CLARIN Federated Search

This discussion paper proposes the use of SRU/CQL as common protocol for a Federated Search together with potential necessary adaptations/extensions.

Changes

|  |  |
| --- | --- |
| *Date, Author* | *Changes* |
| 2011-02-01, md | First draft version actually rolled-out (email and svn); |
| 2011-02-28, db | Commented version (sent by email) |
| 2011-03-14, md | Update reacting on and integrating db’s comments;  Incorporates also discussion from Mannheim + emails afterwards.  + added Restricting Search by Repositories/Corpora/Collections (-> 4.5)  + update to Resource / Fragment/Part (refering to ISO/FDIS 24619) and DataView / data types ( -> 5.2.1, 5.2.2, 5.2.3)  + Short reference to Bird&Liberman's Annotation Graph (needs more work)  (-> end of 4.2.3)  + User Management (only as a stub) (-> 6.4)  (sent per email) |
| 2011-03-15, md | Added Changes-log, updated svn-version |
| 2011-05-13, md(Eric) | Adding comments (+ corrected typos) by Eric Auer , naming issues: VLO -> FacetedSearch; IMDI-Browser -> Metadata Search |

Table of Contents

[1 Introduction 4](#_Toc287824906)

[1.1 Status within CLARIN 4](#_Toc287824907)

[1.2 SRU/SRW/CQL 4](#_Toc287824908)

[2 Query Format 5](#_Toc287824909)

[2.1 Conformance Level 0 - Baseline 6](#_Toc287824910)

[2.2 Conformance Level 1 - Indices and Boolean Operators 6](#_Toc287824911)

[2.3 Conformance Level 2 - Parse any CQL 6](#_Toc287824912)

[2.4 Relations and Relation Modifiers 7](#_Toc287824913)

[2.5 Sorting 7](#_Toc287824914)

[2.6 Named Queries 7](#_Toc287824915)

[3 Further features / operations 7](#_Toc287824916)

[3.1 Context Sets 7](#_Toc287824917)

[3.2 Explain operation -Announce Server’s Capabilities 8](#_Toc287824918)

[3.3 Scan operation – Browse the Indices 8](#_Toc287824919)

[3.4 New features proposed in SRU 2.0 8](#_Toc287824920)

[3.4.1 Facets 9](#_Toc287824921)

[3.4.2 Search result analysis 9](#_Toc287824922)

[3.4.3 resultSetTTL 9](#_Toc287824923)

[3.4.4 resultCountPrecision 9](#_Toc287824924)

[3.4.5 queryType 9](#_Toc287824925)

[3.4.6 window 9](#_Toc287824926)

[4 Proposed extensions 9](#_Toc287824927)

[4.1 Existing extension mechanisms 9](#_Toc287824928)

[4.2 Extension: dynamic Indices 10](#_Toc287824929)

[4.2.1 Current solution in MDService: the cmdIndex 11](#_Toc287824930)

[4.2.2 Identifying Tiers in Text Corpora 11](#_Toc287824931)

[4.2.3 Identifying Tiers in Multimodal Resources 12](#_Toc287824932)

[4.2.4 Alignment of the Tiers 14](#_Toc287824933)

[4.2.5 Announcing the tiers 14](#_Toc287824934)

[4.3 Extension: Sequential Tier Search 16](#_Toc287824935)

[4.3.1 Atomic Single-Tier Query 16](#_Toc287824936)

[4.3.2 Proximity / Sequence / Window /Element Query 17](#_Toc287824937)

[4.3.3 Sequence multi-tier query 19](#_Toc287824938)

[4.4 Binding Indices 20](#_Toc287824939)

[4.5 Restricting Search by Repositories/Corpora/Collections 21](#_Toc287824940)

[4.5.1 Announcing repositories / collections 21](#_Toc287824941)

[4.5.2 Announcing Resources? 22](#_Toc287824942)

[4.5.3 Specifying Collections in the request 22](#_Toc287824943)

[4.5.4 Restricting the search by Repositories / Resources 23](#_Toc287824944)

[4.5.5 Handling in MDService 23](#_Toc287824945)

[4.6 Combined Metadata and Content Query 24](#_Toc287824946)

[5 Result Format 25](#_Toc287824947)

[5.1 Data Model 25](#_Toc287824948)

[5.1.1 Concordances, KWIC 25](#_Toc287824949)

[5.1.2 Full resource browse 25](#_Toc287824950)

[5.1.3 Lists/ Word Summaries 26](#_Toc287824951)

[5.1.4 Parallel sequences / aligned tiers 27](#_Toc287824952)

[5.1.5 Syntax Tree 27](#_Toc287824953)

[5.1.6 Geolocation 27](#_Toc287824954)

[5.1.7 MDRecord 27](#_Toc287824955)

[5.1.8 Multiviews 27](#_Toc287824956)

[5.2 searchRetrieveResponse 27](#_Toc287824957)

[5.2.1 New Elements: Resource, ResourceFragment, DataView 28](#_Toc287824958)

[5.2.2 Identifying the Resources or Resource Parts/Fragments 29](#_Toc287824959)

[5.2.3 DataView - Handling the data types 29](#_Toc287824960)

[5.2.4 Schema for Lists 30](#_Toc287824961)

[5.2.5 Metadata view on Content 30](#_Toc287824962)

[5.3 Multi-server response 31](#_Toc287824963)

[6 Architecture 33](#_Toc287824964)

[6.1 Federated Search Proper 33](#_Toc287824965)

[6.2 Potential Supporting Services 34](#_Toc287824966)

[6.3 Combined Metadata Content Search 35](#_Toc287824967)

[6.4 Distributed User Management – Federated AAI 36](#_Toc287824968)

[7 References 36](#_Toc287824969)

[Appendix A Repository List 38](#_Toc287824970)

[Appendix A.1 ZeeRex format 38](#_Toc287824971)

[Appendix A.2 List of candidate centers / search services 38](#_Toc287824972)

[Appendix A.3 Feature Matrix 38](#_Toc287824973)

[Appendix B Candidate Search Engines 39](#_Toc287824974)

[Appendix B.1 CLARIN MDService 39](#_Toc287824975)

[Appendix B.2 DDC 39](#_Toc287824976)

[Appendix B.3 MPI Tools: ANNEX/Trova/ELAN 39](#_Toc287824977)

[Appendix B.4 Nederlandse Familienamenbank 40](#_Toc287824978)

[Appendix C CQL Examples 41](#_Toc287824979)

[Appendix C.1 Metadata Queries 41](#_Toc287824980)

[Appendix C.2 Content Queries 41](#_Toc287824981)

[Appendix C.3 Sequential Tier Search 41](#_Toc287824982)

[Appendix C.4 Metadata Content Queries 41](#_Toc287824983)

[Appendix D Proposed Extensions 41](#_Toc287824984)

[Appendix D.1 dynamic Indices 41](#_Toc287824985)

[Appendix D.2 Context Set: CMDI - Component Metadata Infrastructure 42](#_Toc287824986)

[Appendix D.3 Context Set: CCS - CLARIN Content Search 42](#_Toc287824987)

[Appendix D.4 New Boolean Operator: IN 42](#_Toc287824988)

[Appendix D.5 CCS response Schema: ResultSet, Resource, ResourceFragment, DataView 42](#_Toc287824989)

[Appendix E Mapping to other query languages 42](#_Toc287824990)

[Appendix E.1 SRU -> XPath 42](#_Toc287824991)

[Appendix E.2 SRU -> DDC 43](#_Toc287824992)

[Appendix E.3 SRU -> CQP 44](#_Toc287824993)

[Appendix E.4 SRU -> manatee 44](#_Toc287824994)

[Appendix E.5 Other potential protocols / query languages 44](#_Toc287824995)

[Appendix F From Repository to ResourceFragment View 45](#_Toc287824996)

[Appendix G Related Formats 45](#_Toc287824997)

[Appendix G.1 SRU: searchRetrieve() 45](#_Toc287824998)

[Appendix G.2 SRU: scan() 45](#_Toc287824999)

[Appendix G.3 ZeeRex explain record 45](#_Toc287825000)

[Appendix G.4 CMD 46](#_Toc287825001)

[Appendix G.5 Annotation file EAF-format 46](#_Toc287825002)

[Appendix G.6 TCF 47](#_Toc287825003)

[Appendix H Remarks on GUI, displaying/viewing 47](#_Toc287825004)

[Appendix H.1 Search 47](#_Toc287825005)

[Appendix H.2 Resource Viewer 48](#_Toc287825006)

# Introduction

This discussion paper proposes the use of SRU/CQL as common protocol for Federated (Content) Search together with potential necessary extensions.

The main goal is to introduce a common protocol, to decouple the search engine functionality and its exploitation (user-interfaces, third-pary applications) and to allow (composite) services to access the search engines in an uniform way, leading to a truly distributed SOA environment a federative web of (search) services.

As this is a difficult undertaking, we want to approach it in little steps.

Therefore we define a minimal baseline, that shall be fairly easy to reach, and would serve primarily for setting up the interfaces and testing the linking, but would still already allow a functional system.

Then we propose further layers of complexity, both in terms of expressive power of the queries and functionality, map these to the available features within the protocol and conclude the need for possible extensions. Although the individual features are interrelated, there shall be enough space for complying search service providers to prioritize the features based on own or project interests. This also implies a robust flexible system, that can handle partial implementations gracefully, at the same time being able to get the most out of the features available (see Appendix A.3 Feature Matrix)

## Status within CLARIN

As with CLARIN as whole there is a multitude of (mostly idiosyncratic) content search engines serving very disparate domains / content-types (huge text-corpora, multimodal resources, name databases).

The need for Federated Search – search over multiple Centers/Repositories/Search Engines – was identified on various occasions, especially also within the European Demo Case.

We may need a note about willingness and timing here

Within the CMDI[[1]](#footnote-1) the exploitation side component CLARIN Metadata Service adopted the protocol for searching within the metadata***[[2]](#footnote-2)***.

## SRU/SRW/CQL

This is a protocol suite endorsed by the Library of Congres as the successor of the pre-web standard: Z39.50, widespread in the library networks. It is used broadly in the US/UK and promoted by OCLC WorldCat[[3]](#footnote-3) and also by the European Library[[4]](#footnote-4).

It was submitted to OASIS for standardization as an item for the OASIS Search Web Services Technical Committee[[5]](#footnote-5), where there is continuous activity specifically currently towards a version 2.0 of the protocol (seemingly mainly by one person: Ray Denenberg.).

Generally the work is on a suite of 7 related documents[[6]](#footnote-6):

APD, Bindings for SRU 1.2, SRU 2.0 and OpenSearch, CQL, Scan and Explain

There is an active mailing list:   
<http://markmail.org/search/?q=&q=list%3Aorg.oasis-pen.lists.search-ws>

Note that while this protocol is rooted in the librarian community and has a bias towards metadata, specifically bibliographic metadata, the specs don’t distinguish between metadata or content.

The main acronyms explained briefly:

SRU

protocol for Search and Retrieval via URL [[7]](#footnote-7)  
In essence, it is Z39.50 stripped-down of configuration and architectural complexity, adapted to XML and the web. The current version is 1.2 and there is active work on version 2.0, which shall be fully backwards compatible and introduce new useful features (see 3.4 New features proposed in SRU 2.0)

SRW

Search Retrieve Webservice  
the SOAP version of SRU, also known officially as “SRU via HTTP SOAP”.

