

iCamp

innovative, inclusive, interactive
& intercultural learning campus

Prototype of the Repository Network

Deliverable Number:	D3.2
Contractual Date of Delivery:	06 / 2006
Actual Date of Delivery:	07 / 2006
Work Package(s) Contributing:	WP3
Nature of the Deliverable:	Prototype
Status:	Final
Security (Distribution Level):	Public
Editor(s):	Stefan Sobernig & Fridolin Wild, VUE
Project start: 1. Oct. 2005	Duration: 36 months STREP / IST

The iCamp Consortium consists of:

Centre for Social Innovation (CSI)	Coordinator	Austria
Jožef Stefan Institute (JSI)	Contractor	Slovenia
University of Leicester (ULE)	Contractor	United Kingdom
Universidad Politécnica de Madrid (UPM)	Contractor	Spain
Vienna University of Economics and Business Administration (VUE)	Contractor	Austria
University of Science and Technology (AGH)	Contractor	Poland
Kaunas University of Technology (KTU)	Contractor	Lithuania
Işik University (ISIK)	Contractor	Turkey
Tallinn University (TLU)	Contractor	Estonia
Tomas Bata University in Zlín (TBU)	Contractor	Czech Republic

Amendment History

Version	Date	Editor(s)	Modification
0.1	23.06.2006	Fridolin Wild	Outline
0.2	29.06.2006	Andrej Afonin	SQL Consumer Portlet with WSRP
0.3	02.07.2006	Andrej Afonin	SQL Consumer Portlet Revisited
0.4	11.07.2006	Fridolin Wild	Integrating descriptions of all SQL providers
0.5	11.7.2006	Fridolin Wild	Update Consumer Portlet, Integration of SQL Mediator Outlook
0.6	14.7.2006	Fridolin Wild	Update Consumer Portlet
0.7	14.7.2006	Stefan Sobernig	Update SQL Mediator, Integrate .LRN/Learn@WU Target, Minor Revisions
1.0	26.7.2006	Fridolin Wild, Stefan Sobernig	Changes According to Peer Review

Contributors

Name	Institution
Andrej Afonin	KTU
Anna Danielewska-Tulecka	AGH
Antonio Tapiador	UPM
Dariusz Górk	AGH
Fridolin Wild	VUE
Gürol Erdogan	ISIK
Jiří Vojtěšek	TBU
Milda Ridikaitė	KTU
Neophytos Demetriou	VUE
Petr Doležel	TBU
Radek Matušů	TBU
Sandra Aguirre	UPM
Stefan Sobernig	VUE
Tomáš Dulík	TBU
Vahur Rebas	TLU

List of Acronyms

Acronym	Description
AC	Accession Countries
AJAX	Asynchronous JavaScript and XML
API	Application program Interface. A set of calling conventions defining how a service is invoked through a software package.
CMS	Content Management System
CQL	Common Query Language
DHTML	Dynamic HTML
HTML	HyperText Markup Language
JRS	Java Specification Request
LMS	Learning Management System
LOM	Learning Object Meta-Data
NMS	New Member States
PLQL	ProLearn Query Language
SOAP	Simple Object Access Protocol. SOAP is a standard for exchanging XML-based messages over a computer network, normally using HTTP
SQI	Simple Query Interface
SQL	Structured Query Language
SRU	Search and Retrieve via URL
UI	User Interface
UML	Unified Modelling Language
VSQI	Very Simple Query Interface
WSRP	Web Service for Remote Portlets

Table of Contents

Executive summary	6
1. Prototype of Repository Network	7
1.1. The Integration Cases of the iCamp SQI Targets	8
1.1.1. Moodle @ AGH	8
1.1.2. Course Online (ISIK)	9
1.1.3. Drupal (UPM).....	10
1.1.4. EducaNext (UPM).....	11
1.1.5. IVA and Zope.....	13
1.1.6. ViPS (KTU).....	14
1.1.7. OAlster (TBU).....	15
1.1.8. .LRN / learn@WU (VUE)	17
1.2. Conclusion	18
2. Additional Services.....	19
2.1. An SQI Mediator (Preview)	19
2.2. An SQI Consumer Portlet (Preview).....	22
2.2.1. Introducing Portlets.....	23
2.2.2. Ajax-based Portlets	25
2.2.3. An SQI Consumer Portlet	27
2.2.4. Additional Proprietary Portlets	28
2.3. Conclusion	28
3. References.....	29

Executive summary

This prototype is targeted towards the task 3.2 of our work package, i.e. “to create an integrated resource repository network suitable for an Enlarged Europe” ([4]).

This accompanying document consists of four sections to be delivered in the context of this task.

First, it gives an overview on the network’s technological basis, a comprehensive description of the Simple Query Interface (SQI) and its presuppositions.

Second, the network nodes, so called SQI providers, which have been fitted with an interface to create interoperability, will be described to give an outline of how the actual solution works. In this deliverable we report the successful prototype implementation for five learning management systems, one learning object repository, one digital library, and – in addition – one content management system. We furthermore successfully developed through these integration cases the base SQI technology for several languages, including PHP, python, xOTcl, C, and Java. We consider the SQI interfaces of the widely used standard soft-ware systems .LRN, Moodle, as well as Drupal, and the interfaces for still considerably wide spread systems as IVA and EducaNext, as being a key achievement enabling the iCamp space to grow beyond the consortium partners.

Additionally and beyond the scope of the network prototype, two building blocks will be described here, which we consider as important in order to ease access to the network from an end-user perspective and to create the pre-conditions for deeper integration and enhanced service quality of the network.

This will be, third, an SQI mediator that encapsulates the functionality of a federated search module and exposes an SQI provider. This allows other SQI consumers to access all connected repositories by communicating with just one target.

Fourth, we investigate and prototypically provide an SQI consumer portlet to deliver network services to end-user environments. The portlet is to be deployed especially within institutional Learning Management Systems, but also beyond.

The prototype implementations, i.e. the prototype of the repository network plus the two additional services beyond the scope of this deliverable (i.e. the SQI mediator service and the SQI consumer portlet) as well as news about recent advances are available through the iCamp project website (accessible through the dedicated area available via <http://search.icamp.eu>).

1. Prototype of Repository Network

One of the aims of iCamp is to provide students as well as the academic staff of European Higher Education Institutions with access to distributed content repositories.

By providing interoperability between learning object repositories or – more precisely – all repositories that contain (preferably digital) artefacts relevant for learning, iCamp enables the users of iCamp compatible learning tools to extend their access beyond only locally available material to the variety of learning objects provided in different systems and places. This distributed network of content repositories is based on the Simple Query Interface (SQI, see [17]) and enhances the already existing network built in the context of several European projects, most notably ProLearn, to a level covering an enlarged Europe by including several of the key institutions in the NMS and AC countries, while still improving coverage in the already contributing member states.

SQI is an interoperability infrastructure that enables heterogeneous systems to communicate with each other for the purpose of distributed learning object retrieval. Typically, this communication is based on web services (using SOAP messaging via HTTP as a network protocol). To be put into practice in the joint network of learning object repositories, SQI relies furthermore on a common schema, defining common semantics, and a common data model as well as a format, specifying which common query language is to be used and how results should be encoded and represented (see Figure 1.1, cf. [10]).

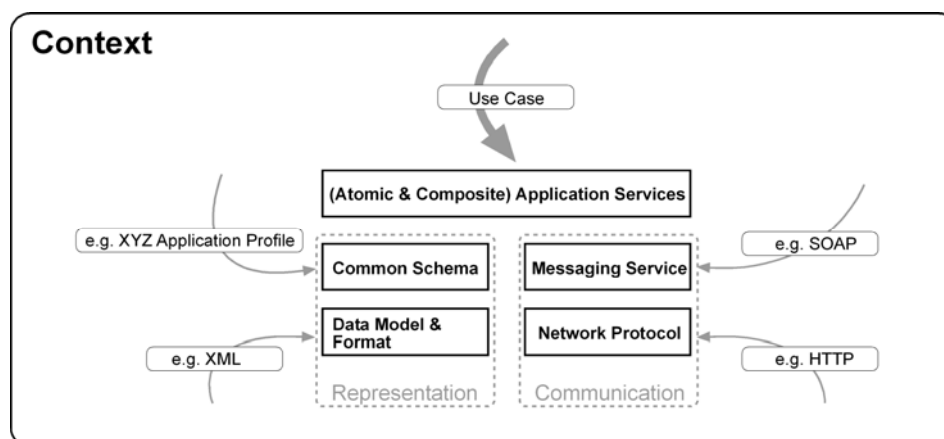


Figure 1.1. SQI Layers.

