested by the application at query presentation time, the repository could be allowed to use the constant values in the exact clauses that are not parsed correctly as free keywords, so as to perform an approximate search based upon the terms indicated in the exact clauses; a return code should then indicate to the application that this case has occurred. Such variant should be evaluated experimentally, to see if it can be useful at least in certain contexts. Note that a repository unable to process exact queries against certain metadata could always resort to such query interpretation.

Syntax

Note that productions with the same number as productions at lower levels substitute for them, e.g. production 1-1 substitutes (generally extends) production 0-1.

```
1-1: PLQLQuery ::= exactclause | approximateclause | exactclause ';' approximateclause
1-10: exactclause ::= pathexpr | '(' exactclause ')' | exactclause 'and' exactclause
1-11: pathexpr ::= standard '.' path operator operand
1-12: path ::= term1 | path '.' path
1-13: operator ::= '='
1-14: standard ::= 'dc' | 'Lom' | 'mpeg'
```

This level of PLQL is identified with the following URI: <http://www.prolearn-project.org/PLQL/1>

Examples

1. Correct queries

The following queries are correct PLQL level 1 expressions:

```
dc.title = "SQL" and Lom.general.title = "SQL"

Lom.general.title = "Design Patterns" and Lom.general.language = "en"

Lom.general.title = "Design Patterns" and Lom.technical.format = "video/mpeg"
and Lom.technical.duration = "PT1H" and Lom.rights.cost="no"

Lom.general.title = "Design Patterns" and Lom.educational.intendedEndUserRole
= "learner" and Lom.educational.typicalAgeRange = "15-18"

((Lom.general.title = abc) and (Lom.general.language="fr")) ; test

tiger

keyword1 and keyword2 and (Lom.general.language = "fr" )
    and (Lom.educational.ageRange="10-12")

keyword1 and keyword2 and (Lom.general.language=en)and(Lom.educational.ageRange=10-12)
```

2. Incorrect queries

The following queries are incorrect PLQL level 1 expressions:

```
dc = 12
Lom.general.title // incomplete
Lom.general.(title = "abc") // uses parenthesis as only allowed at level 2
Lom.general.(title = "abc" and language="fr") // uses parenthesis as only allowed at level 2
tiger or Lom.general.title = "abc" // uses disjunction
```

Level 2

Expressive Power

Compared to levels 0 and 1, level 2 increases the expressive power of supported queries, by enabling disjunction in addition to conjunction; moreover, clauses may use arbitrary comparison predicates. With approximate search, we use the "=" symbol to denote the 'includes' operator and the "exact" symbol to denote exact string matching. Level 2 enables a limited amount of structuring of exact clauses, by supporting parenthesization within path expressions; in this way, it is possible to descend a hierarchical structure up to given nodes and then build conditions which are based upon

the properties of several descendants of that node. Finally, level 2 opens to generic namespaces.

Syntax

```
2-2: approximateClause ::= operand | '(' approximateClause ')' | approximateClause boolean
approximateClause

2-10: exactclause ::= pathexpr | '(' exactclause ')' | exactclause boolean exactclause

2-11: pathexpr ::= (standard . )? path operator operand | pathExp

2-13: operator ::= '=' | '>', '>=', '<', '<=', 'exact'

2-14: standard ::= 'dc' | 'Lom' | 'mpeg' | term1

2-15: pathExp ::= path '.' pathExp | '(' selector boolean selector ')'

2-16: selector ::= path operator operand | selector boolean selector | '(' selector ')' | '(' selector
boolean selector ')'

2-17: boolean ::= 'and' | 'or'
```

This level of PLQL is identified with the following URI: <http://www.prolearn-project.org/PLQL/I2>

Examples

1. Correct queries

```
Lom.general.identifier.(catalog=isbn and entry=xxxxx)

Lom.general.(title = "Design Patterns" and (language = "it" or language = "en"))

Lom.general.title = "Design Patterns" and Lom.technical.(format = "video/mpeg"
and duration <= "PT1H") and Lom.rights.cost="false"

Lom.general.title = "Design Patterns" and
Lom.educational.(intendedEndUserRole = "learner" or typicalAgeRange = "15-18")
```

2. Incorrect queries

The following queries are incorrect PLQL level 2 expressions:

```
Lom.general.(title = "abc") // abuse of parenthesis

Lom.general.title = ("abc" and tiger) // incorrect clause
```

3. Query Results

The result produced after invoking PLQL on a repository also is built by means of “levels”; the information returned by the server may include just the number of selected items in the results up to more specific meta-information about each item. Currently in PLQL we support four levels (ranging from 0 to 3).

1. Input

Any query issued in PLQL should be associated with (through the query transport method) an input parameter (ResultLevel:0-3) indicating to the target repository the level of the result that should be returned by the query. In addition, three optional parameters can specify:

- the maximum cardinality (MaxCard:integer) of the result;
- the name of the search method (Method:string) to be used at server side for result extraction, when the client has such choice.
- the name of the standard (Standard:'dc'|'Lom'|'mpeg') used for returning meta information about the items in the results

2. Result

Results are defined, as with the queries, by means of progressive levels (the higher levels include the lower ones).

At level zero, sources return at least the cardinality of the result.

At level one, sources return in addition at least the list of URIs of the elements which are extracted by the query. Retrieving the actual object referenced through the URI is left to the application. If the result is ranked, the best results must appear first. This is identified by the presence of an appropriate XML attribute (added to the result) called “rankingValue”.