CQL

Context Query Language

is a formal language for representing queries to information retrieval systems such as web indexes, bibliographic catalogs and museum collection information. The design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex languages.  
in line with SRU: currently in version 1.2, draft for version 2.0

APD

Abstract Protocol Definition

… presents the model for the SearchRetrieve operation and serves as a guideline for the development of application protocol bindings describing the capabilities and general characteristic of a server or search engine, and how it is to be accessed.[[8]](#footnote-8)

Currently with bindings defined to SRU1.2, SRU2.0 and OpenSearch

# Query Format

In the following we present the individual features of the Context Query Language.

The basic search clause consists of (up to) three parts:

searchClause ::= index relation searchTerm   
 | searchTerm

With term-only queries the server decides in which indices to search. These atomic queries can be combined with Boolean operators to form more complex queries. There is no fixed set of indices or relations to use, rather these can be defined in the so called context sets (see 3.1).

Based on support for individual features SRU defines Conformance Levels 0 – 2 wrt to CQL. We propose to adopt these within our system a) for defining the baseline, b) as a starting point for a Feature Matrix (see Appendix A.3).

Following is mostly cited from the definition of Conformance Levels[[9]](#footnote-9):

## Conformance Level 0 - Baseline

1. Be able to process a term-only query.  
   either a single word or a phrase (quote marks in the term have to be escaped)
2. Respond with diagnostic message*[[10]](#footnote-10)* to unsupported queries

Sums up to a basic full-text search, where the search engine decides in which indices to search.

fish

system

“language acquisition”

“She said \”Yes\””

## Conformance Level 1 - Indices and Boolean Operators

(Support for level 0 plus: )

1. Ability to parse both:  
   (a) search clauses consisting of 'index relation searchTerm'; and  
   (b) queries where search terms are combined with booleans, e.g. "term 1 AND term2"
2. Support for at least one of (a) and (b).

Not necessarily the combination of the two

Allows explicit querying of specific indices

dc.creator = anderson

title adj “wonderful feelings”

bib.dateIssued < 1998

and combine atomic queries with Boolean operators. They all have the same precedence and are evaluated left-to-right. Parentheses may be used to override left-to-right evaluation:

system AND language

system OR language

system AND (language OR acquisition)

system NOT language /\* read: AND NOT; not an unary operator! \*/

One special boolean operator is the proximity operator “allowing for the relative locations of the terms to be used in order to determine the resulting set of records”. It is also the only Boolean operator to take (Boolean) modifiers:

PROX /unit = {unit}  
 /distance {comparison\_operator} {number}  
 /ordered|unordered

cat prox/unit=word/distance>2/ordered hat

cat prox/unit=paragraph hat

## Conformance Level 2 - Parse any CQL

(Support for Level 1 plus: )  
Ability to parse all of CQL and respond with appropriate diagnostics.  
Note that Level 2 does not require support for all of CQL, just be able to parse it.

title any language AND (identifier contains rosettaproject   
 OR publisher contains "SIL International")

title contains Herz and date > 1910 and date < 1920   
/\* alternatively: \*/  
title contains Herz and date within “1910 1920”

## Relations and Relation Modifiers

The default context set[[11]](#footnote-11) defines a number of basic relations and so called relation modifiers, which allow to fine-tune the results. Defined relations:

= >= <= == adj all any within encloses

Some of the defined (functional and term-format) relation modifiers:

/stem /relevant /fuzzy /respectCase /isoDate /oid

## Sorting

A dedicated context-set[[12]](#footnote-12) defines the sort-clause: sortBy, to be used at the end of a cql query:

"dinosaur" sortBy dc.date/sort.descending dc.title/sort.ascending

## Named Queries

The server can provide a unique identifier for a result set by means of header element: <resultSetId>. This id can be used in subsequent requests to reference the result using the index: cql.resultSetId, allowing referencing the result set within a query. Thus after receiving two result sets (with the ids a and b) one could request an intersection of those two via:

cql.resultSetId = "a" AND cql.resultSetId = "b"

or continue restricting the result with:

cql.resultSetId = "a" AND dc.title=cat

Along with <resultSetId> server may supply <resultSetIdleTime> - a good-faith estimate that the result set will remain available and unchanged (both in content and order).

# Further features / operations

## Context Sets

CQL is so-named ("Contextual Query Language") because it is founded on the concept of searching by semantics and context, rather than by syntax. CQL uses context sets to provide the means to define community-specific semantics. Context sets allow CQL to be used by communities in ways that the designers could not have foreseen, while still maintaining the same rules for parsing.[[13]](#footnote-13)

Context sets allow to define “own” indices, relations, Boolean operators and modifiers. List of existing registered context sets: <http://www.loc.gov/standards/sru/resources/context-sets.html> A few application independent indices and the basic set of relations and modifiers is defined in the default CQL Context Set*[[14]](#footnote-14)*. Another canonical context set is that of the 15 dublincore elements.

I couldn’t find out if there is any specific format for these context sets. The published context sets are simply pages adhering to the obvious template...

## Explain operation -Announce Server’s Capabilities

The Explain operation allows a client to retrieve a description of the facilities available at an SRU server. It can then be used by the client to self-configure and provide an appropriate interface to the user. The record is in XML and follows the ZeeRex Schema.[[15]](#footnote-15) [[16]](#footnote-16) describing the server, the database, context sets and indices used, as well as available response schemas and configuration options.

(See the sample Appendix G.3 ZeeRex explain record)

This record shall be retrievable as the response of an HTTP GET at the base URL for SRU server.

So while the context sets allow to define possible indices (to be hopefully reused by many services), the explain operation lists indices that a given service actually provides.

Furthermore explain does not only allow to describe the service’s own facilities, but allows to announce this information also for other known services, so called Friends and Neighbours (F&N) (see more under Appendix A.1 ZeeRex format)

## Scan operation – Browse the Indices

Scan is a supporting operation, allowing the user to find out, what values are actually there in a given index, to support the user in formulating the query. The main request parameter is scanClause, which is a simplified version of the CQL searchClause, allowing to not only state the index to search in, but also a term to start the search from. For example

dc.title=cat

would search in the index dc.title starting in the ordered listed of terms close to “cat “. Further parameters configure the size of the response etc. The response is a sequence of terms:

<sru:scanResponse xmlns:srw=<http://www.loc.gov/zing/srw/> >  
<sru:version>1.1</sru:version>  
 <sru:terms>  
 <sru:term>  
 <sru:value>cartesian</sru:value>  
 <sru:numberOfRecords>35645</sru:numberOfRecords>  
 <sru:displayTerm>Carthesian</sru:displayTerm>  
 </sru:term>  
 <sru:term>  
 ...

## New features proposed in SRU 2.0

The (backwards compatible) draft for Version 2.0[[17]](#footnote-17) of the SRU-protocol and the accompanying CQL proposes a few interesting extensions of the protocol. Not all of them are listed here:

### Facets

Provides means to supply faceted results, ie the analysis of how the search results are distributed over various categories (or "facets").

However the specification seems to foresee the usage of this feature only for the analysis of a result of a query. It is not clear if and how this is applicable to a full faceted browser, where the client is presented with available facets right away, the whole dataset being subject to facet-analysis. This seems related also to the scan operation, which could actually provide this starting point for individual indices/facets.

Nevertheless at least there is this proposal, which could help us to at least get started the work on harmonizing this feature. It would be mainly interesting to compare this to the approach taken with the VLO-FacetedSearch.

### Search result analysis

Search result analysis is an issue somewhat related to the facets. The idea is to provide information for some or all of the sub queries of a complex query (one with Boolean operators).

### resultSetTTL

As opposed to the <resultSetIdleTime> element defined in SRU 1.2 (see 2.6 Named Queries), the new element <resultSetTTL> can be used also as a request parameter and would thus allow to negotiate between client and serv er, how long the result shall remain available.

If supplied, the client is suggesting that the result set need exist no longer than the specified time. The server may (irrespective of the request) also use this element in the response message to indicate the good-faith estimate, how long the result will stay available (under given <resultSetId>).

### resultCountPrecision

The response element<resultCountPrecision> allows the server to indicate or estimate the accuracy of the result count as reported by <numberOfRecords>. The value is a URI, identifying a term from a controlled vocabulary.

### queryType

A new request parameter to indicate the syntax (query language) of the string in the query-parameter. Default is “cql”, “searchTerms” is reserved, suggesting that the query consists of a list of terms separated by space (like “cat hat rat”).

### window

Be able to formulate a multi-term query within a defined window:

“rat hat cat” within 10 words window.

# Proposed extensions

## Existing extension mechanisms

The protocol caters for extensibility at multiple locations:

Context Sets

One can define own context sets with indices, relations, Boolean operators and modifiers.

See more under 3.1 Context Sets

extraRequestData

Extra information can be provided in the request by extension parameters. These MUST begin with ‘x-’.

extraResponseData

The response can carry extra information in the <extraResponseData> element.

Moreover besides this information on top level for the whole response there are analogous elements foreseen in the detail level of the response[[18]](#footnote-18):

<extraRecordData>, <extraTermData>, <extraOperandData>

Record Format

The defined structure of the searchRetrieveResponse is actually only the envelope down to the individual record. The format of the individual records actual result data can be anything, subject to negotiation between the capabilities (available schemas) of the server and requirements of the client.

The extensions can be registered and could eventually do it into the standard.  
Some registerd extensions: <http://www.loc.gov/standards/sru/resources/extensions.html>

## Extension: dynamic Indices

In the context of SRU/CQL a number of context sets each with a list of indices has been defined, most notably the canonical example of the 15 dublincore terms[[19]](#footnote-19). Most applications in the usual/main application domain (library search) should get by with the already defined indices. If not anyone can decide to define its own context set. However the context set mechanism foresees a static flat list of indices, while within CLARIN we are confronted with the issue of an open index-list due to use of structured data at least in two occasions:

CMDI

CMDI is conceived as an open system where a multitude of profiles (or schemas) can coexist, where every profile can have a different structure with different fields/elements defined.

Annotation Layers / Tiers in Annotation files

The annotations of multimodal resources carry arbitrary (project, or even resource specific) tiers –layers of typed information.

We will go into detail of both aspects in the following chapters, but one general remark beforehand: As the context set is meant as a static list, it is apparently not usable for our needs. But we can simply use the more flexible explain-operation, where the service can indicate on every request the indices actually available.

Nevertheless, and even if only for the sake of simpler referencing in the documentation, we want to introduce these two “virtual” context sets:

CLARIN Metadata

cmd. or cmdi.

CLARIN Content Search

ccs.

### Current solution in MDService: the cmdIndex

MDService allows the so called cmdIndex in the index-part of the CQL-query.

It is based on the recursive component model of CMD and maps every element in every profile to an index based on its “XPath”. So for example: Session/MDGroup/Actors/Actor/Role would trivially map to an index Session.MDGroup.Actors.Actor.Role, or as minimal unique index: Session.Actor.Role.

Furthermore thanks to linking the elements of CMD-profile via attribute @ConceptLink to Data Categories MDService is also able to provide data category based indices, like: dc.title, which are internally translated to appropriate profile based indices.