Figure 1.2 gives an overview over the data exchange process between two communicating repositories through their SQI interfaces. Through this abstraction of the data exchange process between different learning systems and tools, the prototype of a networked learning object repository was realised which we are reporting here.

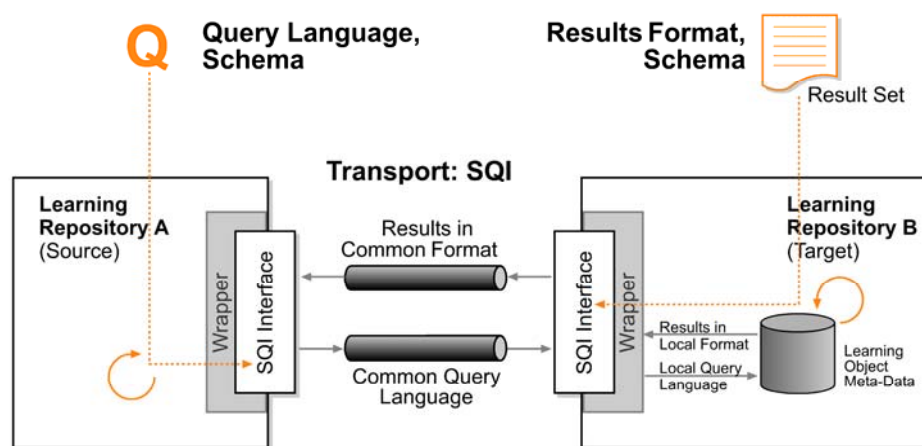


Figure 1.2. SQL Communication Process.

For this prototype network the query language was specified to be keyword bags. As WP4 within the ProLearn Network of Excellence is developing a specification for a joint ProLearn Query Language (PLQL) which has not been finished yet, we intend to reconsider the query language issue to a later point during the project run time. The common schema and accordingly the common results format was specified to be RSS 2.0. In several cases, the SQL targets also support VSQI queries and additional results formats. As soon as the work within ProLearn has been finished we will reconsider the issue of schemata, query languages, and results formats.

The following eight integration cases will be covered by this deliverable: moodle (realised with the AGH instance), Course Online, Drupal (realised with the UPM instance), EducaNext, IVA / Zope, ViPS, OAlster, and .LRN (realised with learn@WU).

1.1. The Integration Cases of the iCamp SQL Targets

The following section contains information about the various target repositories the iCamp partners implemented SQL interfaces for.

1.1.1. Moodle @ AGH

The Distance Education Study Centre (DESC) is a unit of the AGH University of Science and Technology cooperating with all faculties of the university. The e-learning platform used by DESC is based on Moodle.

To connect Moodle to the repository network, the SQL wrapper sends queries to the Moodle database and returns corresponding results. By now, this wrapper accepts simple keywords or VSQI multi-word queries and transforms them to SQL, as Moodle is using MySQL. Results can be returned in a proprietary XML or in RSS 2.0 format (see Figure 1.3).

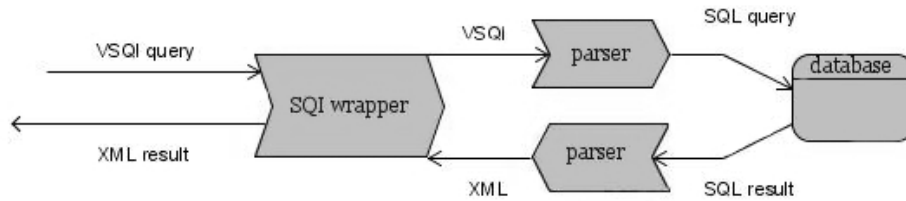


Figure 1.3. Communication Process within the Moodle SQL Interface.

Parsing is performed by classes inheriting from QueryParser (see Figure 1.4), the appropriate QueryParser is built according to SQLI wrapper setting (TargetSoapBindingImpl::setQueryLanguage). The architecture is based on Factory and Builder design patterns. The class QueryParserBuilder is responsible for providing appropriate parsers. The SQLQueryParser has methods performing the connection to database and queries the Moodle database.

For database communication we use adodb.inc.php library which is flexible and can be used with different types of databases (for example mySQL and postgresql).

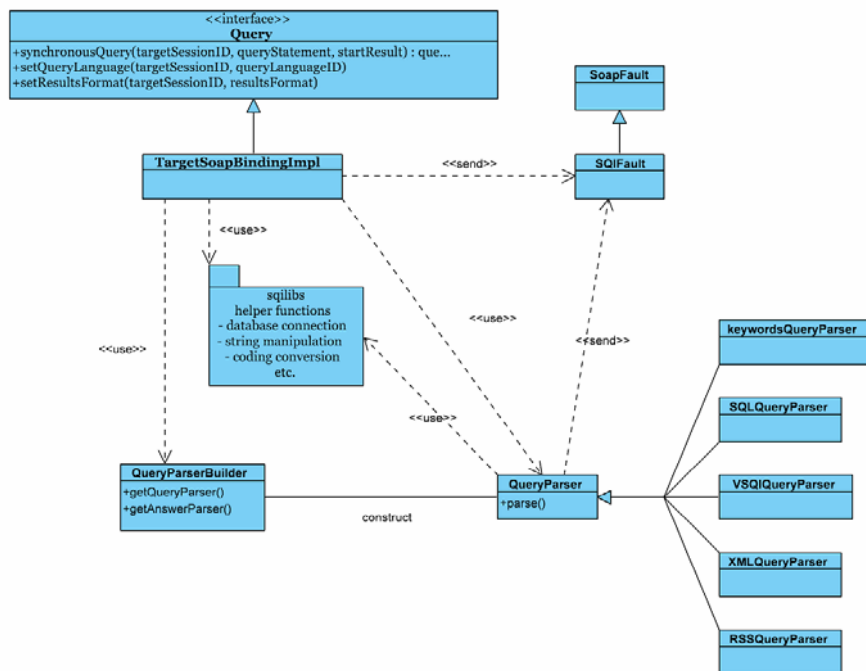


Figure 1.4. System Architecture of the Moodle SQL Interface.

1.1.2. Course Online (ISIK)

Course ON-LINE is central course homepages management tool. Course ON-LINE supplies centralized management of all course homepages of a school and enables a uniform, easy to learn and use structure for each course

homepage. Course ON-LINE uses the common database of Campus ON-LINE (the course registration and student information system), for initially setting up of course homepages automatically as the new semester begins and generates a course syllabus for each course. Instructors can add custom information to the syllabus (i.e. course outline, course objectives, textbooks and references etc.), define grading scheme of the course and assign teaching assistants to the course. Instructors can give assignments, publish course materials, make announcements, add web based tools (i.e. web based simulations or animations related to the course), supply links to different web resources, announce grades of assignments, projects or exams, or communicate with the students using forums or e-mail facilities.

Students can view course syllabus, view assignments and upload assignment files, reach to instructor’s course materials, reach to web resources related to the course supplied by the instructor, view course announcements, grades of homeworks, projects or exams, view and participate to the forums and communicate with their instructors using e-mail.

Course ON-LINE now supports SQL querying in the Course ON-LINE file repository. The following diagram of Figure 1.5 shows the general architecture of the SQL Target support.

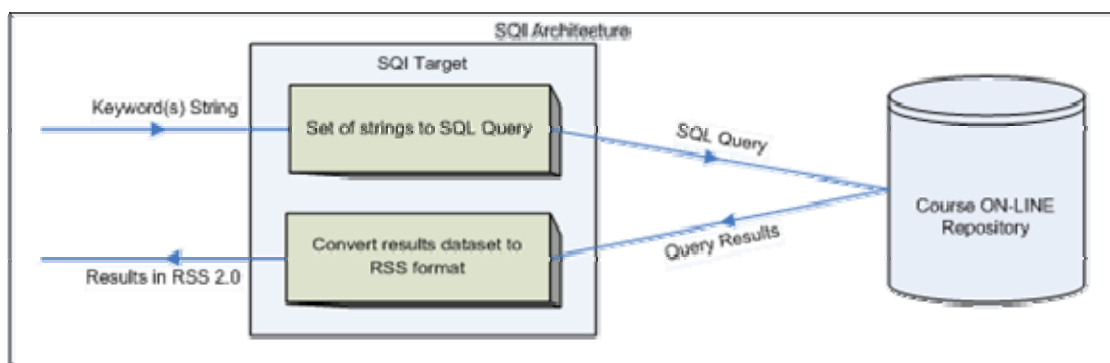


Figure 1.5. Course Online Architecture for the SQL Interface

The diagram represents the general architecture of SQL support in Course ON-LINE. Steps of querying process are as follows:

The keywords string is split by the white spaces and a set of keywords is produced. Using this keyword set, SQL queries that are searching in the content tables of Course ON-LINE are produced. The queries are sent to the Course ON-LINE database. Returning results in dataset form are aggregated into one RSS formatted text. Finally, RSS results are sent as output.