At level two, sources return in addition some specific metadata of the requested metadata format (e.g., Lom, dc, etc.). We do not define the metadata as part of the PLQL standard, but we expect them to include the title, author, and language.

At level three, sources return in addition a numeric ranking value, and if the source supports it a reference (identifier) to the ranking method.

This information is summarized below:

ResultLevel 0: result.cardinality	type integer - size of result
ResultLevel 1: result.list	type array of 0-MaxCard ranked records
result.list[i].URI	URI of selected resource
ResultLevel 2: result.list[i].meta	record of metadata fields
	normally: position,title, language,author
ResultLevel 3: result.method	type string - method used for scoring
result.list[i].rankingValue	type integer - in 0-100 range, giving matching score

We expect most sources to be able to support at least level one (i.e., to return URI ordered according to their ranking).

Syntax

Using the method `setResultFormat` of SQI, a source can use a URI to inform a target of the expected level of the result. Optionally, the URI also indicates also the expected format of metadata. Its BNF is:

```
1: PLQLRES ::= 'http://www.prolearn-project.org/PLRF/' level '/' standard [ / 'method' ]
2: level ::= '0'|'1'|'2'|'3'
3: standard ::= 'dc'| 'Lom' | 'mpeg'
4: method ::= string
```

Result examples

The examples below illustrate result data in XML format.

Result level 0

Suppose that a source wants the following PLQL level 1 query to return level 2 ranked results:

Lom.general.(title = "Design Patterns" and (language = "it" or language = "en"))

For such a query one would first set the query language (`setQueryLanguage` in SQI) to

<http://www.prolearn-project.org/PLQL/l2>

Next, the method `setResultsFormat` of SQI sets the format to the following URI:

<http://www.prolearn-project.org/PLRF/0>

0 indicates the level and specifies that only the amount of results should be returned. It is hence not necessary to specify a metadata standard in the results format.

Finally, after this query is submitted using the `synchronousQuery` method, the target could return the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.prolearn-project.org/PLQLRES/
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.prolearn-project.org/PLRF/ http://www.cs.kuleuven.be/~stefaan/plql/plql.xsd
http://ltsc.ieee.org/xsd/LOM http://ltsc.ieee.org/xsd/lomv1.0/lom.xsd"
xmlns="http://www.prolearn-project.org/PLRF/">
  <ResultInfo>
    <ResultLevel>http://www.prolearn-project.org/PLRF/0</ResultLevel>
    <QueryMethod>http://www.prolearn-project.org/PLQL/l2</QueryMethod>
    <Cardinality>38</Cardinality>
  </ResultInfo>
</Results>
```

Result level 1

Suppose that one wants to execute a PLQL query with the following configuration:

Query Language: <http://www.prolearn-project.org/PLQL/I0>

Result format: <http://www.prolearn-project.org/PLRF/1/lom>

Query: "learning object" and dog

After submitting the query and results format, the result would now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.prolearn-project.org/PLRF/ http://www.cs.kuleuven.be/~stefaan/plql/plql.xsd
    http://ltsc.ieee.org/xsd/LOM http://ltsc.ieee.org/xsd/lomv1.0/lom.xsd"
  xmlns="http://www.prolearn-project.org/PLRF/">
  <ResultInfo>
    <ResultLevel>http://www.prolearn-project.org/PLRF/1/lom</ResultLevel>
    <QueryMethod>http://www.prolearn-project.org/PLQL/I0</QueryMethod>
  </ResultInfo>
  <Record>
    <Metadata>
      <lom xmlns="http://ltsc.ieee.org/xsd/LOM">
        <general>
          <identifier>
            <entry>ARID43_12395</entry>
            <catalog>ARIADNE</catalog>
          </identifier>
        </general>
      </lom>
    </Metadata>
  </Record>
  <Record>
    <Metadata>
      <lom xmlns="http://ltsc.ieee.org/xsd/LOM">
        <general>
          <identifier>
            <entry>ARID43_12395</entry>
            <catalog>ARIADNE</catalog>
          </identifier>
        </general>
      </lom>
    </Metadata>
  </Record>
  ...
</Results>
```

Note that that, as <http://www.prolearn-project.org/PLRF/1/lom> was given as a resultsFormat, the URI's are encoded as LOM identifiers.

Result level 2

Given are the following query parameters:

Query Language: <http://www.prolearn-project.org/PLQL/I0>

Result format: <http://www.prolearn-project.org/PLRF/2/dc>

Query: "Germany and its Tribes" and Tacitus

As a result, the target can return the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xsi:schemaLocation="
    http://www.prolearn-project.org/PLRF/ http://www.cs.kuleuven.be/~stefaan/plql/plql.xsd
    http://purl.org/dc/elements/1.1/ http://dublincore.org/schemas/xmls/simpledc20021212.xsd
    http://www.openarchives.org/OAI/2.0/oai_dc/ http://www.openarchives.org/OAI/2.0/oai\_dc.xsd"
  xmlns="http://www.prolearn-project.org/PLRF/">
  <ResultInfo>
    <ResultLevel>http://www.prolearn-project.org/PLRF/2/dc</ResultLevel>
    <RankingMethod>NrOfDownloads</RankingMethod>
    <QueryMethod>http://www.prolearn-project.org/PLQL/I0</QueryMethod>
  </ResultInfo>
  <Record position="1">
    <Metadata>
      <oai_dc:dc>
        <dc:title>Germany and its Tribes</dc:title>
        <dc:creator>Tacitus</dc:creator>
        <dc:language>en</dc:language>
        <dc:source>Complete Works of Tacitus. Tacitus. Alfred John Church. William Jackson Brodribb. Lisa Cerrato. edited for Perseus. New York: Random House, Inc. Random House, Inc. reprinted 1942. </dc:source>
        <dc:identifier>http://www.perseus.tufts.edu/cgi-bin/ptext?doc=Perseus:text:1999.02.0083</dc:identifier>
      </oai_dc:dc>
    </Metadata>
  </Record>
  ...
</Results>
```

Result level 3

Given are the following query parameters:

Query Language: <http://www.prolearn-project.org/PLQL/I1>

Result format: <http://www.prolearn-project.org/PLRF/3/lom>

Query: lom.general.language = "en"; code

As a result, the following can be returned by the target.

```
<?xml version="1.0" encoding="UTF-8"?>
<Results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.prolearn-project.org/PLRF/ http://www.cs.kuleuven.be/~stefaan/plql/plql.xsd
    http://ltsc.ieee.org/xsd/LOM http://ltsc.ieee.org/xsd/lomv1.0/lom.xsd"
  xmlns="http://www.prolearn-project.org/PLRF/">
```

```
<ResultInfo>
  <ResultLevel>http://www.prolearn-project.org/PLRF/3/lom</ResultLevel>
  <RankingMethod>NrOfDownloads</RankingMethod>
  <QueryMethod>http://www.prolearn-project.org/PLQL/11</QueryMethod>
  <Cardinality>5</Cardinality>
</ResultInfo>
<Record position="1" rankingValue="80">
  <Metadata>
    <lom xmlns="http://ltsc.ieee.org/xsd/LOM">
      <general>
        <identifier>
          <entry>ARID43_12395</entry>
          <catalog>ARIADNE</catalog>
        </identifier>

        <title>
          <string language="en">The Da Vinci Code</string>
        </title>
        <language>en</language>
      </general>
      <technical>
        <location>http://ariadne.cs.kuleuven.be/AWS/files/davincicover.jpg</location>
      </technical>
    </lom>
  </Metadata>
</Record>
...
</Results>
```

More examples are available at: <http://www.cs.kuleuven.be/~stefaan/PLQL/>

4. Mapping PLQL to metadata management systems

This section explains how PLQL can be mapped to the following metadata management paradigms.

- RDBMS: a mapping from PLQL level 0 to SQL will be contributed.
- Lucene: a mapping from both PLQL level 0 and level 1 to Lucene, a full featured text search engine will be contributed.
- XML database management systems. Both PLQL level 0 and level 1 have been mapped to XQuery.

First, an approach for creating compilers using flex/yacc will be outlined. This will give perspective adopters of PLQL reusable information about how we proceeded in our implementation. Next, the following 3 subsections will deal with the specific mappings to SQL, the Lucene query language and XQuery.

Parsing PLQL with flex/yacc

Flex (fast lexical analyzer generator) is a lexical analyzer generator. These lexical analyzers can recognize lexical patterns in text (PLQL queries in this case). Flex is often used in combination with yacc (Yet Another Compiler Compiler) [FlexYacc], a computer program that can generate parsers. Used together, a lexical analyzer will generate tokens out of an input stream, which are parsed with a parser (generated with yacc). This section describes how analyzing and parsing of PLQL queries was done using flex/yacc technology.

For each level of PLQL, general purpose lexical analyzers have been generated that will later serve for creation parsers for specific. All this code is open source and available on the PLQL wiki page:

http://ariadne.cs.kuleuven.be/Lomi/index.php/QueryLanguages_v1.0

A java flex file is organized in three parts, as illustrated below:

```
User code
%%
options and declarations
%%
lexical rules
```

The first part (“user code” section) contains text that is copied to the top of the generated lexer class. The following code illustrates this for the PLQL level 0 lexer:

```
package org.eun.PLQL.layer0;
import java.io.*;
%%
```

Next, the “options and declarations” section consists of options that allow customizing the generated lexer and the declarations of macro’s and lexical states:

```
%class PLQLLayer0Parser
%byaccj
%unicode

%{
    private String temp;

    public PLQLLayer0Parser(java.io.Reader r, PLQLLayer0Analyzer yyparser) {
        this(r);
        this.yyparser = yyparser;    }

}%

CHARSTRING1 = [^ .\t"()=<>\]+
CHARSTRING = [^"]
NL = \n | \r | \r\n
AND = [aA][nN][dD]
LEFT_PATENTHESIS = "("
RIGHT_PATENTHESIS = ")"
REAL = [ ][0-9]*\.[0-9]+[ ]
INTEGER = [ ][0-9]+[ ]

%state STRING2
```

```
%%
```

The “lexical rules” section contains rules (expressed as regular expressions) and actions that are to be executed when a scanner matches the associated regular expression. The following excerpt from the “lexical rules” section illustrates this:

```
/* Match charString2*/
<YYINITIAL>\|" {yybegin( STRING2 ) ; temp = "" ; }
<STRING2> {
\\|"          {temp += "\\|" ;}
{CHARSTRING} {temp += yytext() ;}
\|"          {
              yybegin( YYINITIAL ) ;
              yyparser.yylval = new PLQLLayer0AnalyzerVal(temp);
              return PLQLLayer0Analyzer.CHARSTRING2;
            }
}
```

The rules presented in this excerpt present how one can parse a charString2 and can only be matched when the parser is within the lexical state. The lexical state YYINITIAL is predefined and it is the state in which the lexer begins scanning. When the scanner matches a double quote, the state is changed using the yybegin method to STRING2 and the variable temp that was defined earlier is reset to an empty string. As long as the lexer is in this STRING2 state, the following regular expressions can match:

- \\|" matches against backslash-double quote and concatenates this to the temp variable.
- {CHARSTRING} matches against the CHARSTRING regular, and adds this text to the temp variable.
- \|" The next occurrence of a double quote terminates the charString2. The state is again set to YYINITIAL. The parsed value is assigned to the ylval field so that it can be read by read by yacc. Finally, the CHARSTRING2 code is submitted to the yacc parser.

Mapping to SQL

Using the PLQL syntax for PLQL level 0 described in section 2, it was trivial to write the parser that turns PLQL 0 queries into SQL (The Parser can be seen in Appendix A). However, some interoperability issues were due to differences between SQL supported at the repositories. Unlike the lex-generated tokenizer that can be reused by all the PLQL parser, only the rule-part of the yacc parser (i.e., the part that describes the grammar) does not need to be changed. The actions attached to each rule, responsible for generating repository-specific queries, need to be adapted for each repository.

Mapping to the Lucene query language

Lucene [Lucene] supports data with fields. Each document that is indexed in Lucene has a number of fields with corresponding values associated to them. Unlike XML, Lucene imposes a flat metadata structure on its document which makes full support for PLQL level 2 very difficult to achieve.

The following table illustrates how a simple LOM metadata instance is mapped on a Lucene document:

Lom instance	Lucene document	
	Fieldname	Value
<pre><Lom xmlns="http://ltsc.ieee.org/xsd/LOM"> <general> <title> <string language="en"> Pascal's Law</string> </title> <language>pl</language> <keyword> <string language="en"> piston</string> </keyword> <keyword> <string language="en"> pressure of a liquid</string> </keyword> </general> </Lom></pre>	<pre><Lom.general.title.string> <Lom.general.title.language> <Lom.general.language> <Lom.general.keyword.string> <Lom.general.keyword.language> <Lom.general.keyword.string> <Lom.general.keyword.language></pre>	<pre>Pascal's Law en pl piston en pressure of .. en</pre>

As a result of this mapping, the hierarchical structure of the original metadata instance is lost. However, as this serialization into Lucene documents is straightforward, setting up a Lucene index that offers native support for level 0 and 1 is now easy to achieve.

The following table exemplifies how PLQL queries can be mapped on this structure.

PLQL level	PLQL query	Corresponding Lucene query
0	"learning object" and dog	"learning object" AND "dog"
1	Lom.general.title.string = "Design Patterns" and Lom.general.title.language = "en"	Lom.general.title.string:"Design Patterns" AND Lom.general.title.language:"en"

PLQL level 0 is very easy to map to Lucene and only requires the operator to be put in capitals. In Lucene, a query can consist of different terms combined together with Boolean operators. A Lucene term can be optionally preceded with the field name, instructing the Lucene engine to search in the given field only. For PLQL level 1, a path (eg. Lom.general.title.string) is hence mapped on such a Lucene field, while the operand is mapped to the Lucene term.

Mapping to XQuery

This section will deal with the mapping of PLQL level 0 and 1 to XQuery. The advantage of using an XML database management system with XML support is that translating PLQL level 2 to XQuery is possible, as suggested by the following examples:

PLQL level	PLQL query	Corresponding XQuery
0	"learning object" and dog	<pre>xquery version "1.0"; <results> { for \$Lom in collection("null")/Lom where contains(\$Lom,"learning object") and contains(\$Lom,"dog") return \$Lom } </results></pre>
1	Lom.general.title = "Design Patterns" and Lom.general.language = "en"	<pre>xquery version "1.0"; <results> { for \$Lom in collection("null")/Lom where contains(\$Lom/general/title,"Design Patterns") and contains(\$Lom/general/language,"en") return \$Lom } </results></pre>

The mapping to XQuery is, at the time of writing, still work in progress. Future versions of this mapping will offer support for:

- Conjunction of exact and approximate clauses as specified by PLQL level 1. The current implementation is still based on the 0.8 version of the PLQL specification (http://ariadne.cs.kuleuven.be/Lomi/index.php/QueryLanguages_v0.8)
- PLQL level 2.
- The query+result language that is defined in this deliverable. Currently, this mapping still results in an XML "result", listing each individual LOM instance that matches the query.

5. Experimentation of PLQL

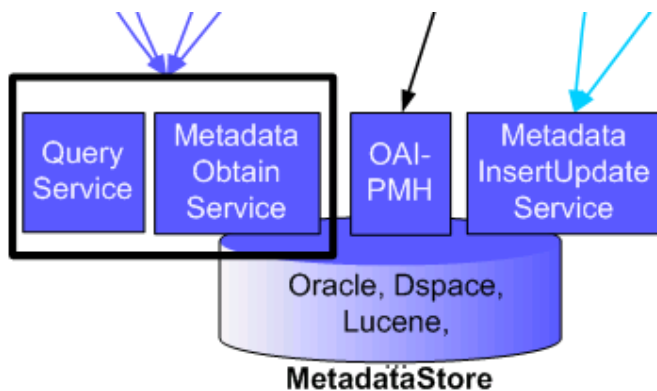
This section sheds light on the current implementations of PLQL; all implementations were done on IEEE LOM compliant repositories.

PLQL in ARIADNE

PLQL level 0 and 1 have been implemented on top of the ARIADNE repository component. The source code of this implementation is available on sourceforge: (<http://ariadnekps.svn.sourceforge.net/viewvc/ariadnekps/repository/>).

The following figure is part of the ARIADNE's AriadneNext architecture and presents the ARIADNE metadata store. The entire architecture is available on:

<http://ariadne.cs.kuleuven.be/mediawiki/index.php/AriadneNextArchitecture>.



This metadata store has recently been implemented on top of an eXist XML store that implements XQuery as a query language. The Query Service and Metadata Obtain Service are thus implemented as an SQL service that offers support for PLQL level 0 and 1 and incorporates the “PLQL to XQuery” mapping, discussed earlier. The second interface, the OAI-PMH harvest interface, will be discussed in the next section. The Metadata Insert & Update Service does not fall within the scope of this deliverable.

OAI-PMH on top of SQI/PLQL

The Open Archives Initiative OAI-PMH is a protocol for metadata harvesting through which harvesters can copy and obtain the metadata records of a repository.

This protocol can select records based on three features:

- A unique identifier that identifies an item within a (metadata) repository
- A datestamp representing the date when the record was last modified.
- A record can be part of certain sets, that can be used to selectively harvest records from a repository.

The OAI-PMH interface on the ARIADNE metadata store was conceived such that a harvesting request can be translated into an SQI/PLQL level 1 request. This strategy makes this OAI-PMH implementation reusable on other SQI/PLQL metadata stores. The OAI-PMH verbs that need to be mapped on PLQL are the following:

- **GetRecord.** This verb is used to retrieve an individual metadata record from a repository. It takes an identifier as argument and uses that to pull the record out of the repository. Such an OAI-PMH request is mapped on the following PLQL query that returns one result:

```
Lom.metaMetadata.identifier.entry = "someId"
```
- **ListRecords.** This verb is used to harvest a repository. Optional arguments such as from (lower bound for datestamp), until (upper bound for datestamp) and set enable selective harvesting. For now, only support has been implemented for datestamp-based selective harvesting. The following example PLQL query is currently used for this purpose:

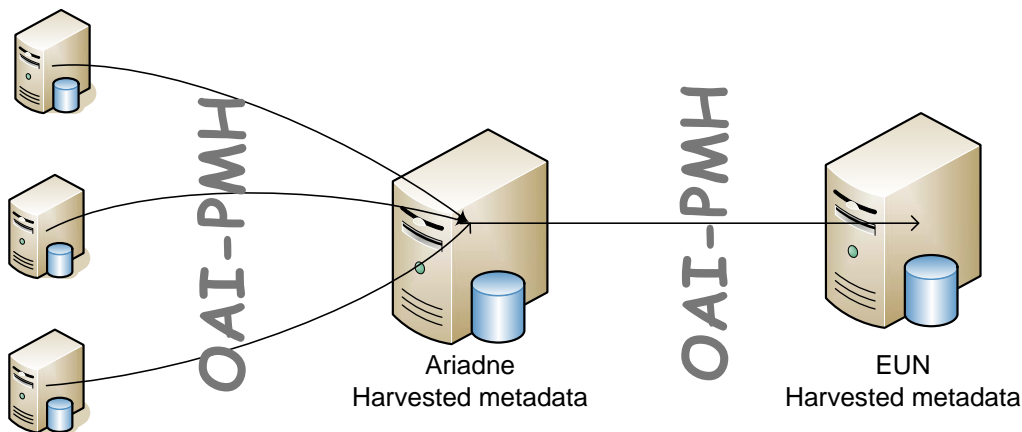
```
Lom.metaMetadata.contribute.date.dateTime > 2000-01-01
and Lom.metaMetadata.contribute.date.dateTime < 2006-05-05
```
- **ListIdentifiers.** This verbs offers similar characteristics to ListRecords but only lists the header records including the identifier and the datestamp.

As a query results format, this SQI/PLQL to OAI-PMH gateway uses result level 2. Note that, not supporting level 2 PLQL queries imposes restrictions on the LOM application profile that is used at a repository, i.e. a contributed element can appear only once under the metaMetadata category in LOM.

If two contributed elements were present with the first one having a date lower than the lower bound and the second one having a date higher than the upper bound, this LOM record would match against the PLQL query as both contribute elements would match against one clause. A PLQL level 2 query like the following one, indicates that the dateTime of one of the contribute elements needs to match against both conditions.