Currently MDService does not really honor or understand context sets. But the CMD profiles imply a natural grouping of the indices and could be seen as individual context sets. On the other hand they could just as well be prefixed with a common “virtual” context set cmdi wrapping all the profiles. However this is a bit of a futile thought experiment, as the protocol does not deliver a possibility to formulate a context set with open or recursive index definition. Thus we can let us guide by rather pragmatic considerations.

For illustration - given current state of the repository[[20]](#footnote-20) - following are some of the “context-sets” available:

Session [cmdi:imdi-session] OLAC-DcmiTerms [cmdi:olac]

teiHeader [cmdi:tei] TextCorpusProfile [cmdi:tcp]

Dublin Core Elements [cmdi:dce] Dublin Core Terms [cmdi:dct]

ISOcat DCR [cmdi:isocat] …

### Identifying Tiers in Text Corpora

The situation for text-corpora seems rather manageable: The canonical and most-widespread indices /annotation layers are PoS and lemma. Even if some collection define and use further indices, they are defined at the corpus level and could be easily bound to appropriate data category in the Technical Metadata Component of the metadata record of the corpus.

Also the usual primary sequence are the running words/tokens and the other annotation levels map one to one on these, in other words every token has a PoS and lemma assigned. This allows a very efficient structure for encoding the corpus as input for the corpus indexer - the verticale – a file, where every running word is on one line together with its (linguistic) annotations:

*#word #pos #lemma*

Das PRO d

ist V sein

ein ART e

Haus N Haus

. \$. .

The corpus search engines also support a grouping annotations or “breaks”, defining segments/regions within the text, the canonical examples being: sentence, paragraph and document (<s>, <p>, <doc>).

Consider TCF wrt definition of Annotation Layers!

### Identifying Tiers in Multimodal Resources

For multimodal resources the situations seems more complicated: The individual tiers are named ad hoc and on record-level. Thus every resource has its own tiers, that are defined in the annotation files to the resource.

For multimodal resources the situations is more complicated: The individual tiers are named “ad hoc” and on record-level. Thus every resource has possibly its own tiers, that are defined in the annotation files to the resource. These can be even named equally, although they are not related.

If we consider an example Annotation File (seeAppendix G.5) which carries 14 <TIER>s:

<TIER DEFAULT\_LOCALE="nl" LINGUISTIC\_TYPE\_REF="Words" PARENT\_REF="V40069-Spch" PARTICIPANT="V40069" TIER\_ID="V40069-Words">

|  |  |  |  |
| --- | --- | --- | --- |
| ***@PARTICIPANT*** | ***@PARENT\_REF*** | ***@TIER\_ID*** | ***@LINGUISTIC\_TYPE\_REF*** |
| V40069 |  | V40069-Spch | Spch |
| V40069 | V40069-Spch | V40069-Words | Words |
| V40069 | V40069-Words | V40069-PoS | PoS |
| V40069 | V40069-Words | V40069-Lemma | Lemma |
| V40069 | V40069-Words | V40069-Phonetic | Phonetic |
| V40069 |  | V40069-Phonetic@left | Phonetic |
| V40069 |  | V40069-Phonetic@right | Phonetic |
| UNKNOWN |  | UNKNOWN-Spch |  |
| *UNKNOWN analogous to first Participant …* | | | |

we can distinguish/classify the tiers by at least three aspects:

ID

@TIER\_ID

Type

@LINGUISTIC\_TYPE\_REF = Lemma Phonetic PoS …

Participant/Actor

@PARTICIPANT

Hierarchy

the tiers can refer to a parent tier, yielding a reference hierarchy:

V40069-Spch

+ V40069-Words

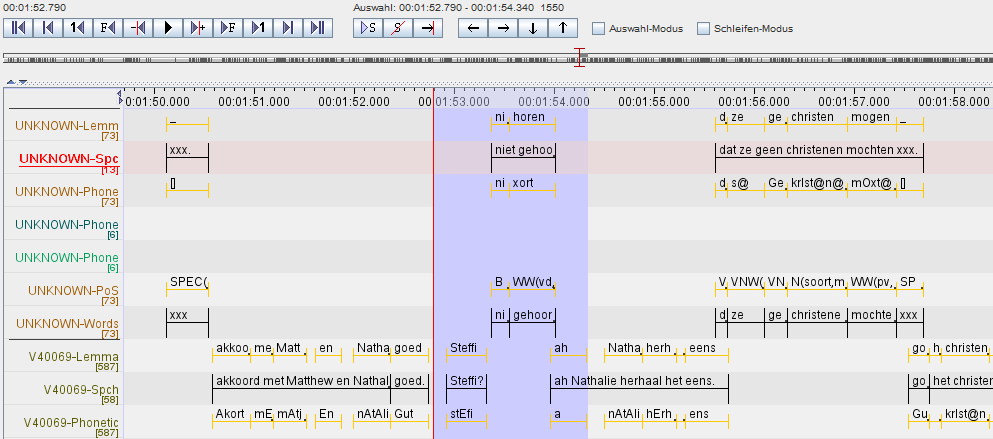
+ V40069-PoS

+ V40069-Lemma

+ V40069-Phonetic

Not all tiers are linked to a parent tier. The affiliation to a @PARTICIPANT can be seen as top-level parent, but also this is an optional attribute.

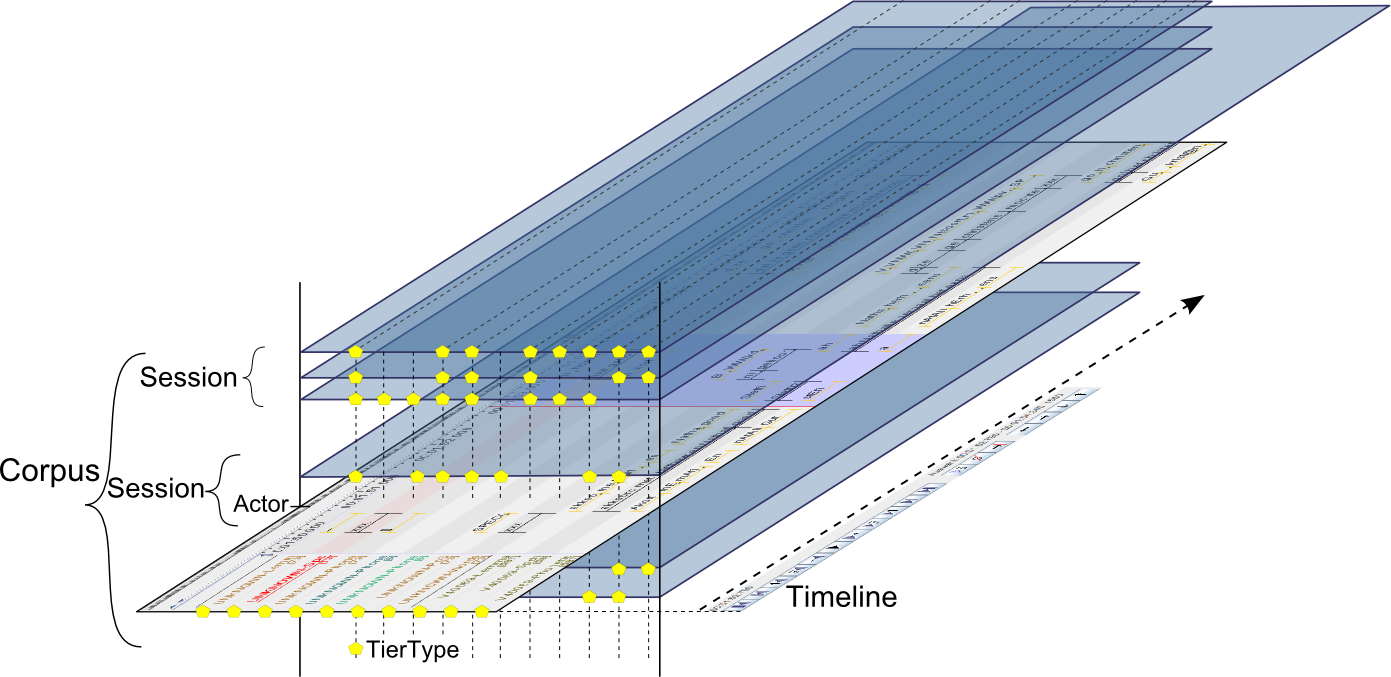
Figure 1 Screenshot of the TimeLine UI-widget in Annotation Editor ELAN



For better grasping of the data we are dealing here with Figure 1 depicts the timeline with the multiple tiers and annotations therein as provided by the annotation editor ELAN[[21]](#footnote-21). Figure 2 tries to put this “single piece of data” into a broader perspective. It illustrates a collection/corpus (within a content provider’s repository) consisting of Sessions (each being associated with the original Multimedia file), which in turn “accommodate“ multiple Actors/Participants, with a vector of tiers for every Actor. The vertical guiding principle is the TierType, suggesting that all tiers can be (or even have to be) typed according to one typing system. Although probably no Session or Actor will cover all TierTypes. Or in other words: TierType’s domain is the aggregation of all types used in the repository.

It may feel a bit confusing having the individual Actors of a Session not on one layer, but rather grouped under each other. This tries to emphasize the fact that within one Session there are tiers with the same type for every Actor (Actor1.gesture, Actor2.gesture). Moreover next to metadata about Corpus and Session, the MDrecords usually accommodate additional information about the Actor (.Role, .Sex, .EthnicGroup, …)[[22]](#footnote-22).

Figure 2 TierRack - A conceptual visualization of the structured collection of multimodal resources



We have to be careful not to overspecialize on one format. For a more general coverage/discussion about possible structures we should consider the Annotation Graph proposed by Bird & Liberman[[23]](#footnote-23).

### Alignment of the Tiers

The crucial difference between annotation layers and any other fields or indices is, that they are aligned among each other, ie an annotation in one tier applies to a specific point or region in the sequence of another annotation. This referencing is rooted in a primary sequence, normally running words/tokens in a text, or the timeline of the soundwave, but not all tiers have to reference the primary sequence, instead they can refer to other tiers (see the attribute <TIER@PARENT\_REF>)

If we look again in the example EAF-file we can see mulitple levels:

* TIME\_SLOTs segmenting the time-line.

<TIME\_SLOT TIME\_SLOT\_ID="**ts1**" TIME\_VALUE="950"/>

* Basic tiers linking Annotations to the TIME\_SLOTS

<ANNOTATION>

<ALIGNABLE\_ANNOTATION ANNOTATION\_ID="**fv400279.1.1**"

TIME\_SLOT\_REF1="**ts1**" TIME\_SLOT\_REF2="ts2">

* and aligned tiers, referring to the Annotations in the Basic tiers.

<ANNOTATION>

<REF\_ANNOTATION ANNOTATION\_ID="fv400279.1.1-pos"

ANNOTATION\_REF="**fv400279.1.1**">

### Announcing the tiers

Given the structure described in section 4.2.3 we need to find paths to convey the information about the available tiers to search in to the client.

The primary source for this information are the annotation files or any dedicated records in the repository of the content provider. So in general the content provider has to expose the list of available tiers by aspects mentioned earlier: TierType, TierID/TierName, Participant. From the point of view of search-protocol/request the TierType seems to be the most useful information. However the TierIDs can still be useful, if one wants to search in specific individual tiers. And Participant is important as it constitutes the link to the additional information about the Actor/Participant stored in the MD description.