1.1.3. Drupal (UPM)

Drupal (<http://drupal.org/>) is a popular content management system (CMS) written in PHP. It is free software, distributed under the GNU GPL license. Drupal provides extension facilities through the modules method. Drupal

modules are PHP files stored in a specific directory and implementing some of the Drupal modules hooks.

The SQI implementation provides the sqi.module file which implements a hook that maps certain URLs – e.g.

`http://www.example.com/drupal/?q=SQISessionManager`

– to the SQI class calls.

Drupal organizes and stores the content as so called ‘nodes’. Every node has its own entry in the database. Drupal supports MySQL and PostgreSQL database systems, and provides mechanisms for extending it to other database systems, through a database abstraction layer. It also provides search facilities through the Drupal API. SQI implementation uses the `do_search` function (see [18]) in order to perform the query, and abstracting in this way from the database model Drupal is installed in.

Currently, Session Management is implemented in Anonymous access. Target class performs searches based on SQI query keywords, results are returned in RSS format.

1.1.4. EducaNext (UPM)

EducaNext (<http://www.educanext.org>) is a multilingual, academic exchange portal, where members of higher education, research organisations, and professional communities can go to share, retrieve, and reuse learning resources. EducaNext fosters collaboration among educators and researchers, allowing its users to participate in Knowledge Communities, to communicate with experts in your field, and to exchange Learning objects. Furthermore, it facilitates working together on the production of Educational Material: Textbooks, lecture notes, case studies, simulations, etc. EducaNext helps to deliver distributed Educational Activities, among others lectures, courses, workshops, and case study discussions. EducaNext supports its users in distributing electronic content under license.

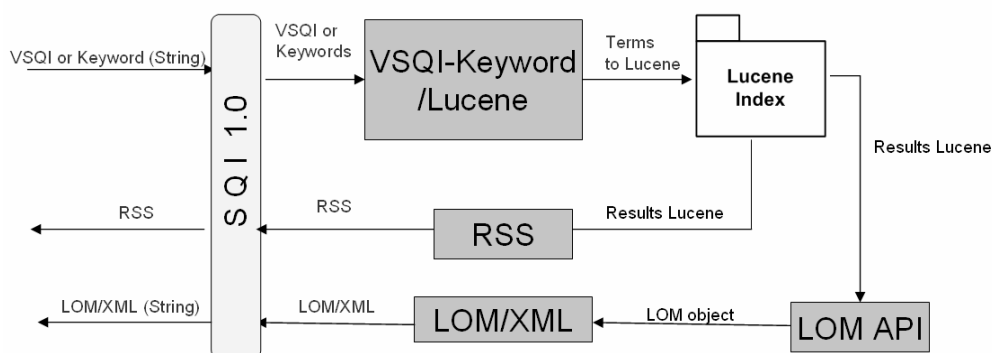


Figure 1.7. EducaNext SQI Implementation (Overview).

Figure 1.7 represents the general architecture that supports SQI. When a query (VSQI or keyword query language) is received by the SQI interface, the process that takes place in order to achieve results is as follows: First of all the queries are parsed to obtain the search terms. Those will later-on be sent to the Lucene engine.

Once the Lucene has obtained results, we will use the LOM API to build the LOM object if the result format is LOM. Finally, we develop a parsing process from the LOM object to LOM XML.

```
<results>
<lom>
<general>
<identifier>
<catalog>EducaNext</catalog>
<entry>lr-wuw-wild-1110818181664</entry>
</identifier>
<title>
<string>SQI / HCD-Suite: Building a Network
of Educational Resource Repositories</string>
</title>
<language>en</language>
<description>
<string>Structure of the Talk: - A Vision
of Interoperability, - Interoperability
Framework, - Status Quo of the evolving
Educational Repository Network, - Integration
Patterns, - Future Issues, - SQI Developers
& Implementers Community</string>
</description>
</general>
<technical>
<location>http://www.educanext.org/ubp/search@srchDetails
LR?lrID=lr-wuw-wild-1110818181664</location>
</technical>
</lom>
</results>
```

If the result format is RSS (Real Simply Syndication), we will need a parsing process from Results Lucene to RSS format.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
<channel>
<title>EducaNext Results</title>
<link>http://www.educanext.org</link>
<description>EducaNext Learning Resources</description>
<item>
<title>SQI / HCD-Suite: Building a Network
of Educational Resource Repositories</title>
<link>http://www.educanext.org/ubp/search@srchDetailsLR?lrID=lr
-wuw-wild-1110818181664</link>
<description>Structure of the Talk: - A
Vision of Interoperability, - Interoperability
Framework, - Status Quo of the evolving Educational
Repository Network, - Integration Patterns, -
Future Issues, - SQI Developers & Implementers
Community</description>
<pubDate>dom, 13 mar 2005 23:00:00 GMT</pubDate>
<guid>http://www.educanext.org/ubp/search@srchDetailsLR?lrID=lr
-wuw-wild-1110818181664</guid>
</item>
```