```
Lom.metaMetadata.contribute.date.(dateTime > 2000-01-01  
and dateTime < 2006-05-05)
```

Although, support for PLQL level 2 has not been implemented yet, this binding has already been successfully implemented in the MELT eContentPlus project. In this project, the architecture that is illustrated bellow has been implemented.



This architecture is constituted of an intermediate repository, filled with harvested records from MELT content providers that expose metadata again through an OAI-PMH target. This OAI-PMH target runs on the ARIADNE PLQL enabled metadata store (see previous section) and hence implements the gateway, discussed in this section.

PLQL in the EUN Learning Resource Exchange

The EUN Learning Resource Exchange (LRE) is a service that provides the means to unlock the educational content hidden in digital repositories across Europe and share it among all partners of the LRE and their users. The service is offered to actors providing digital content: Ministries of Education, regional educational authorities, commercial publishers, broadcasters, cultural institutions and other non-profit organisations who are offering extensive but heterogeneous catalogues and repositories of online content to schools.

From a technical standpoint, the LRE is a federation of heterogeneous learning resource repositories. These repositories rely on technologies as diverse as relational databases, XML databases, text file indexers and other ad hoc solutions to manage their metadata. The LRE uses an LRE LOM application profile both as a common metadata exchange format and as a federated schema (i.e., unified data schema) for getting access to the federation.

Given the requirements of the LRE users (i.e., teachers, pupils) in terms of learning resource discovery, a LRE query language should allow for expressing the following 12 basic query types (and their Boolean combinations: and, or, not). Search for:

1. Resources in a given language (LOM 1.3)
2. Resources described by metadata in a given language
3. Resources of a given structure
4. Resources with a given status (e.g., draft)
5. Resources targeting an audience within a given age range
6. Resources with a given creative commons license
7. Resources with a given author
8. Resources of a given "learning resource type"
9. Resources with a given creation date
10. Resources of a given mime-type
11. Resources covering a given curriculum subject (competencies)
12. Resources described with a given thesaurus descriptor

Among them, queries 1, 2, and 10 can be expressed in PLQL layer 1.

Query Type Number	Example of Query of This Type	Query Example in PLQL Level 1
1	Resources in English	lom.general.language=en
2	Resources described by metadata in English	lom.metametadata.language=en
10	Gif images	lom.technical.format = image/gif

Queries 3, 4, 5, 6, 8 can only be expressed in PLQL layer2.

Query Type Number	Example of Query of This Type	Query Example in PLQL Level 2
3	Atomic resources	lom.general.structure.(source="LOMv1.0" and value="atomic")
4	Resources that are draft	lom.lifeCycle.status.(source="LOMv1.0" and value="draft")
5	Resources intended to pupils from 12 to 13	lom.educational.typicalAgeRange.(string = 12-13 and

		language = x-t-lre)
6	Attribution share-alike resources	lom.rights.description.(string = "by-sa" and language = "x-t-cc")
8	Exercises	lom.educational.learningResourceType.(source=LREv3.0 and value="exercise")

The remaining 4 queries (i.e., 7, 9, 11, and 12) involve constraints between different LOM elements that cannot be expressed directly with the current version of PLQL. To overcome this limitation, an LRE Application Profile of PLQL level 2 was created that defines a new ad hoc "lre" context in addition to the three contexts already supported by PLQL (i.e., "lom", "dc", and "mpeg"). As showed in the table below, the lre context was also used to propose a simplified alternative for queries only expressible with PLQL level 2.

Query Type Number	Examples of queries of this type	Query examples expressed in PLQL Level 2	Query examples expressed in LRE PLQL
3	Atomic resources	lom.general.structure (source="LOMv1.0" and value="atomic")	lre.structure = atomic
4	Resources that are draft	lom.lifeCycle.status.(source="LOMv1.0" and value="draft")	lre.status=draft
5	Resources intended to pupils from 12 to 13	lom.educational.typicalAgeRange. (string = 12-13 and language = x-t-lre)	lre.typicalAgeRange= 12-13
6	Attribution share-alike resources	lom.rights.description.(string = "by-sa" and language = "x-t-cc")	lre.cc=by-sa
7	Resources by Frans Van Assche	NA	lre.author="Frans Van Assche"
8	Exercises	lom.educational.learningResourceType.(source=LREv3.0 and value="exercise")	lre.learningResourceType = "exercise"
9	Resources created on April 4, 2004	NA	lre.creationDate= 2007-04-04
11	Resources addressing competency [act_3, top_5, top_7]	NA	lre.competency = [act_3,top_5,top_7]
12	Resources indexed with LRE thesaurus descriptor 195	NA	lre.discipline = 195

As a proof of concept, a first repository supporting the LRE PLQL profile was implemented using LUCENE and connected to a test instance of the LRE. It can be accessed at <http://stove.test.eun.org:9080/LRE-Search/>. Currently, this prototype returns results in layer 0 and strict LOM (the default result format supported by the LRE). The LRE PLQL was presented in Brussels to the developers of the CALIBRATE project on June 11, 2007 and should be supported by most of the LRE repositories by the end of September 2007.

6. Related works

The work presented in this deliverable is the result of a long stream of research. A fundamental step in this research is the ARIADNE project [ARIADNE]. The core of this project is a distributed network of learning repositories. ARIADNE offers 3 tier architecture: at the bottom a repository enables searching, publishing and retrieving learning objects; an API, that is bound to web services, enables loosely coupled manipulation of the bottom layer, applications such as the moodle plugin or the ALOCoM office plugin, make the knowledge pool transparently accessible from within third party applications. A lot of work [ARIADNE, 2001] was devoted to increase the interoperability between ARIADNE and other Learning Object Repositories that rely on IEEE LOM. To achieve this goal, ARIADNE profile has been mapped into the LOM structure. XSLT has been used in order to transform ARIADNE XML into IEEE LOM XML.

ARIADNE is not the only project devoted to managing learning object metadata. Nowadays, more and more systems face the problem of mapping and representing metadata according to some metadata standard [Rehak, D., 2003]. This mapping allows to share and to exchange learning objects as well as their metadata. Metadata is defined as “information about an object; be it physical or digital” [Duval, E., 2001], it is used to facilitate search, evaluation, acquisition and use of learning objects. Typically, Learning objects and their associated metadata are located in distributed Learning Object Repositories (LOR's). However, there is more than one approved standard used to describe the properties of learning objects, for example the Learning Object Metadata (LOM) standard [IEEE, 2002] and the Dublin Core standard [DCMES, 1999].