Trova currently already provides the list of tiers, grouped by: Tier Type, Tier Name and Participant. This is achieved by parsing the documents when inserted into the archive and putting the annotations with relevant attributes into a DB.

(In the following we focus on the TierType, but analogous holds for the other aspects.) As Figure 3 suggests the Content Provider has to aggregate over all annotation files, to yield a list of TierTypes covering all tiers in all files in the whole repository. In a federated search scenario, the composite service Federated Search has to collect this information from every Repository and produce and expose an aggregated list thereof.

An important aspect is the need for “early” linking to recognized data categories. The aggregated list provided by the repository actually already needs to be a list of data categories. Otherwise no useful merging between repositories is feasible. There is a data category AnnotationLevelType[[24]](#footnote-24), which seems to be a starting point here, but it has an open Conceptual Domain, so the actual list of data categories still needs to be worked out and agreed on.

Does it? Or what is there already? For example there is partOfSpeech: DC-1345 and DC-396 (Profiles Terminology and Morphosyntax) …? http://www.mpi.nl/IMDI/Schema/AnnotationUnit-Type.xml

The second challenge is (if and) how to fit this procedure within the limits of the SRU protocol-suite. In the following two variants are proposed:

1. Announce every (aspect of a) tier as own index in the explain operation (similarily to as Trova currently provides). So the <indexInfo> list would contain indices like:

TierName:I’sGest, TierName:Damian, TierName:Unknown.WORD, TierType:English, TierType:PoS, TierType:Word, TierType:Gesture

While this could work for TierType, it would lead to an probably unbearable bloating of the explain-response for TierName/ID or Participant. ad TierName,

Figure 3 Providing the information about available Tiers - Version 1 explain

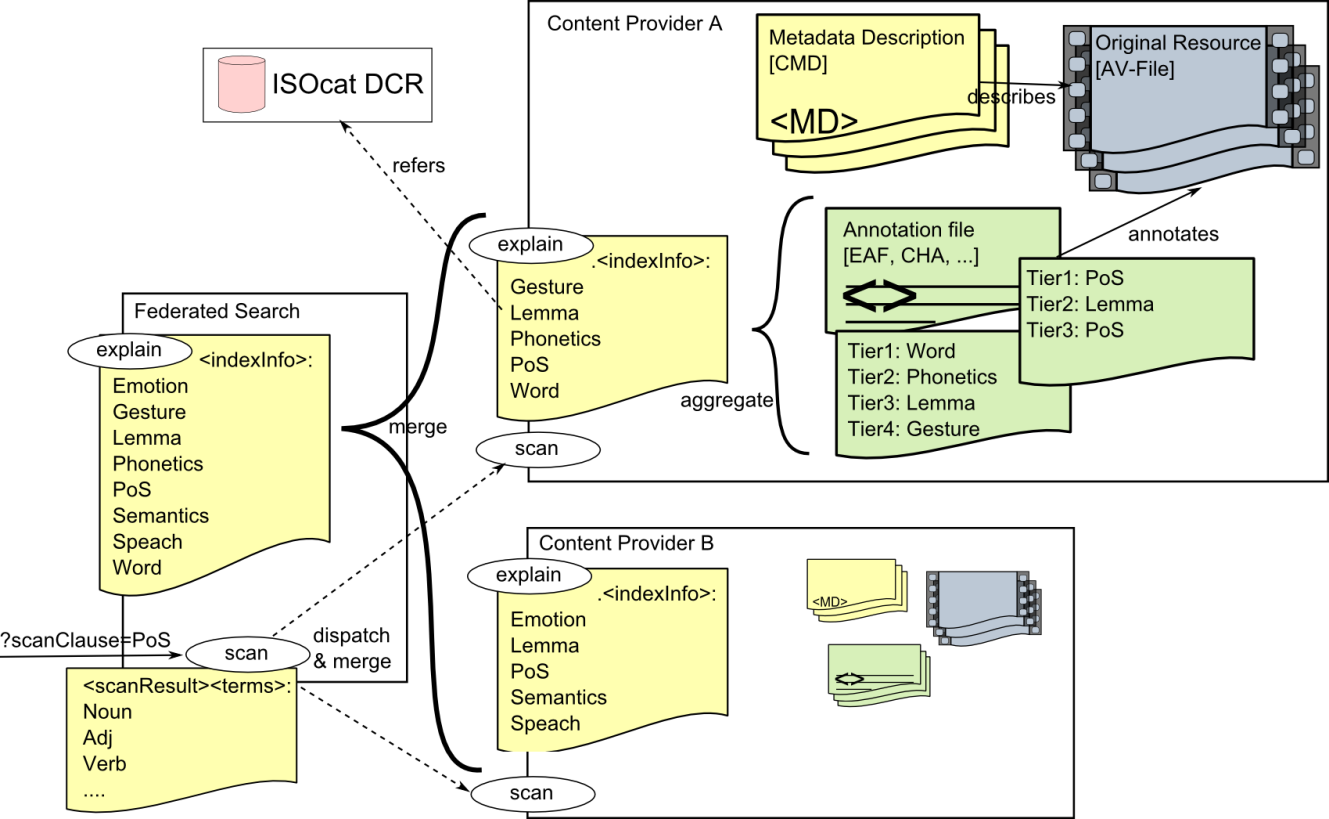
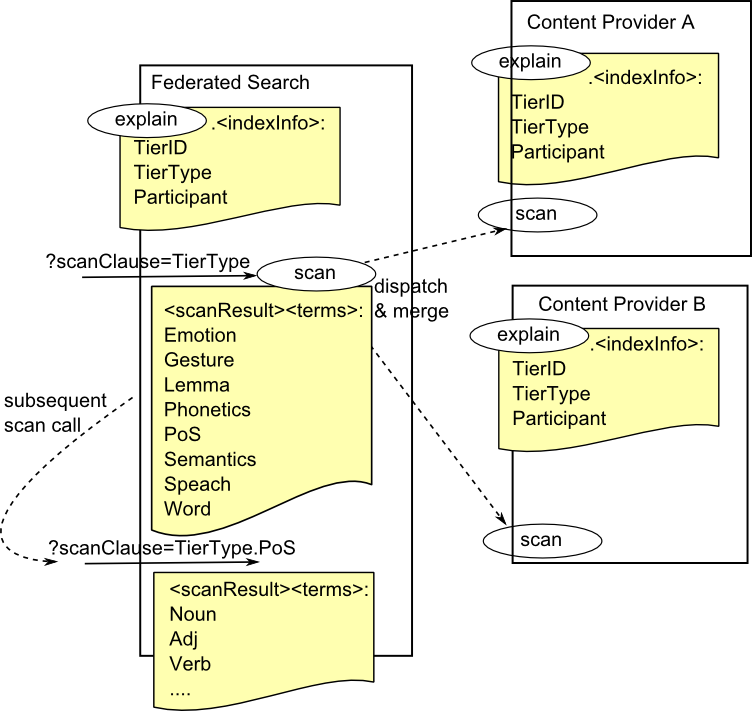


Figure 4 Providing the information   
about available Tiers - Version 2 scan

1. Provide (tentatively) three static indices:

TierType  
TierName  
Participant

and scan those to get the available (aspects of the) tiers. The drawbacks is that the client wouldn’t be able to appropriately self-configure based on the explain-response, but only after 3 further scan-calls, which is counterintuitive as scan-operation is expected to return values present in an index and not “sub-indices”. Also it would need to perform a “second level scan” to get listed the actual values used in a given index/tier.

Any way we choose, as we already stated at the beginning of this chapter, we have to deal here with a dynamic system of indices. Thus we have to agree on the syntax of the indices and should take care of naming the indices be consistent between Metadata and Content – the cmd and ccs ”context sets”. This means especially a kind of hierarchical path-like syntax for the indices.

A few examples (See the proposal and examples in Appendix D.1 dynamic Indices):

cmd.Session.Actor.Name

Collection.Project.Title

ccs.TierType.PoS

isocat.PoS

## Extension: Sequential Tier Search

The special data structure of multimodal resource implies special requirements on the search process, the two crucial aspects being

Tiers

The original resources are associated/enriched with multiple tiers or annotation layers - carrying different types of data - that are aligned among each other and aligned with the

Sequence

a fixed order constituted by the primary resource. (Which is usually the sound-wave or sequence of tokens.)

In the following we try to work out the individual aspects and the possible binding to SRU:

### Atomic Single-Tier Query

SRU defines the basic search clause (index relation term) and expects indices statically defined (in context sets) and announced in the explain-operation. Here we refer to the discussion in previous chapter (4.2 Extension: dynamic Indices) about the issues wrt to defining indices specific to our application domain and assume here that the client was able to retrieve enough information about available indices.

Some examples of atomic queries we would like to express and possible CQL encodings (We will come to more complex queries in the following chapters):

search in all tiers of type:word for the sequence “To be or not to be”

word adj „To be or not to be”

search in all tiers of which type is linked to ISOcat’s data category partOfSpeech for the tags that are linked to ISOcat’s data category noun

isocat.partOfSpeech = isocat.noun

As search terms are not expected to be bound to a namespace we would probably need to rewrite the above with the help of a (term-format) relation modifier.

isocat.partOfSpeech =/isocat noun

search in all tiers of type:PoS for a tag starting with N

TierType.Pos regexp /N.\*/

search in the tier named “V40069-Lemma” for any of the words “symboliek”, “godsdienst” or “christen”

TierName.V40069-Lemma any „symboliek godsdienst christen”

Wrt previous discussion about dynamic indices and “sub-indices” we could think of alternative ways of formulating the query.

1. sub-index as relation modifier

Tier regexp/tierType=Pos /N.\*/

Tier any/tierName=”V40069-Lemma” „symboliek godsdienst christen”

1. sub-index as separate atomic query

TierName = V40069-Lemma AND text any „symboliek godsdienst christen”

### Proximity / Sequence / Window /Element Query

The other important aspect is the ability to search along the defined sequence, especially find co-occurrences of “events” within a given relative distance. Simple example:

“Er” followed by “sie” within 10 words

“Herz” and “zerreißen” within one sentence

For this type of query CQL already provides the boolean operator PROX .

PROX /unit = {unit}  
 /distance {comparison\_operator} {number}  
 /ordered|unordered

Yielding following CQL-query for the examples above:

Er PROX/unit=word/distance<=10/ordered sie

Herz PROX/unit=sentence/distance=0 zerreißen

Proximity queries take a few parameters bloating the combinatory space:

1. number of regarded terms (usually two, but may be more)
2. unit of distance (word, sentence, paragraph, seconds)
3. distance (0-n, zero meaning *in the same unit*)
4. comparator ('=' exactly, '<' less than (no more than), '>' more than (at least))
5. in element, meaning the terms shall both occur in the same containing element.

Parameters 2. to 4. are well supported in CQL (and Z39.50). However there are two distinct problems wrt to **proximity**: *same element* and *window*, which are both recognized by the authors of CQL and addressed in the version 2.0 of CQL.[[25]](#footnote-25) (Although same element is not yet explicitly mentioned in the CQL-2.0 draft.)

window

(a, b, c) IN window(#)

Find "cat", "hat", and "rat" within a 10-word window.