```
</channel>
</rss>
```

The interface implementation has been contributed back to the EducaNext SVN, so it can be re-used in the instances in Vienna and Tallinn.

1.1.5. IVA and Zope

Zope is an open-source content management system written in Python. It offers a transactional object database to store content, but is able to work with dynamic templates, scripts, a search engine, and more.

IVA is a web-based learning management system, which is developed in Tallinn University in order to advocate constructivist approaches and practices in e-learning.

IVA is built on Zope. Authentication, user and rights management is done with the built-in functionality of Zope. Data processing is mostly done with Python scripts and methods, while web pages are mostly built with Zope Page Templates (ZPT). Data is stored in the Zope Object Database (ZODB).

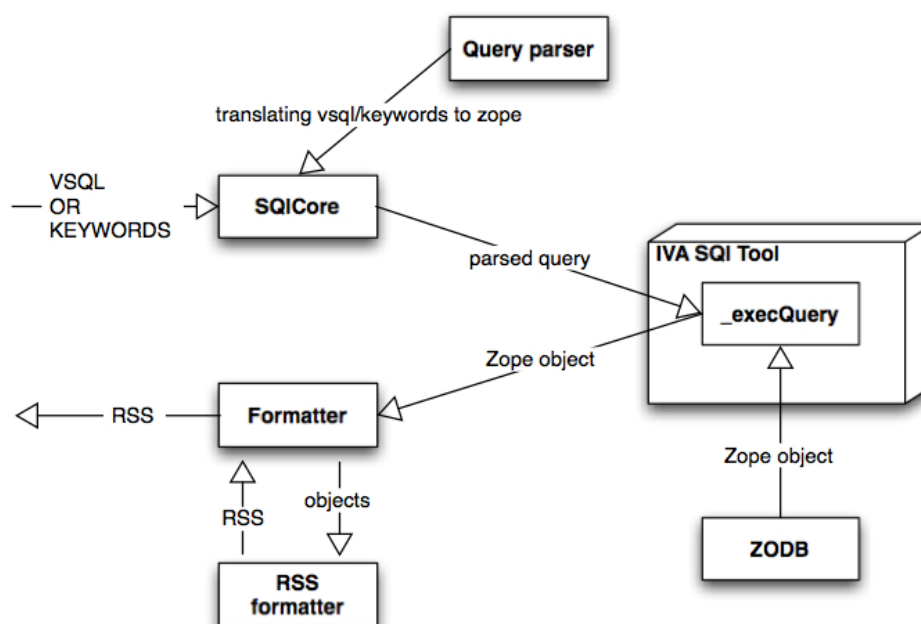


Figure 1.8. SQL Integration in IVA/Zope.

IVA Simple Query Interface is divided into two packages: SQICore and SQL-Tool. SQICore is a general implementation of SQL as Zope package (see Figure 1.8). Once the query arrives, the parsers will create a new query that is understandable to Zope.

Currently SQICore implementation knows VSQL and keywords search. As the query is parsed and translated to Zope it is then handed over to the IVA-SQITool in IVA. IVA will execute this query and returns Zope objects. These

are then passed to the formatter. Currently there is only one formatter – RSS. SQI session management implementation handles anonymous and authenticated sessions

1.1.6. ViPS (KTU)

VIPS (abbreviated for ‘Video Paskaitų Sistema’ = Video Lectures System) is a system which is designed for broadcasting lectures and papers over the internet with support for recording and reviewing. During the real-time broadcasting, VIPS allows to transfer lecturers’ and speakers’ video, audio, and slide-shows. It also allows to arrange interactive communication between lecture participants (through, e.g., questions & answers, polls, voting features). VIPS is a pure web application and needs only a standard web browser such as Internet Explorer without any additional software.

The basic idea of the integration of the VIPS repository to the repository network is to allow user from distance locations get information about video lectures that are stored in VIPS. For integrating VIPS to the iCamp network of repositories, we have to develop an SQI interface that will work as middleware between VIPS and any SQI consumer – a piece of software that is responsible for processing queries from clients and deflection them to the storage system.

The SQI interface for VIPS has the following basic architecture as shown in Figure 1.9.

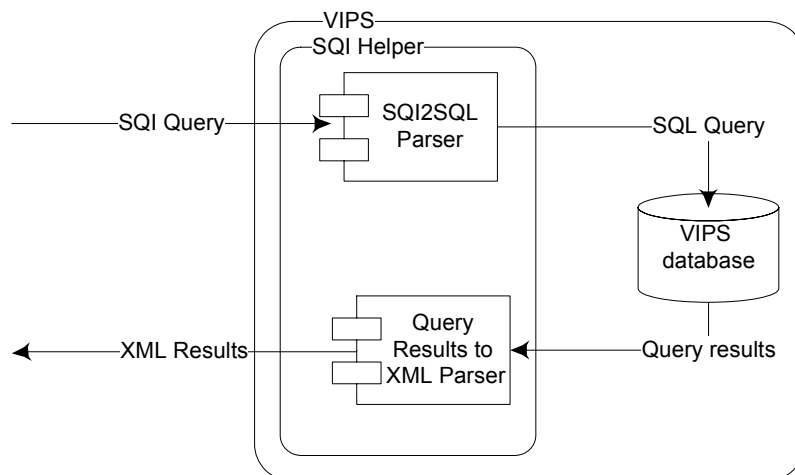


Figure 1.9. SQI Interface in VIPS (basic architecture).

Figure 1.10 provides overview through a detailed step-by-step communication architecture sketch. SQI Helper is the SQI interface, which is included as a module in VIPS. SQI Helper is responsible for getting an incoming query from an SQI consumer. SQI Helper passes this query to the SQI2SQL module responsible for transforming the query to SQL. After that, SQI Helper contacts the VIPS database and fetches the search results. The module Query Result to XML Parser gets data that was picked from the database and transforms it to XML document.

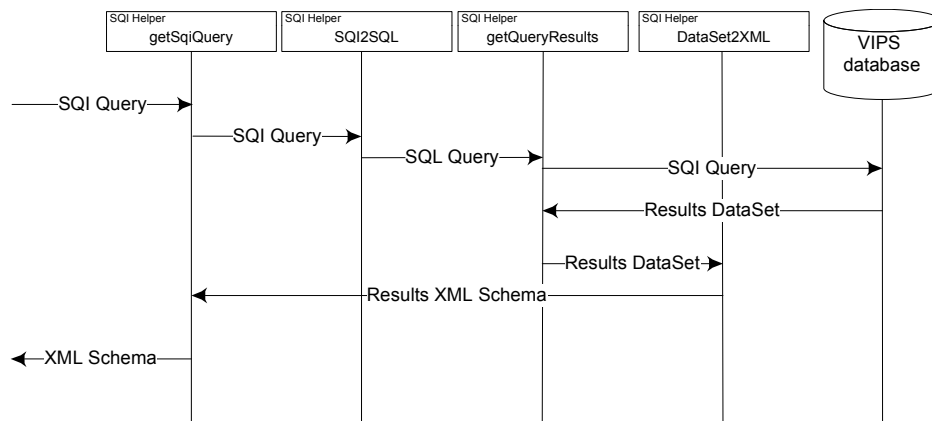


Figure 1.10. Step-by-Step Architecture.

In VIPS, meta-data of each course is stored in the database in the table 'meta_info'. Query results are selected according to the data that is stored in 'meta_info'. The relationship between course data and meta-data is shown in Figure 1.11.

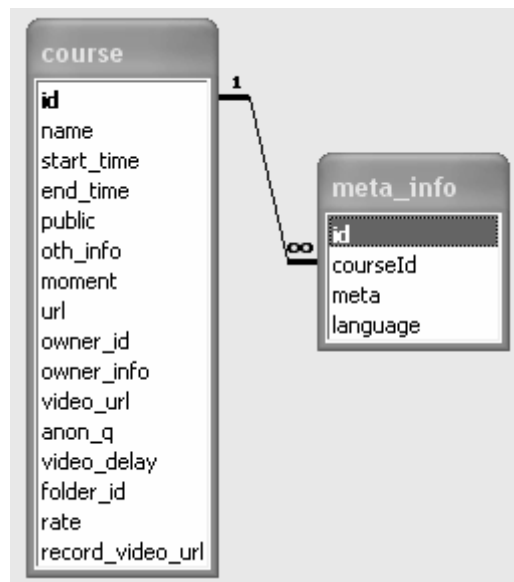


Figure 1.11. Relationship between the Tables 'course' and 'meta_info'.

At this moment meta-data for each course have to be generated manually by each course's author.

1.1.7. OAlster (TBU)

OAlster is a project of the University of Michigan Digital Library Production Service. The goal of this project is to create a collection of previously difficult-to-access, academically-oriented digital resources that are easily searchable by anyone. This archive contains more than 8 million records from various areas (see [5]).

The OAIster project already offers an SRU search interface (see [5]). Because SRU supports standard XML formats, we decided to choose SRU rather than the other interface (openURL) offered. The OAIster SRU search interface allows to search only with CQL queries level 0 (see [19]). According to CQL specification the level 0 allows term-only query. The term is either a single word or if multiple words separated by spaces then the entire search term is quoted ([6]).

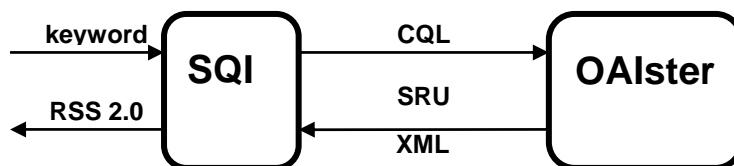


Figure 1.12. Communication Process of the OAIster-SRU Gateway.

OAIster uses CQL for querying and sends the XML document via SRU protocol (cf. Figure 1.12). The synchronous query in SQI transforms the ‘keywords’ query language to CQL. After data is retrieved, the RSS output (cf. [7]) is generated. The RSS contains only title, link and description of the record.

The SQI provides these functions:

- setQueryLanguage ()
- setResultsFormat ()
- setMaxQueryResults ()
- setResultsSetSize ()
- synchronousQuery ()
- getTotalResultsCount ()

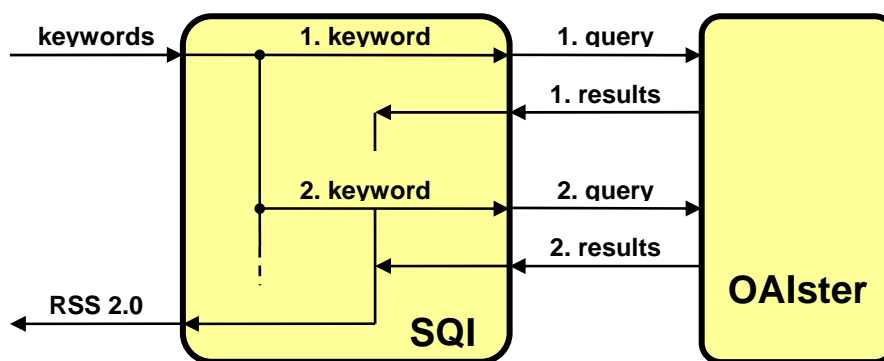


Figure 1.13. System Architecture of the OAIster Gateway.

We used multi synchronous queries for simple multi keyword search (cf. Figure 1.13). This method has three limitations: The total results count can't be detected. Furthermore, the multi-keyword search uses only the AND operator for the keywords. Third, the search for each separate keyword is limited according to constant maxMultiSearchResults.

The implementation contains also simple session management and other supporting functions:

- nusoap.php (standard PHP SOAP library, see [8])
- sessionmgt.php (session management)