Different Learning Object Repositories try to address different needs. Therefore, metadata designers may select a number of metadata elements as well as their related value sets from one or more metadata standards [Heery, R., 2000]. The specification of these metadata elements and value sets is called an “application profile”. For simplicity, we will use the term ‘Profile’ instead of ‘application profile’ in this paper. Profiles are used to adapt metadata specifications to the requirements of the local community such as multilingual and multicultural requirements [Duval, E. et al, 2002]. Therefore, each of the Learning Object Repositories uses a different profile to define learning objects. Examples of such profiles are the metadata sets of CanCore [CanCore, 2002], SingCore [SingCore, 2002], SCORM [ADL, 2001]. Learning Object Repositories aim to share and reuse metadata. Therefore, syntax and semantics of metadata elements, as well as their value sets, should be represented according to a particular metadata standard. To do so, we need to conceptually map data elements of different profiles and their related values into elements and values of a standard schema, and represented in a semantically interoperable [Euzenat, J., 2001] and technically sharable representation such as XML or RDF [IEEE RDF, 2002].

The Networked Digital Library of Theses and Dissertations (see www.theses.org) is more limited than ARIADNE in scope, as it is restricted to theses and dissertations from around the world. Architecturally, it operates on a rather different model from the KPS of loosely coupled

sites, so searches need to be federated or based on harvesting, as in the Open Archives Initiative [OAI]. The Computer Science Teaching Center [CSTC] is more similar to the ARIADNE, as it attempts to build a digital library of computer science teaching resources. Although its review process is more elaborate, its metadata set is much simpler, and the number of documents in CSTC today is limited. The U.S.-based Science, Mathematics, Engineering, and Technology Education initiative is also building a digital library [SMETE]. Registered users can add comments on resources, a feature supported in ARIADNE, but that has attracted limited user activity. It should be relatively straightforward to establish interoperability between ARIADNE and SMETE, as the latter generates LOM instances in XML. The Education Network Australia (EdNA) supports a repository of 10,000 evaluated resources called EdNA Online (see www.edna.edu.au). Its metadata structure is based on the 15 Dublin Core metadata elements, extended with nine EdNA-specific elements. These nine elements relate mainly to meta-metadata and reviews of resources. One pedagogical metadata element—the “user level”—is included, and searching is supported over only five fields. EdNA metadata is typically stored in HTML metatags, and items may be suggested by submitting their URLs.

The metadata structure of the Gateway to Educational Materials (GEM) digital library (see www.geminfo.org) is also based on the Dublin Core metadata element set but includes more pedagogical attributes adopted from the IEEE LTSC LOM work. As in EdNA, only a limited number of elements, in this case five, are searchable. The organizational structure is similar to ARIADNE's, requiring consortium members to not only consume but contribute to the resources accessible through the gateway. Neither EdNA nor GEM include the actual document in the server.

7. Conclusions

In this deliverable we have presented PLQL, ProLearn's proposal for a query language for learning object repositories. This work has followed on the heels of previous achievements in defining SQL (the Simple Query Interface). The interface was born as a unified access point to possibly distributed and heterogeneous repositories. Therefore, PLQL has to provide a unique language that can be used regardless of the many underlying repositories actually being queried. This positions PLQL as a "query interchange format". To accommodate the great diversity of the different repositories and their capabilities, we have shown that PLQL must be able to combine exact search and approximate search. This lead us to define the language hierarchically, using incremental layers of complexity and expressiveness. Moreover, to provide interoperability we have also provided a detailed query result format.

In order to achieve lasting impact with PLQL, we will combine the following approaches:

- We will continue to collaborate with many of the leading repositories (<http://globe-info.org/>) and hope to convince them to adopt PLQL, or at least to provide support for receiving and sending queries in PLQL as an exchange format.
- We will introduce PLQL as a candidate in either the CEN/ISSS Workshop on Learning Technologies, the newly formed CEN Technical Committee on Learning Technologies, or the IEEE Learning Technologies Standards Committee. Preferably, we would initiate this as a joint activity of both of them, but it remains to be seen if this will be feasible.

8. Bibliography

[ADL, 2001]. SCORM Metadata set. Available at: <http://www.adlnet.org>. ARIADNE, 2001. ARIADNE Foundation. Available at: <http://www.ariadne-eu.org/>.

[ARIADNE] <http://www.ariadne-eu.org/index.php>

[ARIADNE 2001] Michael Day, Preservation 2000, report on the International Conference on the Preservation and Long Term Accessibility of Digital Materials, York, 7-8 December 2000. Ariadne, No. 26, January 2001. <http://www.ariadne.ac.uk/issue26/metadata/>

[BNF] http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

[CanCore, 2002]. CanCore Application Profile. Available at: <http://www.cancore.ca>.

[CQL] Common Query Language: <http://www.loc.gov/standards/sru/cql/>

[CSTC] www.cstc.org

[DCMES,1999]. Dublin Core Metadata Element Set v1.1, available at: <http://dublincore.org/documents/1999/07/02/dces/>.

[Duval, E., 2001]. Metadata Standards: What, Who & Why. Journal of Universal Computer Science, Vol. 7, No. 7, pp 591-601, Special Issue: I-Know 01 - International Conference on Knowledge Management.

[Duval, E. et al, 2002]. Metadata Principles and Practicalities. D-Lib Magazine, Vol. 8, No.3.

[Euzenat, J., 2001]. Towards a principled approach to semantic interoperability. IJCAI 2001 Workshop on ontology and information sharing, Seattle.

[FlexYacc] Lex and YACC primer/HOWTO, <http://ds9a.nl/lex-yacc/cvs/lex-yacc-howto.html>

[Heery, R. 2000]. Application profiles: mixing and matching metadata schemas. Ariadne, issue 25. Available at: <http://www.ariadne.ac.uk/issue25/app-profiles/intro.html>.

[IEEE, 2002]. IEEE 1484.12.1-2002. IEEE Learning Object Metadata.

[IEEE RDF, 2002]. IEEE Learning Object Metadata RDF binding. at: <http://kmr.nada.kth.se/el/ims/mdlomrdf.html>

[Lucene] <http://lucene.apache.org/java/docs/index.html>

[OAI] www.openarchives.org

[Rehak, D., 2003]. Rehak, D. and Mason, R., 2003. Keeping the Learning in Learning Objects. Carnegie: Learning Systems Architecture Lab, Mellon University. Available at: <http://www.lsal.cmu.edu/lsal/expertise/papers/>

[SingCore, 2002]. SingCore Application profile. at: <http://www.ecc.org.sg/eLearn/MetaData/SingCORE/index.jsp>.

[SMETE] www.smete.org

[VSQL] Stefaan Ternier, Ben Bosman, Erik Duval, Connecting OKI And SQL: One Small Piece Of Code, A Giant Leap For Reusing Learning Objects. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2006* (pp. 825-831). Chesapeake, VA: AACE

Appendix A: Parser for translating PLQL Level 0 into SQL