Currently it is not possible to formulate these types of queries for more than two terms (or atomic queries in general). Of course one can concatenate multiple terms with PROX-operator:

hat PROX/w/<=10 cat PROX/w/<=10 rat

but one cannot say “hat”, “cat” and “rat” within 10 words window.

To tackle this problem the new version of CQL proposes a new relation modifier windowSize to be used with the relation all:

all/windowSize={number}

allowing queries like:

word all/windowSize=10 “hat cat rat”

However this works only for simple terms, not for full search Clauses or even more complex subqueries. (See next chapter.)

same element

Similar to window query, but the “window” is defined by a structural element.

(a, b, c) in Elem

CQL proposal (bib-data biased):

example: Find the name "adam smith" and date "1965" in the same author element.

bib.name ="adam smith" PROX/element=bib.author dc.date =1965

So CQL proposes the relation modifier element, but why not use existing modifier unit. There does not seem to be a principal difference between a word or sentence as an element and any other complex element (i.e. element with internal structure)

bib.name ="adam smith" PROX/unit=bib.author/0 dc.date =1965

If we apply this on a tier search we get similar as at the starting example (trying to exclude the multi-tier issue here yet):

Herz PROX/unit=sentence/distance=0 zerreißen

And as long as we are “willing to stay” really within one element, we could even extend this to multiple sub queries

Herz PROX/s/0 zerreißen PROX/s/0 (Leid or Liebe)

However if we wanted to go over the boundary of one element, we are faced with the same problem as in window.

other element

The CQL does not cover this issue at all. But starting from the proposals on same element, one possible (elegant?) encoding could be:

bib.name ="adam smith" PROX/unit=bib.author/>0 dc.date =1965

meaning the element is bib.author, but the distance has to be more than zero, i. e. it may not be the same element.

### Sequence multi-tier query

As we showed in the previous section, the current CQL restricts the proximity queries to either only simple terms or only two subqueries.

Words starting with “N” followed within one sentence by a pos-tag starting with “V”.

Word “Ja” followed by a gesture laugh within 3 seconds.

word=N\* PROX/sentence/0 pos=V

word=Ja PROX/seconds/<3 gesture=laugh

So to reach the goal of a general sequential tier search – allowing complex combinations of tier- and proximity-conditions we propose another extension:

multi-tier window

We want to be able to express a query   
with n(>2) sub queries cooccurring within a given relative distance .

For this we propose a new boolean operator: IN (or HAS as inverted syntactic sugar)

(Q|A, B, C) IN Window?  
or:  
Q has (Q)

IN being interpreted on the sequence.

( Actor.X.w=Ja PROX/w/4 Actor.Y.emotion =laugh   
AND Actor.Z.gesture=”clap hands”

AND Actor.w adj “wonderful feeling”

) IN Paragraph

or:

) IN PROX/min/2

Unfortunately this is not exactly CQL, because the second part of the query is not a valid searchClause, neither a simple-term. But we could make it one:

Paragraph = \*

Time PROX/min/2 \*

Generalizing the latter part to a query would deliver a sort of a general filtering-mechanism

We could even think of applying this recursively: Q IN Q IN Q …   
or positive negative filters Q NotIN Q IN Q … but this is pure speculation.

The IN operator will need a modifier to further fine-tune the proximity relation. It could be called overlap and take one of a set of values inspired by these, used as distance modifiers in TROVA:

* Fully aligned
* Overlap
* Left Overlap
* Right Overlap
* Surrounding
* Within
* All combinations of: begin/end time, begin/end time and =/>/<
* No Constraint

Moreover TROVA provides special modifiers between individual conditions on one tier:

{number} (=|>|<) (annotations|miliseconds)

## Binding Indices

We have to be aware that one index can describe/match multiple tiers even in the same resource.

For example index TierType.PoS would match two tiers in the sample resource one for each actor. So we need special provisions to express, that two conditions apply to the same Tier:

Inspired by the solution in the advanced search IMDI browser, we propose a modifier var

{rel}/var=(X|Y|Z,…)

This would allow queries like:

TierType.PoS =/var=X noun PROX/s/0 TierType.PoS =/var=X verb

But also query explicitly distinct tiers:

TierType.PoS =/var=X noun PROX/s/0 TierType.PoS =/var=Y verb

In the metadata search the queries could be:

Actor.Role =/var=X Annotator AND Actor.Age >/var=X 40

AND Actor.Role =/var=Y Speaker AND Actor.Sex =/var=Y Female

As we mess around with the index part of the search clause anyhow, we could also think of a shorthand notation:

Actor.(X).Role

which would have the advantage of not being ambiguous wrt which element shall be bound. The brackets are necessary to distinguish between X as variable and X as possible component of Actor. Another more complex example:

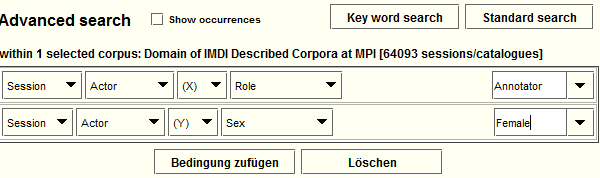
One Actor said “Ja” and an other Actor reacted with laugh within 4 words, or yet another Actor clapped the hands within 3 seconds and this all should have happened within 2 minutes with any Actor stating: “wonderful feeling”

(Actor.(X).w=Ja PROX/w/4 Actor.(Y).emotion =laugh)

OR (Actor.(X).w=Ja PROX/sec/3 Actor.(Z).gesture=”clap hands”)

PROX/sec/120 Actor.w adj “wonderful feeling.”

Figure 5 Advanced Metadata Search for IMDI-Corpora distinguishing the indices with variables (X, Y)



## Restricting Search by Repositories/Corpora/Collections

Until now we only “searched” on the content (xor in metadata). But obviously – especially if we think of a federated search with many repositories linked in – the client will seldom want to search over all the content available (Apart from the question if it is a good idea performance-wise.).

So we need means to restrict the dataset to search in. The simplest way is to select one repository/collection/dataset out of a list the search service provides. The corresponding feature in Trova search is the Domain-parameter. This is a required parameter that allows to select the resource (or corpus) to perform content search in.

If we generalize this a bit, we should of course accept a multi-selection as well. (And it may be necessary to allow to search even in individual resources in some cases – see 4.5.2).

And although following are tightly related we still have to distinguish between

1. restricting the federated search by repositories,
2. restricting the search (even a non-federated search in one repository) to a list of collections
3. a federated search in selected collections (in multiple repositories), where either
4. the federated search is able to backtrack the right repository for given resource or
5. the user would have to provide both repository and collection parameter, or
6. all repositories would have to be queried.

The first option seems the most “right” one, but depends on the way the information about existing collections will be communicated between the components (see 4.5.1 Announcing Repositories / Collections) .

### Announcing Repositories / Collections

Theoretically we could start with just a simple list of URLs in a config-file of the composite service, but such solution would most probably not suffice even in the simplest setup. Even in the first EDC prototype we already need at least two levels:

Repository list

the root-list that lists all available entry points

Collections/Corpora list

the list offered by individual repositories announcing the their searchable collections.

Even if there may be repositories, that can be only searched in whole, already in the first phase we have repositories offering individually searchable collections (TROVA, C4).

It is only natural to think this list nested (yielding a collection tree).

One task of the composite service should be to provide an aggregated collection-list, e.g. as one tree with all the repositories listed in the first level, continuing with the collections as children. However this is just one possible data structure. In the concrete realization, this tree may be inverted, listing all collections in a flat-list, linking to their respective repositories.

We could think of handling repositories and collections (resources) uniformly, ie within the same data structure (tree). However the available SRU features suggest a separate handling:

There is a special provision in the explain-record – the default mechanism for announcing capabilities – for describing not only the originating server, but also other servers, so called Friends and Neighbours[[26]](#footnote-26). (The corresponsing element serverInfo is described in Appendix A.1 ZeeRex format). That seems a clear candidate for the repository list.

Regarding the collections the adaptation of the protocol is not so clear: The explain operation allows among others to list the available indices – followed by the scan-operation to list the values of those indices.

However although it would be straight-forward to introduce an index collections and in a simple (non-hierarchical) case this should work without any problem, it is questionable if the hierarchical nature of the collections can be expressed within the scan-response.

This problems seems similar to that of Announcing the Tiers (chapter 4.2.5), although there we had to deal only with two levels.

Proposed solution for request: The scan operation (see 3.3) allows to specify a starting point within the index, by passing a simple searchClause on the scanClause parameter. So it would be perfectly valid to request:

?operation=scan&scanClause=collection {=} {collection-handle}

The logic would be left to the server, of not trying to scan a flat index, but interpreting it as the root-node of a sub-tree in the collections-hierarchy.

In an simple/flat one-step response (only the children are listed), we could do fine with the present structure:

<sru:scanResponse xmlns:srw=<http://www.loc.gov/zing/srw/> >  
<sru:version>1.1</sru:version>  
 <sru:terms>  
 <sru:term>  
 <sru:value>{child-collection-handle}</sru:value>  
 <sru:numberOfRecords>{resource count in collection}   
 </sru:numberOfRecords>  
 <sru:displayTerm>{collection label/name}</sru:displayTerm>  
 </sru:term>  
 <sru:term>  
 ...

However we would have to extend both the request and response, if we would like to be able to retrieve a resulting tree of arbitrary depth.

### Announcing Resources?

Although this is probably not a high-priority feature, the user may even be interested in restricting the search to only specific Resources, so we also may need to consider the traversing of the collections-structure down to the leaves, i.e. the individual resources. It does not seem to be a problem to fit this information in the above data structure, as Collections and Resources can be seen as equivalent (they are both Resources identified by a PID), but it does not feel like a task for the federated-search component and rather an issue to be handled only in the Metadata-Content Search (4.6) as the MDService is concerned extensively exactly with the issue of Resource Collections (4.5.5). In this context also the Virtual Collection Registry has to be mentioned, which would fit in perfectly, exactly for the very likely requirement/task of defining a (reusable) selection of resources to be searched in.

### Specifying Collections in the request

There seem to be basically two ways of identifying the collections in the query-request:

in the query

integrating them in the CQL-query itself, introducing a separate index "collections" or so, where allowed values would be a list of handles, so that the queries look like:

{search-term} AND collections= {collection-handle}

extra parameter

SRU foresees extraRequestData within its extension mechanism:  
<http://www.loc.gov/standards/sru/specs/common.html#extraData>

where additional parameters can be introduced, they just have to start with "x-" (and should continue with some short namespace identifier) so in our case the parameter could be:

?x-cmd-collections={collection-handle}

### Restricting the search by Repositories / Resources

Next to accomodating the collections in the query-request, it has to be decided yet how to handle repositories on one hand and individual resources on the other.

Whatever solution we choose for the collections (parameter or index - see previous chapter), we are left with two basic options for repositories and resources (in relation to the collections):

separate handling

there would be a separate parameter (or index) for each: repository, collection and resource.

This may make it easier for the processing service to decide what to do with the parameter/index.

uniform handling

that would mean, that the same way as the query can be restricted by specifying one or more collections, one could equally well constraint it by a resource (by its PID) or a repository (by its endpoint?).