- sqitarget.php (SQI and other functions)

1.1.8. .LRN / learn@WU (VUE)

Learn@WU is a large-scale learning management system deployed institution-wide at the Vienna University of Economics and Business Administration, supported by a content provision program devised and enforced at the entire university. Currently, it serves more than 13.000 active users (at least one log-in per week) with more than 9.000 learning objects made available to them.

Learn@WU is conceptualised and realised by extending the community framework provided by .LRN (see [16]) and OpenACS (see [15]), two widely adopted open-source projects.

The SQI target for learn@WU, more precisely the content repository system provided by OpenACS and .LRN, is realised by adopting the wrapper pattern (as described by [12] and [13]). An architectural scheme is given by Figure 1.14 below.

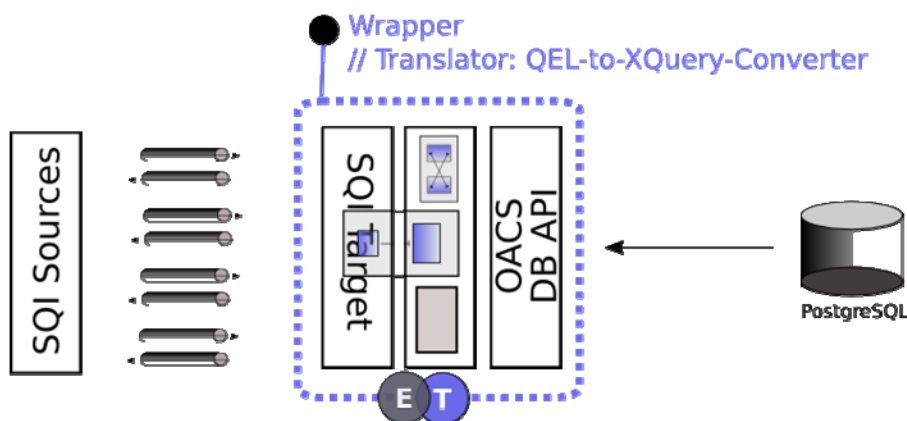


Figure 1.14. Scheme of Learn@WU target.

As can be seen above, the implementation comprises three elements: First, an extender that represents a SQI SOAP stub that is realised by the SOAP server and client framework xosoap that was developed in this context and is about to be released as re-usable service package to the OpenACS and .LRN communities. Keyword-bag queries and resulting requests from the Content Repository are processed by a translator component that reflects keyword bags and their implicit conjunction in a subset of SQL statements that are issued against the data base backend by means of the existing OpenACS data base API. Result sets are transformed into a RSS format conforming to the negotiated meta-data schema and returned by the extender, i.e. the SQI stub. The implementation and its translator can be easily adapted to a variety of query languages to be handled, including semi-structured query languages.

1.2. Conclusion

In the technical annex of the iCamp contract we specified the evaluation criteria for measuring the success of our overall interoperability activities. We quantify this success rate by evaluating, how many open source systems can successfully be integrated into an emerging iCamp space (for more information about the iCamp space see D2.1).

For the prototype of the network fulfilling task 3.2 of our work package, we break this down – as specified – to encompass at least two digital libraries / learning object repositories, and at least two learning management systems.

In this deliverable we report the successful prototype implementation for five learning management systems, one learning object repository, one digital library, and – in addition – one content management system.

We furthermore successfully developed through these integration cases the base SQL technology for several languages, including PHP, python, xotcl, C, and Java. We consider the SQL interfaces of the widely used standard software systems .LRN, Moodle, as well as Drupal, and the interfaces for still considerably wide spread systems as IVA and EducaNext, as being a key achievement enabling the iCamp space to grow beyond the consortium partners.

We consider the OAlster interface as the first big digital library connection and currently plan to integrate more digital libraries during the project run-time. Additionally, this implementation realizes a gateway to SRU bearing the potential to quickly connect other SRU compliant nodes in the future.

We intend to wait for developments in the context of the ProLearn Query Language (PLQL) to evaluate its relevance for iCamp and we are already involved in the discussion process through our collaboration set-up within the PRO-LC cluster.

The implementations for the particular systems specified within this deliverable have been tested and the code has been stored to a common subversion repository. Till the first trials the prototypes, i.e. the prototype of the repository network, will be turned from production state to a live version accessible by students and facilitators in the participating institutions.

2. Additional Services

In addition to the prototype of the repository network, the consortium members identified two additional services that can facilitate easy access for end-users to the network, mediated through the user interface of a given learning tool. Basically, these supporting services can be differentiated into two types, the separation being inevitably inherent in the nature of services targeted to and provided for end-users.

The first service, the SQL mediator, encapsulates the functionality of a federated search and exposes itself again as an SQL target. The mediator distributes queries through the repository network, collects hits, and returns a ranked result-set to the requesting SQL consumer.

The second service, the SQL consumer portlet, complements the mediator by providing a highly re-usable web-based search client that easily can be integrated in a wide range of existing learning tools in the iCamp tools portfolio.

As ultimate re-use of the consumer portlet is neither possible nor desired, it makes sense in our point of view to separate the middle-ware logic (i.e. the mediator) from the presentation logic (i.e. the consumer portlet).

In the next section (2.1) we provide some architectural considerations on such a mediator service, including a fundamental functional scheme. In Section 2.2, we enter into a discussion on how to realize the aforementioned consumer portlet in the context of iCamp. In doing so, we discriminate between two approaches, Web Service Remote Portlets (WSRP) and Ajax-based portlets, that are not strictly mutually exclusive but can rather be considered complementary. Nevertheless, as will be outlined in detail in 2.2, they focus on different layers, the former on the application and the latter on the presentation layer.

2.1. An SQL Mediator (Preview)