```
import java.io.*;
import java.util.StringTokenizer;
%}

%token NL          /* newline */
%token <sval> CHARSTRING1 /* First kind of keyword */
%token <sval> CHARSTRING2 /* Second kind of keyword */
%token <sval> AND /* Second kind of keyword */
%token <sval> LEFT_PARENTHESIS /* Second kind of keyword */
%token <sval> RIGHT_PARENTHESIS /* Second kind of keyword */
%token <ival> INTEGER /* Integer number */
%token <dval> REAL /* real number */

%type <sval> plql
%type <sval> clause
%type <sval> keywordClause
%type <sval> operand
%type <sval> term1
%type <sval> term2
%type <sval> charString1
%type <sval> charString2
%type <sval> integer
%type <sval> real

%start plql

%%

/* rule 0-1 */
plql: clause {
    $$ = "SELECT sto.content \n FROM fire_search_tab sea, fire_store_tab sto
\n where \n sea.xml_id = sto.xml_id \n and " + $1;
    query = $$ ;};
/* rule 0-2 */
clause:keywordClause{
    |LEFT_PARENTHESIS clause RIGHT_PARENTHESIS {
        $1 = " ( " + $2 + " ) ";};
    |clause AND clause {
        $1 = $1 + " and " + $3;};
/* rule 0-3 */
keywordClause:operand;

/* rule 0-4 */
operand:    term1
           |term2
           |integer
           |real;
/* rule 0-5 */
term1:    charString1;
/* rule 0-6 */
term2:    charString2;
/* rule 0-7 */
charString1: CHARSTRING1 {
    keyword1 = $1;
    $1 = " \t( \n";
    $1 += "\t\tlower(sea.title) like lower('" + keyword1 + "')\n";
    $1 += "\t\tor\tlower(sea.keywords) like lower('" + keyword1 + "')\n";
    $1 += "\t\tor\tlower(sea.description) like lower('" + keyword1 + "')\n";
    $1 += "\t)";};
```

```

    };
/* rule 0-8 */
charString2: CHARSTRING2 {
    keyword1 = $1;
    $1 = " \t( \n";
    $1 += "\t\tlower(sea.title) like lower(' + keyword1 + ')\n";
    $1 += "\t\tor\tlower(sea.keywords) like lower(' + keyword1 + ')\n";
    $1 += "\t\tor\tlower(sea.description) like lower(' + keyword1 + ')\n";
    $1 += "\t)";
};
/* rule 0-9 */
integer: INTEGER {
    int number = $1;
    $$ = " \t( \n";
    $$ += "\t\tlower(sea.title) like lower(' + number + ')\n";
    $$ += "\t\tor\tlower(sea.keywords) like lower(' + number + ')\n";
    $$ += "\t\tor\tlower(sea.description) like lower(' + number + ')\n";
    $$ += "\t)";
};
/* rule 0-10 */
real: REAL {
    double number = $1;
    $$ = " \t( \n";
    $$ += "\t\tlower(sea.title) like lower(' + number + ')\n";
    $$ += "\t\tor\tlower(sea.keywords) like lower(' + number + ')\n";
    $$ += "\t\tor\tlower(sea.description) like lower(' + number + ')\n";
    $$ += "\t)";
};
%%

```