This would require a renaming of the parameter/index. One proposal being context:

ccs.context or cmd.context /\* as index\*/

x-ccs-context or x-cmd-context /\* as parameter \*/

And oit would also put an additional load on the processing logic of the composite service. Especially the mapping between the colleciton/resource and the appropriate endpoint may be an resource-intensive task and should be externalised (as proposed in 4.5.2).

In the light of the proposed uniform handling we should consider to also generate a MDRecord for the repositories. The MDRecord would minimally carry the end-point url and some human-readable name, but could be used also for all sorts of descriptions, especially also containing the Technical Metadata Component, formally describing the interface (the available Tiers, and ResultFormats and their encoding as the Input/Output parameters).

### Handling in MDService

In MDService/Repository we talk of collections (as the "natural" hierarchy of the datasets).

Wrt to announcing of the collections and specifying them in the request we adopted the latter approach (extra parameter):

MDRepository and MDService provide separate operations to retrieve the tree of collections

getCollections() /\* MDRepository[[27]](#footnote-27) \*/

collections /\* MDService[[28]](#footnote-28) \*/

and they accept the (optional repeatable) parameter collections in the search request[[29]](#footnote-29) to specify the collections (by their handle). Although MDService accepts CQL-queries the REST-interface is not SRU-conform yet, but a SRU-conformant interface-extension is planned (, that would adhere to the decision made in this FCS working group wrt to handling of the collections argument).

MDService also provides a repository-parameter, that enables the client to address dynamically any of the installed MDRepositories. Currently these are configured internally in the MDService and identified by an integer :-/.

## Combined Metadata and Content Query

Another related requirement is to be able to define the set of resources to search in intensionally – by a metadata query. The simple example (multimodal):

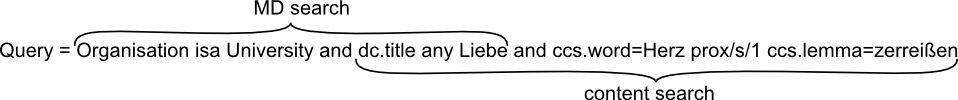
Find all occurrences where a female Actor said “Ja”.

Or textual:

Find all occurrences of “viable system” in texts where the Organisation responsible for the collection the text is part of is a university.

Part of this request can only be answered (if at all) with the information from metadata descriptions.

There is a further complication to this: parts of the query may be relevant both for the metadata search and for the content search. In the following example the middle part of the query:   
dc.title any Liebe , means the title of the resource shall contain the word “Liebe” and could be used by Metadata search to restrict the resources/collections to search in, but could (needs to) be also used by the search engine aware of the titles of the documents it is providing:



If we want to express a query like the one above with female Actor saying “Ja”, we need a way to bind the individual conditions. Just stating:

Actor.Sex = F AND TierType.w = ”Ja”

is not enough, because, there are normally multiple actors in a Session and a male Actor could have said “Ja”. So we need to employ the binding of indices as introduced in the previous chapter, to yield something like:

Actor.Sex =/var=X F AND TierType.w =/var=X ”Ja”

Another example with a sequence query involved:

Find passages, where Actor with the Role Interviewer said “Ja” and another (the second) actor laughed within 3 seconds after.

Again here part of the query can be only answered from within the metadata. We find the information in the CMDI/IMDI-Actor component:

...<Actor>  
 <Name>  
 <Role>  
 <Sex>

If we try to visualize this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *index* | *sub-index* | | *modifier* |  | *Sequence* | | | |
| *MDQuery* | | | | |  | | | |
| Actor | .Role | | X | Interviewer |
|  |  | |  |  |
| *Content Query* | | | | |
| Actor | | .w | X |  | Ja |  |  | *continue 🡪* |
| Actor | | .emotion | Y |  |  |  | laugh |
| *distance* | |  |  |  |  | 4 words  | 3 sec |  |
| *add Tiers...* | | | | | | | | |

Sidenote: CQL is agnostic about the query being on metadata or content.

# Result Format

## Data Model

This chapter shall collect the various data-types we need to represent in the response. We need to provide for the variety/diversity of data, but at the same time prevent proliferation, i.e .we have to think of minimizing the number of types, providing a few generic ones but each optionally sub-typable on demand.

The definition of data-types and their expression in the result format is also related to the use of the data at the client for display (see Appendix H Remarks on GUI) .

### Concordances, KWIC

Keyword in Context - a basic response format in text-corpora (or any text-search from Google to Drupal) . The matching keyword with surrounding textual context and some (usually bibliographic) metadata about the text/document/resource the snippet originates from.

[(bib-metadata fields,  
 text snippets: left context, keyword, right context)]

For text-corpora even if the match is on some other tier (e.g. PoS=Noun), the keyword is highlighted in the primary sequence - the text. The other tiers may even not be available in the result. If they are present, then they are aligned on the token level.

Special problem is the count of hits, the two options being counting just the Resources where the hit occurs, or counting the actual hits. Both numbers can be relevant.

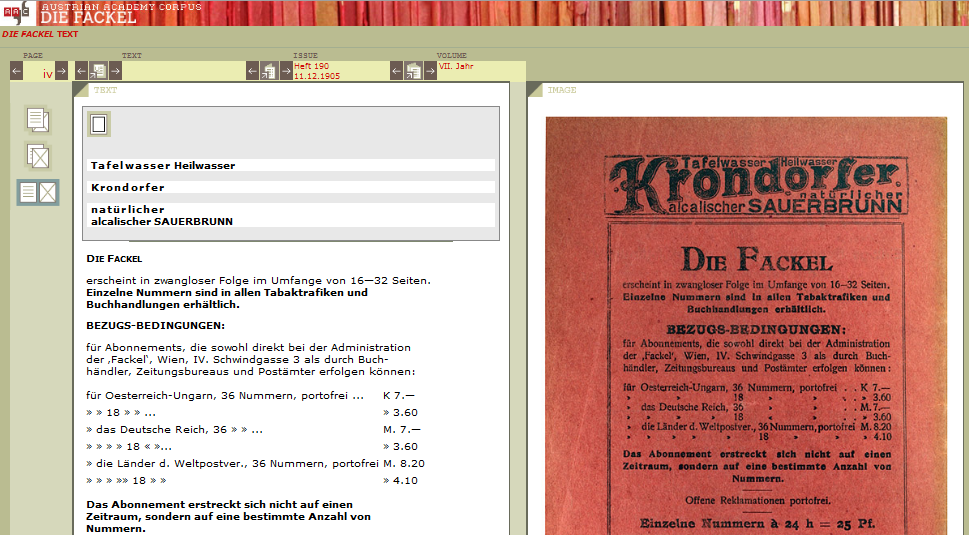
How would this look like for multimodal resources? The matching tier should definitely be shown, probably together with all the other tiers? However it may be complicated to extract the context from every tier and send just an aligned snippet (see Figure 1) and easier to identify a fragment of the resource in terms of time-code.

### Full resource browse

Basically this means, that the service is capable (and willing) of serving the whole Resource either in one, or by fragments/snippets, allowing paging.

It may be the original digital objects: Images/Facsimile, AV-Files, but it can also be the digitized text (in XML, HTML) or the annotation files (EAF, CHA …).

Figure 6 The online-edition of Fackel[[30]](#footnote-30) allows browsing through all pages of the original available as digital texts and as facsimile images.

C:\Users\m\3lingua\clarin\FCS\resources\screenFackelnavigation.png

### Lists/ Word Summaries

These are not primary resources, but rather derived information usually aggregations / frequency lists. This is similar to the scan-operation, or in other words: the result of a scan-operation is also such a list.

[key (token|annotation) + number]

Haus 45

Liebe 60

ADJA 145.320

NN 3.039.828

VFIN 1.920.101

These lists may contain links that allow digging deeper, often resolving to another data-type. For example the entries in the above list could link to another list of words of given PoS (all adjectives), or to a KWIC-result listing the occurrences.

Also to mention here nested or grouped lists (aka lists of lists aka trees), main example being probably the word profiles (aka collocational profiles aka word sketches). They are special in that they describe pairs (lemma + key), usually for one lemma (i.e. one result lists all collocational partners for one lemma).

lemma + [group   
 [key(=collocate) + number + link?]  
 ]

Snippet of a wordsketch provided by the sketchengine[[31]](#footnote-31)

<wordsketch corpname="gtbrg1.cpd">

<keyword pos="" freq="3389">Platz</keyword>

<gramrel>

<grname hits="816" score="2.2">AdjY SubstX</grname>

<collo hits="58" score="9.19" query="w3029822">frei</collo>

<collo hits="20" score="8.51" query="w3029857">öffentlich</collo>

<collo hits="9" score="8.04" query="w3029783">fünft</collo>

...

### Parallel sequences / aligned tiers

Both text corpora and multimodal resource potentially serve resources with aligned tiers. There are various encoding formats (EAF ([Appendix G.5), TCF (Appendix G.6](#_Annotation_file_EAF-format)), idiosyncratic solutions of individual search engines).

Perhaps a related but special case are **Parallel Corpora** in the sense, that there are two “primary sequences” being aligned.

### Syntax Tree

The output of TreeBanks, parsed sentences. Formats: GrAF, SynAF?, TCF?

### Geolocation

Some services may wish to express geographic locations as result. (See Appendix B.4 Nederlandse Familienamenbank).

### MDRecord

All resource within CLARIN shall be described by a MDRecord in CMDI-format.

### Multiviews

The services may be able to return different views, i.e. different formats of the result. As trivial example they may return the video-file and the corresponding annotation-file.

## searchRetrieveResponse

The response of the searchRetrieve operation defines only the envelope, allowing any internal record structure, typed record-wise by an arbitrary schema (<recordSchema> element):

<record>  
  <recordSchema>info:srw/schema/1/dc-v1.1</recordSchema>  
  <recordPacking>xml</recordPacking>  
  <recordData>  
    <srw\_dc:dc xmlns:srw\_dc="info:srw/schema/1/dc-v1.1">  
     <dc:title>The bicycle in its natural environment</dc:title>  
    </srw\_dc:dc>  
  </recordData>  
  <recordPosition>1</recordPosition>  
  <extraRecordData>  
    <rel:score xmlns:rel="info:srw/extensions/2/rel-1.0">  
      0.965  
    </rel:rank>  
   </extraRecordData>  
</record>

We want to continue the harmonization by finding common ground on the record level, ie proposing structures for individual records specific for different types of data.

### New Elements: Resource, ResourceFragment, DataView

Starting from the element recordData we propose three basic (still generic) extension elements:

Resource

element representing a resource, carrying the identifier.

It may represent anything that has a PID (and a MDRecord).

So in particular it may also be collections, aggregating other Resources.

Allowed children are: Resource, ResourceFragment and DataView

ResourceFragment / ResourcePart

A part of a resource, without own PID, i.e. something addressable with: PID of the Resource + Part/Fragment Identifier.

Fragment Identifier to be used depends on the resource type, it may be: XPointer, timecode, sequence-offset, etc.