As prominently pointed out in [10] and [11], mounting and deploying a SQL-based network requires several steps. These involve the deployment of a distributed wrapper or proxy architecture consisting of SQL Targets (see 1.1) and a previously negotiated meta-data schema realized by this architecture. The latter step is completed by the deployment of global network supporting facilities, such as annotation services or, as in the concrete case of the iCamp repository network, a query mediating service, the SQL mediator. These examples of global facilities help to overcome shortcomings of pure wrapper or proxy architectures and represent integration strategies that shift responsibility from implementing institutions to the repository network as such. Limitations identified so far and addressed by global facilities such as the SQL mediator are reduced complexity of the entire integration architecture based upon query translation mechanisms and centralized pre- and post-processing of requests, queries, and result sets (see especially [10] and [11] for details).

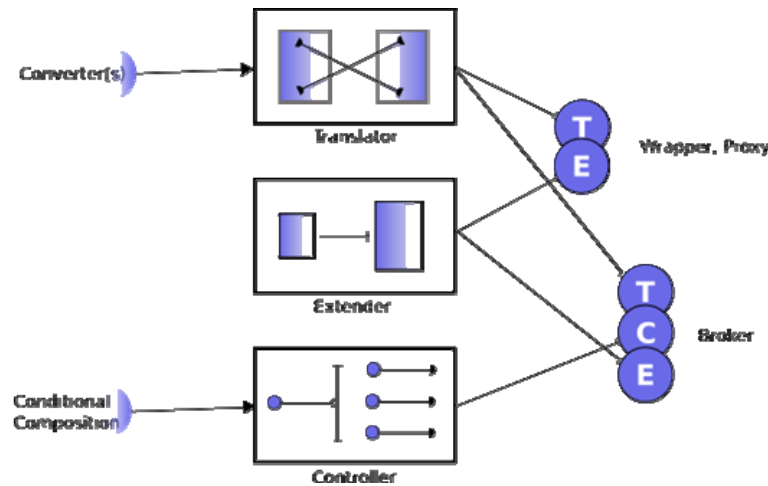


Figure 2.1. Broker Concept: Integration Strategies, Architectural Styles, and Patterns.

The SQL mediator formally takes the architectural role of a ‘broker’, as thoroughly described in the literature on design and architectural patterns, especially in remoting and distributed system environments (see e.g. [12] and [13]). A broker, generally speaking, comprises three functional spheres: a translator, an extender, and a controller facility (see Figure 2.1).

Regarding the SQL mediator, the broker serves as a translator in the sense of a federated search vehicle. As depicted in Figure 2.2, the primary objective is to collect queries issued by an SQL source and translate them into a request understood by potential SQL targets and deliver this processed and translated query to the target systems registered at the SQL mediator. In the result flow, result sets are collected, transformed into a canonical representation format, processed by aggregation facilities, and delivered back to the issuing SQL source system.

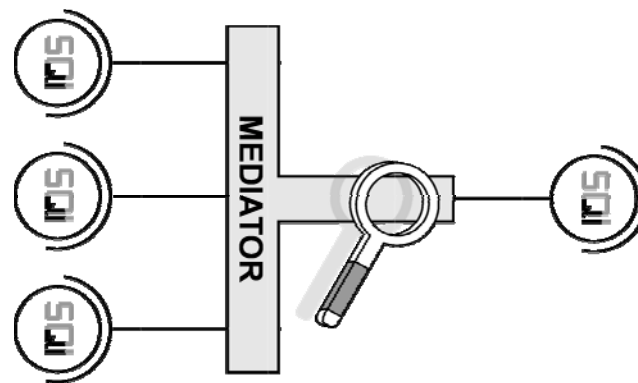


Figure 2.2. Working Principle of the Mediator.

Therefore – considering the identified architectural styles realised in the SQL mediator –, its translator facility takes care of query translation and result set post-processing in the sense described above. The mediator’s controller is represented by a set of functionalities with a registry of repositories and repository descriptions at their very core. The registry serves both as configura-

tion service for SQL source systems that do not rely on the extended mediating facilities provided, and is the primary source for routing and dispatching mediated queries, as well as performing transformations and processing results. The mediator's extender is partially realised by extension features that come both with the translator and the controller. The mediator, however, also provides facilities for caching of queries and results in order to ascertain performance and scalability. A comprehensive functional scheme is provided in Figure 2.3 below in terms of an UML sequence diagram.

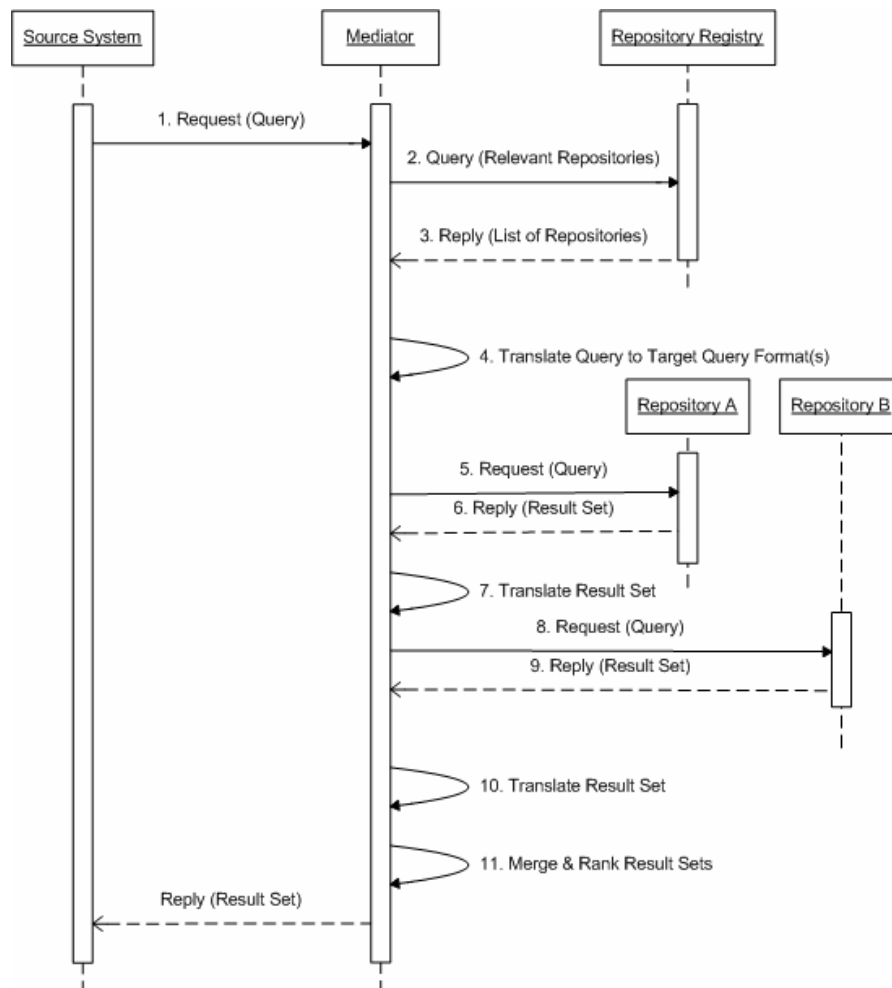


Figure 2.3. Sequence Diagram for the Mediator.

A macro-perspective on the combined wrapper/proxy-mediator architecture reveals that SQL-based repository networks follow the information-querying paradigm as primary mode of interaction, in contrast to information filtering (see D3.1, Sections 5.1.4 and 5.1.5 for details, i.e. [14]).

Global network facilities such as the SQL mediator could pave the ground for an integration of these two modes of interaction within a single network. This might allow to combine a publish-subscribe and a conventional retrieval architecture. At the same time it allows for adding further features such as re-use of cached queries for filtering items in a publish-subscribe setting or such as

using filtered content items for providing additional annotations to resources retrieved. This promising idea will be further investigated within the next advances of the iCamp repository network.

The SQL mediator has prototypically been implemented and tested, thereby demonstrating the feasibility of the functionality depicted in Figure 2.3.

2.2. An SQL Consumer Portlet (Preview)

A SQL consumer portlet is a visible by end-user web component that processes requests to another web interface, handles responses and renders response results. Portlets are used in portals as pluggable user-interface components that represent data in information systems. Section 2.2.1 introduces and reviews the portlet concept in greater detail.

A SQL consumer portlet is a search interface, placed in a virtual learning environment that communicates with the SQL mediator. The SQL consumer portlet could be defined as a search interface to retrieve learning objects in the federated network of repositories.

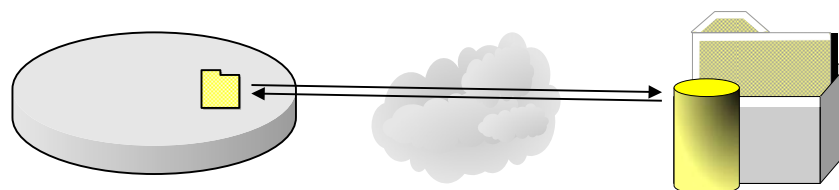


Figure 2.4. Communication between SQL Consumer and Mediator.

In serving this purpose, it sends requests to the SQL mediator and shows interaction results in its window after handling responses from the mediator.

In the following section, the Web Service Remote Portlet (WSRP) standard will be briefly evaluated and its applicability within the iCamp space will be assessed. Moreover, the availability of programming language support and appropriate development environments deployed in the iCamp tools is analysed exemplarily for PHP. Traditionally, portlets reference implementations exist for Java, a fact that introduces entry barriers when developing in and for environments different to Java.

We have come to the conclusion that support for WSRP is still lacking coverage for the programming languages and development frameworks we deal with. Therefore, we opted for the alternative to deploy an Ajax-based portlet. We will reconsider WSRP for future enhancements and work, in continuation of our Ajax-based artefacts.