Allowed children are: DataView

DataView

the element carrying the typed data  
Content can be anything that is in other namespace.  
The content has to be possible inline or referenced. Important for Images and AV-Files.  
(see 5.2.3 DataView - Handling the data types)

An example: two records both with two different data-types, first record returning a ResourceFragment, the second a whole Resource:

<sru:record>

<recordSchema>info:srw/schema/1/ccs-v1.0</recordSchema>

<sru:recordData xmlns:ccs="http://clarin.eu/ContentSearch" >

<ccs:Resource pid="123">

<ccs:ResourceFragment pid="123#a">

<ccs:DataView type="text/xml"><meertens:any/>

</ccs:DataView>

<ccs:DataView type="image/jpeg" ref=””></ccs:DataView>

</ccs:ResourceFragment>

</ccs:Resource>

</sru:recordData>

</sru:record>

<sru:record>

<sru:recordData>

<ccs:Resource pid="124">

<ccs:DataView type="text/x-eaf+xml">

<ANNOTATION\_DOCUMENT xsi:noNamespaceSchemaLocation=   
 ”<http://www.mpi.nl/tools/elan/EAFv2.2.xsd>” >   
 ...</ANNOTATION\_DOCUMENT>

</ccs:DataView>

<ccs:DataView type="video/mp4" ref=””></ccs:DataView>

</ccs:Resource>

</sru:recordData>

</sru:record>

### Identifying the Resources or Resource Parts/Fragments

This approach should also be able to cope with the issue of granularity, i.e. the fact that every repository can decide to structure their collections differently. For example only exposing the collection and an endpoint to that, individual resources not being addressable at all.

Accordingly in the result you could have:

<ccs:Resource pid="123"> /\* parent collection \*/

<ccs:Resource pid="123.1"> /\* the resource \*/

<ccs:ResourceFragment pid="123.1#a"> /\* part of the resource \*/

If the collection is represented in the result, we may need more information about it, then just the PID. Usually the only further information about collections are the MDrecords. And while the client could fetch these for every collection, there will hopefully be more efficient ways (For example in a combined metadata content search, we should have the MDRecords for all the resources already as result of the first – metadata search - phase.)

See the ISO/FDIS 24619 – PISA[[32]](#footnote-32) for detailed discussion of referencing Resources and parts of Resources, granularity issue and also about the distinction between ResourcePart and ResourceFragment.

In simple terms the difference is, that with ResourceFragment the client fetches the whole resource and navigates to or extracts the part of Resource being refered to (the well established URL “#” anchor-mechanism), whereas with ResourcePart the server is able to extract the part of a resource based on a parameter and sends only the requested snippet to the client. This is of course important distinction wrt performance and network-load especially for multimedia content, but we probably need both (as not all server may be capable of providing parts).

### DataView - Handling the data types

As we listed in previous chapter (5.1 Data Model), we have to deal with a broad range of data types, now we want to see, how we can fit them within the proposed structure (Resource, ResourceFragment, DataView).

We can distinguish three situations:

textual/XML datatype, existing schema

CMD, EAF, TCF, syntax trees, Geolocation(?)

These could be simply transported inline embedded or referenced in the defined envelope.

textual/XML datatype, no schema:

concordances/KWIC, Lists / Word summaries

Here we need to find consensus about the actual xml-structures.

binary resource

AV-Files, Images

These have to be referencable.

We need to define a mechanism to reference the actual content instead of inline embedding, i.e. something like a @link-, @url-, @ref- or even @pid-attribute on the DataView element. This is necessary for the binary resources, but may be equally usable for text/xml data-type as well.

<ccs:Resource pid="123.1"> /\* resource \*/

<ccs:DataView type=" text/x-eaf+xml ">

<ANNOTATION\_DOCUMENT> ... /\* annotation file\*/

</ccs:DataView>

<ccs:DataView type="video/mp4" ref=”{someURL/handle?}“>

</ccs:DataView>

So in IMDI we could use PIDs (see above comment), as every format has its own PID. But this most probably does not apply to all repositories.

Still another question is how to request one (or more) specific datatype(s). SRU proposes following parameters[[33]](#footnote-33) in the searchRetrieve-request:

recordPacking

currently just determining how the record should be escaped: string or xml

recordSchema

The schema in which the records MUST be returned.

This sounds like the nearest to what we need. Probably insufficient, if we will want to specify multiple (possibly alternative) schemas, or non-XML data types (binary resources)

stylesheet

identified by URL. Just to be returned with the response as PI.

extraRequestData

the generic extension mechanism, allowing us to define new parameters, if really needed.

### Schema for Lists

As identified earlier we need to be able to represent also list-, tree-, table- structures in general. The nearest SRU is offering here, are the term lists in the scanResponse, but we shouldn’t force ourselves into, which doesn’t really suit our needs. Still we need some starting point.

### Metadata view on Content

MDService (and also VLO-FacetedSearch for that matter) also has to deal with Resources. Every MDRecord has the <ResourceProxy> pointing to the resource described in the MDRecord. And the user shall be able to access the Resource (subject to authorization) as easy as possible. The trivial way to achieve this is to simply click/open the link/handle. But we may wish more integrated ways of activation.

Snipppet of a CMDI MDRecord:

<CMD>

<Header>...</Header>  
 <Resources>  
 <ResourceProxyList>  
 <ResourceProxy id=”{record-internal id for the proxy}” >  
 <ResourceType>Resource</ResourceType>

<ResourceRef>{PID handle}</ResourceRef>

</ResourceProxy>

...

</Resources>

<Components>

<someCMDprofile>

...

A realistic use case would be, that the user wants the result of a content query enriched with information about the context, i.e. with Metadata of the Resource and the Collection it came from. Currently the MDRecord (root element <CMD>) is delivered (roughly) as the recordData of the SRU-response by the MDService. In the case of such a combined result for uniformity reasons we could think of putting the CMDI-record into the DataView, like this:

<ccs:Resource pid="123"> /\* parent collection \*/

<ccs:DataView type="text/cmdi+xml">

<CMD>.../\* MDRecord for the collection \*/ </CMD>

</ccs:DataView>

<ccs:Resource pid="123.1"> /\* resource \*/

<ccs:DataView type="text/cmdi+xml">

<CMD>.../\* MDRecord for the resource \*/ </CMD>

</ccs:DataView>

<ccs:DataView type="text/x-eaf+xml ">

<ANNOTATION\_DOCUMENT> ... /\* annotation file\*/

</ccs:DataView>

## Multi-server response

In the federated search scenario, the composite service Federated Search shall dispatch the requests to the list of target repositories and return the collected results back to the requesting side.

The question is how to package the results, how to structure the aggregated result.

One general requirement should be to be able to work with partial results, together with the need for the composition of the result be transparent, i.e. the requesting side be able to reconstruct the individual sources of the result. In practice that means that the full response from every server (especially the header) would have to be passed.

This seems to push a lot of logic wrt to handling of this multi-result to the client, but it seems imperative for a client of federated search to be aware of the individual sources and be able to pass this information to the user.

On the other hand Federated Search should also offer the option to deliver the full composed result merged as one piece of data and to provide a full summary for the whole result. Where merged could mean a number of things: be it just sorted according to any index/field across all results, or even possibly the need to deduplicate and of course paging (if the client requests only first 10 records, how to decide which records, from which server to take?).

For a nice (client-side) handling of the multi-server processing see pazpar2[[34]](#footnote-34)

This would imply that the response cannot be nested as in:

<sru:records>

<sru:record>...

<sru:recordData xmlns:ccs="http://clarin.eu/ContentSearch" >

<sru:searchRetrieveReponse>

response from first server

</sru:searchRetrieveReponse>

</sru:recordData>

</sru:record>

<sru:record>...

<sru:recordData xmlns:ccs="http://clarin.eu/ContentSearch" >

<sru:searchRetrieveReponse>

response from second server

</sru:searchRetrieveReponse>

</sru:recordData>

</sru:record>

Even if that would be valid SRU-format.

One solution could be the composite response consisting of pointers to the partial results, the composite result being a regular searchRetrieveResponse, where the partial results are individual records represented by element <ccs:ResultSet>:

<sru:searchRetrieveResponse>

<sru:resultSetId>{composite-response-id}</sru:resultSetId>

<sru:records>

<sru:record>

<sru:recordData xmlns:ccs="http://clarin.eu/ContentSearch" >

<ccs:ResultSet ref=”{fragment\_identifier}” status=”finished” />

</sru:recordData>

<sru:recordData xmlns:ccs="http://clarin.eu/ContentSearch" >

<ccs:ResultSet status=”processing” />

</sru:recordData>

The composite result could serve as a complex/structured status update. The interaction would be as follows:

1. Upon search request the user receives the first (surrogate) response immediately.
2. This response carries   
   a) the resultSetId, which can be used as a ticket, so that the client can call back and  
   b) the list of partial results (one for every target-repository) .
3. The client can then periodically request the status-information of the result, by passing the resulSetId. The response will be basically always the same, however with the status of the partial results and the overall-summary information updated.
4. When one partial result gets ready, it gets a ref-attribute and the client can fetch it (as separate searchRetrieveResponse!).

This would cater for a robust system, that can handle different response times gracefully and would suit itself well for “semi-asynchronous” interaction, ie the user more-or-less waiting for the result, being informed about the progress and receiving partial results without further delay.

This is especially important for complex workflows, in particular involving computational steps (aggregation) over large resultset needed for advanced visualizations.

Another special issue are the responses of combined metadata/content search. The simplest solution seems to be to handle the metadata response the same way as the other responses as a partial result.

# Architecture

In this chapter some considerations regarding the architecture, focus on the main components of the Federated Search, but we may want to mention also other related components here.

## Federated Search Proper

The federated search, ie search over multiple search engines is the actual core issue of this paper.

The three parts to this module/component are

composite service - aggregator

dispatching the request to individual search services and merging the incoming partial results. Ths service has to a) expose/announce the list of searchable repositories and b) accept the selection on a separate parameter.

The interaction is then as follows:

1. user (or foreign application) selects the corpora that they want to search, based on the list provided and submits the query together with this selection

2. the service iterates through the list and queries individual candidate target services

3. it returns the aggregated/merged result, either

a) as the complete result, ie waiting till the last target service answered or timed out , or

b) as temporary status-result (available under a ticket) informing about the progress of the search at individual target services and either incorporating or linking to partial results.

(see 5.3 Multi-server response for detailed discussion of the alternatives for response format)

mapping

between the common protocol and the idiosyncratic interfaces of the individual search services.

We assume, that the mapping will be usually a wrapper around the existing target service, but should be seen as a separate (transformer) component and could be implemented either on the side of the individual search engine or the federating component or as a real separate service.