2.2.1. Introducing Portlets

A portal is a web application that aggregates content from different web sources and presents it ergonomically sound to the end-user. According to IBM (see [1]), a “portal is a Web-based application that is customizable by end-users both in the look and feel of the portal and in available content and applications which the portal contains. A portal, furthermore, can be thought of as an aggregator of content and applications or a single point of entry to a user's set of tools and applications.”

A portal consists of portlets, which – again according to a definition given by IBM in [1] – “can be thought of as a miniature Web application that is running inside of a portal page along side any number of similar entities”. Furthermore, this document states, that “a port-let is a Web component which is managed by a container and can process requests and generate dynamic content”. Figure 2.5 depicts a basic portal model.

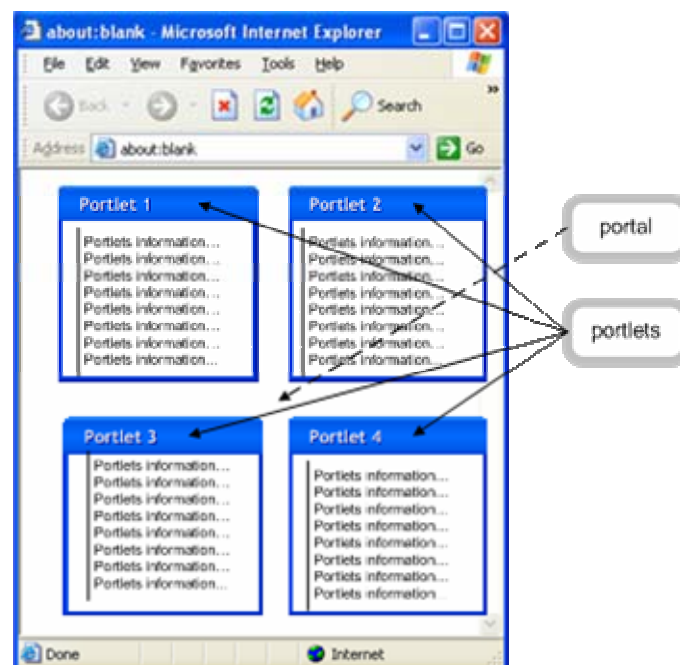


Figure 2.5. Portal Page.

The web application invokes a portlet and displays it on the current page a user is visiting. The portlet located in the webpage calls a portlet container via the Portlet Invoker API and retrieves information. The Portlet Provider SPI is user by the container to retrieve information about the portal. This is schematically presented in Figure 2.6.

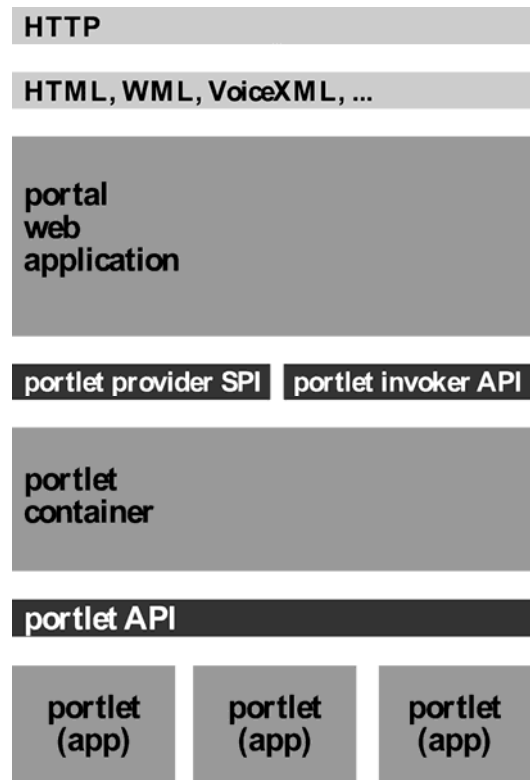


Figure 2.6. Portal Architecture
(redrawn from [2])

2.2.2. Portlet Remoting: JSR 168 and WRSP

The goal of JSR (Java Specification Request) 168, the Portlet Specification, is to enable interoperability between portlets and portals (see [2]) in Java-based environments. The WRSP (Web Service for Remote Portlets) was fundamentally inspired by JSR 168 and extends its ideas to portlet interoperability beyond platform boundaries. The WRSP specification draws upon existing Web Service currents and conceptualizes platform-independent portlet remoting.

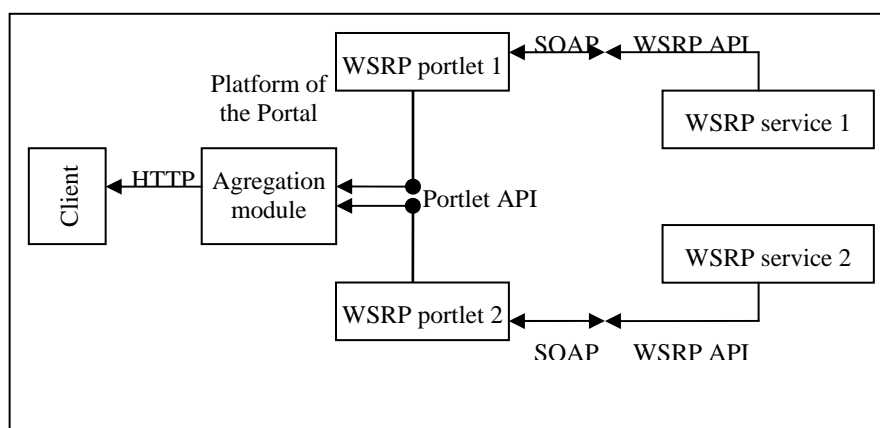


Figure 2.7. Integration according WSRP

The basic integration scheme according to the WSRP specification is shown in Figure 2.7.

2.2.2. Ajax-based Portlets

Ajax, shorthand for Asynchronous Javascript and XML, is a web-based technique for creating interactive web application (see [3]).

The main idea of AJAX technique is that a web page does not have to be re-loaded every time responding to the user change action. A component on a page is interacting with an Ajax producer via XML and changing the client's state on getting XMLHttpRequest object.

2.2.4. Scenarios considered: WSRP vs. Ajax

The two approaches to the portlet concept briefly outlined above operate at different layers considering application architecture. WSRP is a specific form of middleware that targets exchange and delivery of remotely provided application functionality, while Ajax with its dynamic HTML (DHTML) background operates at the presentation level of web applications. Against their different backgrounds, they serve slightly different scenarios, but are still compatible and thus can be considered complementary.

There are two different levels of interoperability which offer to two different architectural scenarios in implementing interoperability for the functionalities that are planned to be delivered within the iCamp space. The first scenario is to implement interoperability in the middleware where the learning systems' interactions are mediated and where systems integrate remote functionality delivered by this middleware based on their own user interfaces (UI) (see Figure 2.8).

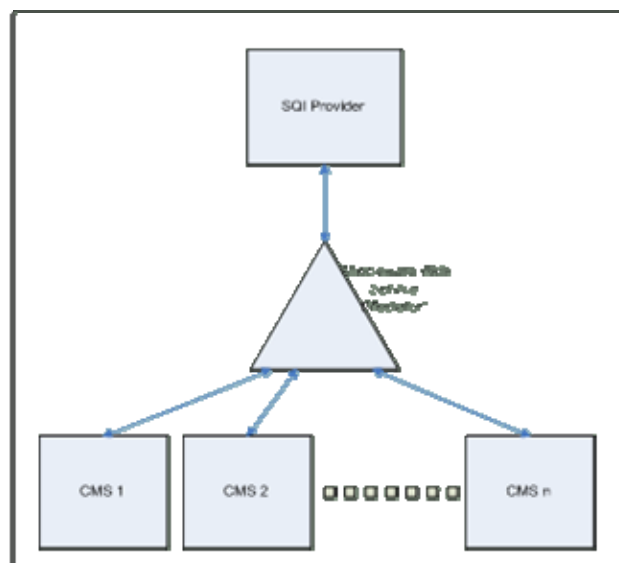


Figure 2.8. Middleware-based Interoperability

In the basic scenario, the iCamp Space would provide mediating services as described in Section 2.1 and produce add-on interfaces for each and every system in use of the iCamp community (e.g. Plone, Moodle, or Course Online). This would require new learning systems intended to be integrated into the iCamp Space either to use the suite of UI artefacts or to implement an own UI which accesses the iCamp middleware, e.g. the SQL mediator.

WSRP would extend or facilitate the basic scenario in terms of providing a platform and learning-system independent portlet rendering service. Ajax-based UI artefacts aim at increased re-usability of generic UI nuggets that can be integrated into any browser-based learning system (and beyond).

Against this background, the deployment of a WSRP infrastructure requires several steps: Front-end systems consume a WSRP-compliant portlet with a WSRP consumer framework for each and every learning system or their underlying platforms. One producer implementation, which resides on a unique portlet provider framework, serves the front-end systems with mark-up fragments as well as with the portlet application logic – and integrates the SQL mediator functionality.

This configuration (see Figure 2.9) enables primarily interoperability at the application layer. As a first step, the main aim is, however, to provide re-usable user interfaces. Furthermore, this architecture also brings the need to have a consumer development framework for any infrastructure: if a new front-end system would like to use those portlets, they will need to find a way to consume the portlets, either by using the portlet consumer instances developed within existing systems participating in the iCamp Space, or by developing own, new WSRP consumers.

There is the possibility that a new partner might not be using an infrastructure that provides a consumer development framework. This bears the threat that the iCamp services might not be re-usable in some infrastructures, especially in view of the lacking WSRP support for many platforms or language environments.

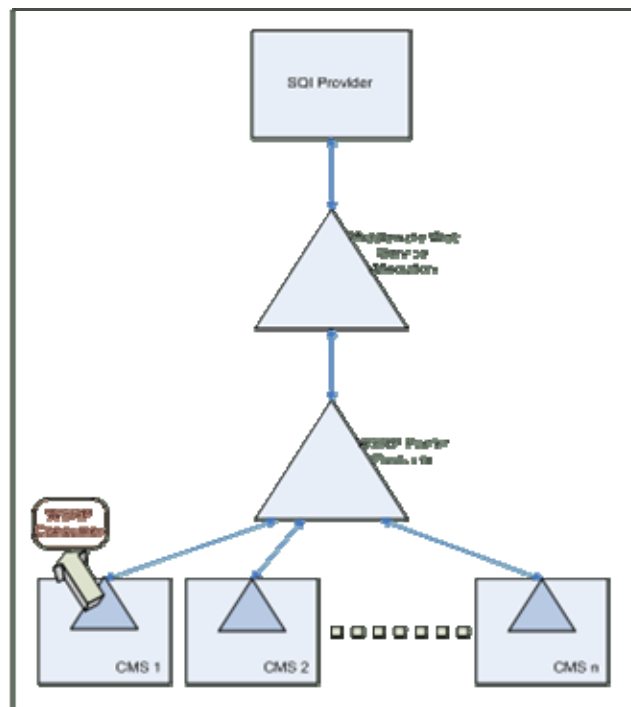


Figure 2.9. WSRP-based Infrastructure.

Current status of availability of WSRP consumer building frameworks in different infrastructures have been investigated and the following results could be obtained: For Java, there exists a producer and consumer implementation under Java Community Process (JSR 168) and a pilot project is being developed within the Apache Portals Project for delivering WSRP support to Java environments, i.e. WSRP4J (see portals.apache.org/wsrp4j/).

For Microsoft .NET, there is a commercially available framework for building WSRP compliant portlet producers (www.netunity.com) and there is a smart client application example including the source codes of the API, which implements almost all aspects of WSRP specification (see [20]). For PHP, we could not identify any efforts in building a consumer or provider implementation for WSRP.

Investigation of the existing frameworks shows that WSRP standard is not sufficiently supported, which leads to our conclusion to tackle a WSRP infrastructure as a continuing step.

2.2.3. An SQL Consumer Portlet

Figure 2.10 shows the basic communication architecture of the SQL consumer portlet.

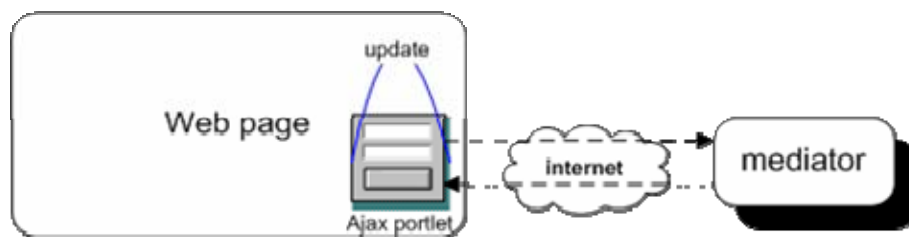


Figure 2.10. Ajax Portlet Architecture

The basic interaction scheme, as depicted above, involves an Ajax portlet that encapsulates a search form. Requests issued by the portlet are directed to the SQL mediator (see Section 2.1). The SQL mediator acts as broker, and in doing so, multiplexes the requests to the registered repositories, collects and aggregates result sets, and returns a consolidated result set to the Ajax portlet that takes care of rendering results.

The Ajax-based SQL consumer portlet has prototypically been implemented and tested.

2.2.4. Additional Proprietary Portlets

In addition to the portlet implementations outlined above, we also built two search clients, one for Moodle and one for Wordpress, the open-source blogging tool.

The Wordpress interface was originally intended as test client for interacting with the SQL target interface implemented for IVA and Zope (see Section 1.1.5). This enabled a new usage scenario, i.e. allowing Wordpress to integrate users' blog entries from IVA.

Originating from testing the OAlster SQL Target, a SQL source implementation in terms of a Moodle block was developed (see [9]).

2.3. Conclusion

Two additional services have been identified in order to ease access for end-users to the iCamp repository network. First, an SQL mediator offers federated search capabilities to any requesting system, thus hiding complexity in a network service. Second, an SQL consumer complements the mediator with a highly re-usable web-based search client that can easily be integrated into the various tools and systems in the iCamp tools portfolio.

Both additional services have prototypically been implemented, thus demonstrating the feasibility of the approaches chosen. Now, they have to be fully realized and extended in order to support end-users, meaning both facilitators and students, during the first iCamp trials.

3. References

- [1] IBM (2006): Introduction to Web Services for Remote Portlets. <http://www-128.ibm.com/developerworks/library/ws-wsrp/>, last access: July 24th 2006
- [2] Hepper, Stefan; Hesmer, Stephan (2006): Introducing the Portlet Specification, http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet_p.html, last access: July 24th 2006
- [3] Wikipedia (2006): Ajax (programming), <http://en.wikipedia.org/wiki/AJAX>, last access: July 24th 2006
- [4] iCamp Consortium (2005): iCamp Technical Annex, p. 36
- [5] OAster (2006): OAster, <http://oaister.umdl.umich.edu/o/oaister/>, last access: July 24th 2006
- [6] The Library of Congress (2006): SRU: search / retrieval via URL, <http://www.loc.gov/standards/sru/>, last access: July 24th 2006
- [7] RSS Advisory Board (2006): Real Simple Syndication, <http://www.rssboard.org/>, last access: July 24th 2006
- [8] Dietrich Ayala (2006): nuSOAP, <http://dietrich.ganx4.com/nusoap/>, last access: July 24th 2006
- [9] Jon Papaioannou (2006): Moodle Blocks Howto, http://docs.moodle.org/en/Blocks_Howto, last access: July 24th 2006
- [10] F. van Assche, E. Duval, D. Massart, D. Olmedilla, B. Simon, S. Sober-nig, S. Ternier, F. Wild (2006): Spinning Interoperable Applications for Teaching & Learning, In: Journal of Educational Technology & Society, 9(2), IEEE Technical Committee on Learning Technology.
- [11] B. Simon et al. (2006): Building Blocks for a Smart Space for LearningTM, In: Proceedings of IEEE ICALT'2006, Kerkrade, The Netherlands, July 2006, pp. 309 – 313.
- [12] Keshav, R. and R. Gamble (1998): Towards a taxonomy of architecture integration strategies, In Proceedings of the 3rd International Workshop on Software Architecture, pp. 89-92.
- [13] P. Avgeriou and U. Zdun. Architectural patterns revisited - a pattern language. In Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), pages 1-39, Irsee, Germany, July 2005.
- [14] iCamp (2006): Interoperability Framework Draft for the Distributed Open Virtual Learning Environment, Technical Report (D3.1), http://www.icamp.eu/content/d3-1_icamp_v16.pdf, last access: July 24th 2006

[15] OpenACS Core Team, Open Architecture Community System, <http://openacs.org/>, last access: July 24th 2006

[16] .LRN Consortium, .LRN Website, <http://dotlrn.org>, last access: July 24th 2006

[17] Bernd Simon, David Massart, Frans Van Assche, Stefaan Ternier, Erik Duval (2005): Simple Query Interface Specification, http://nm.wu-wien.ac.at/e-learning/interoperability/SQI_V1.0beta_2005_04_13.pdf, last access: July 24th 2006

[18] Drupal (2006): Drupal API – do_search, http://api.drupal.org/api/4.6/function/do_search, last access: July 24th 2006

[19] The Library of Congress (2006): Common Query Language, <http://www.loc.gov/standards/sru/cql/>, last access: July 24th 2006

[20] Pavonis (2006): WinForms WSRP Viewer, <http://www.pavonis.co.uk/devnet/framework/wsrpviewer.htm>, last access: July 24th 2006