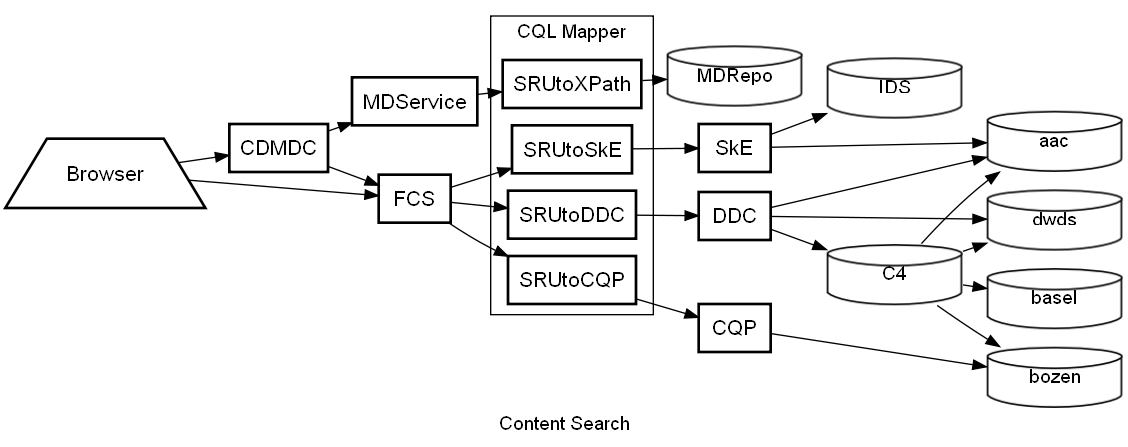
discovery service

a way to announce available repositories. This could either be provided by the composite service itself (in a kind of explain-operation), or this could be a separate service (a centers registry), being used by the composite service. This is in general a registry as it is already started at <http://www.clarin.eu/centres> but with more formalized output. (See more under 4.5.1 Announcing Repositories / Collections and Appendix A Repository List)

Figure 7 sketches a tentative setup for a distributed system of german text corpora rooted in the EDC/C4W democase. In the left part the whole communication is expected in SRU/CQL only.

The common components are the Federated Content Search, Combined Metadata Content Search, CMDI-MDService. The browser represents any user interface - SRU/CQL-aware web application, building on these services.

Figure 7 A schematic view of distributed search system, on the example of a few german corpora



## Potential Supporting Services

Data Category Registry, DCR

Primarily isocat[[35]](#footnote-35), defines data categories needed for harmonization of the indices and vocabularies

Virtual Collection Registry, VCR[[36]](#footnote-36)

this CMDI-component allows to define (both extensional and intensional) Virtual Collections. This seems to suit for the need of defining context for the search, i.e. defining which Resources shall be searched.

Vocabulary Alignment Service

This component is only in discussion, but its need was observed on multiple occasions.

It can be seen as a service under (or related to) DCR, a “soft registry” providing lists of possible (recommended) values for data categories, of which domain can’t be closed, but neither it is completely open – in the IMDI tradition we speak of “open vocabularies” - the canonical example being the Organization names.

The service shall provide lists of existing names and their aliases, allowing various application to provide the user pick-lists, when editing fields in metadata records, and similar.

The service itself shall be kind of patch-panel aggregating the information from existing vocabulary services like, e.g. the EC-organisation database and probably also manage the linking to the data categories…

## Combined Metadata Content Search

Following has been described within the EDC discussion. The proposal foresees two separate phases of the combined metadata content search:

metadata search – restricts the candidate collections to search in, based on metadata part of the query.   
It returns a list of candidates, which is used in:

content search – by the federated content search, to iterate through and issue the content query to each candidate in turn.

As was stated before (see 4.6 Combined Metadata and Content Query), it is not easy to distinguish clearly between the metadata and content part of a query, thus in both phases the protocol could encourage to send whole query, leaving it up to the target service to make the best out of it (i.e. utilize whatever parts of the query it can interpret). Nevertheless the composite service should first try to execute the metadata search to retrieve the candidate target resources (or collections).

This scenario requires a clear definition of how the metadata records are linked with the (endpoints of the) repositories/search engines.

A basic solution is that the ResourceRef of the Collection MDRecord points to the endpoint of given search engine, but this seems to simplistic. We have to distinguish at least three entities (see also discussion in 4.5 Restricting Search by Repositories/Corpora/Collections):

Collection

represented by the MDRecord, which describes the collection (originating Project, Editors etc.)  
links to member Resources (or sub-collections)

Resource

represented by own MDRecord, refers to its parent collection.

However not every Resource needs to have own MDRecord -> Issue of granularity

Search Service

the active component allowing to search in given collection.

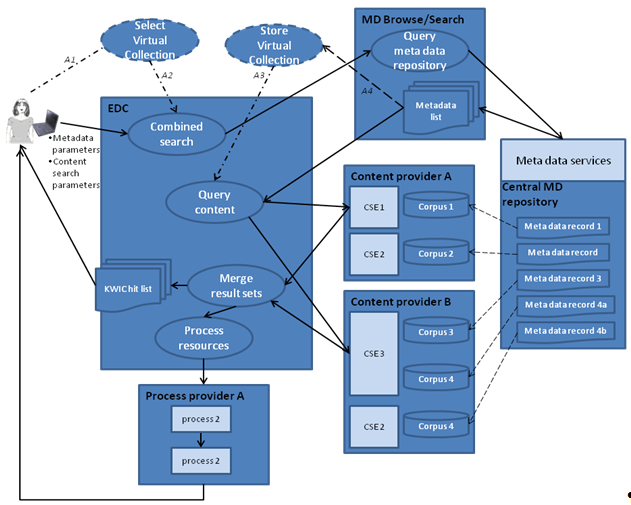
There should be a MDRecord for this entity containing the Technical Metadata Component, describing the technical details of the interface.

It will not necessarily be able to deliver the Resource itself. Due to IPR-issues, it may return only resource fragments – small snippets of the Resource in a KWIC-mode (as is the usual in text-corpora)

So the requirement for the metadata phase is to deliver not only the matching resource, but if the resource is not accessible directly, to also traverse the collection hierarchy up to a searchable/accessible collection, ie one that points to a search service.

It is not clear yet which software component would/shall have the intelligence/logic to match the information in the metadata record, with the corresponding information in the annotation file (Actor components to the appropriate Participants) or even how to do it at all.

Figure 8 A schematic view on a combined metadata content search scenario



## Distributed User Management – Federated AAI

In a distributed environment as envisaged, we will very soon have to handle the authentification and authorization. Especially as there are restriction (licensing) accessing a substantial part of the resources in question. Albeit this issue is being discussed in separate track within CLARIN and in tight cooperation with concerned players (eduGAIN, GEANT, …), we should sketch the specifics of this problem in the context of federated search:

In the default scenario, we expect an user interacting with some web application, so authentification (via **Shibboleth** at home organization) should be applicable here. However this web-application acting as an aggregator, will try to access multiple Content Providers with this user’s request, where it would need to identify itself with user’s authority (**delegation** of authentification).

This configuration gets one step more complicated, when considering the federated search **as a service** being invoked by some third application (e.g. workflow-engine), so the service itself would have to be equipped with delegated identity.

Even before such requests can technically happen and be handled, the Content and Service Providers have to agree upon **harmonizing licensing schemes** on the orgnizational/contractual level.

# References

The documents of the OASIS - Search Web Services TC:  
(at <http://www.oasis-open.org/committees/documents.php?wg_abbrev=search-ws>)

[OASIS-APD] 2010-09-17

[OASIS-SRU2] version

<http://www.dlib.org/dlib/january09/denenberg/01denenberg.html>

Appendices

1. Repository List
   1. ZeeRex format

Format: ZeeRex F&N <http://explain.z3950.org/overview/#3>

<explain>

<serverInfo>

<host>gondolin.hist.liv.ac.uk</host>

<port>210</port>

<database>l5r</database>

</serverInfo>

</explain>

* 1. List of candidate centers / search services

Combine with the centers-list <http://clarin.eu/centres>

Center/Repository, Search engine, Protocol

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Center*** | ***Status*** | ***MD*** | ***Content*** | ***Search engine*** |
| MPI Nijmegen |  | IMDI/CMDI searchable  In CLARIN Repository | Sessions, multimodal | ANNEX/TROVA,  MD Search |
| Meertens |  |  | Various Databases[[37]](#footnote-37) |  |
| INL |  |  |  |  |
| AAC/ICLTT |  | CMDI-teiHeader | Diachronic corpus , de | DDC |
| DWDS |  |  | text corpus, de | DDC |
|  |  |  |  |  |
|  |  |  |  |  |

* 1. Feature Matrix

Besides the conformance levels defined for CQL, there are also other conformance requirements on the client and server applications stated in the SRU 2.0 draft (chapter 14).

We want a compmrehensive list of features, which could be used by the services to announce their capabilities and for clients to search for services with required features.

|  |  |  |
| --- | --- | --- |
| CQL |  |  |
| Conformance level 0 |  |  |
| Conformance level 1 |  |  |
| Conformance level 2 |  |  |
| dynamic Indices (cmd., ccs.) |  |  |
| Sequential Tier Search |  |  |
| Explain operation |  |  |
| Scan operation (on which indices) |  |  |
| Serving Metadata |  |  |
| Serving original resource (subject to accessibility) |  |  |
| Full-text Browse |  |  |
| Facsimile |  |  |
| AV-resource |  |  |
| Serving Resource Fragments |  |  |
| KWIC |  |  |
| AV-snippet |  |  |
| Service associated Resources |  |  |
| Annotations |  |  |
| Result Formats |  |  |
| Authentication |  |  |

1. Candidate Search Engines

We may not need a description of all potential search engines, but we should try to find representatives for every type and go through the “binding” SRU <-> given Service exemplary.

* 1. CLARIN MDService

The CMDI component to search in the MD records collected in the Joint MD Repository

<http://clarin.aac.ac.at/MDService2/>

Information about binding to SRU: <http://clarin.aac.ac.at/MDService2/static/CMDRSB_20110123.pdf>

* 1. DDC

<http://www.ddc-concordance.org/>

corpus search engine used at DWDS Berlin, Basel, Bozen, Wien (C4 Project)

<http://chtk.unibas.ch/korpus-c4/search>

* 1. MPI Tools: ANNEX/Trova/ELAN

Annotation search and viewer

<http://www.lat-mpi.eu/tools/annex/manual/ch03.html>

<http://corpus1.mpi.nl/ds/annex/search.jsp?nodeid=MPI76418%23&row=37>

Figure 9 The multi-tier search in Trova

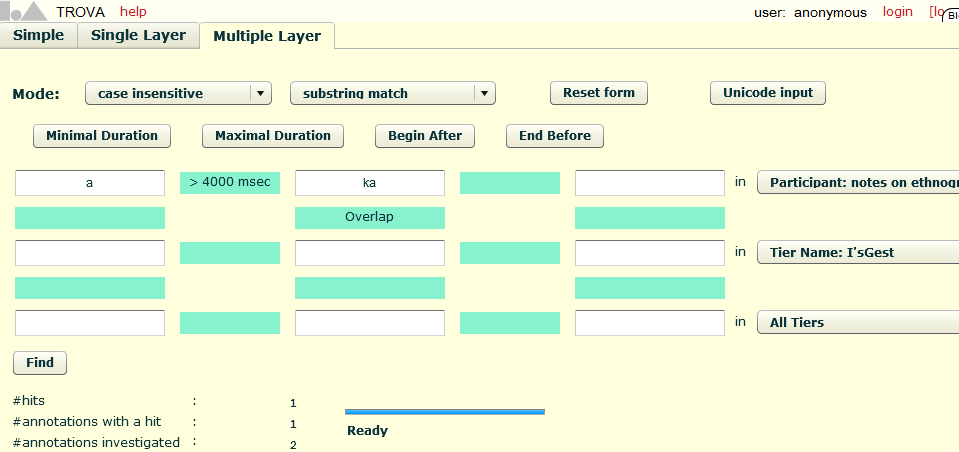
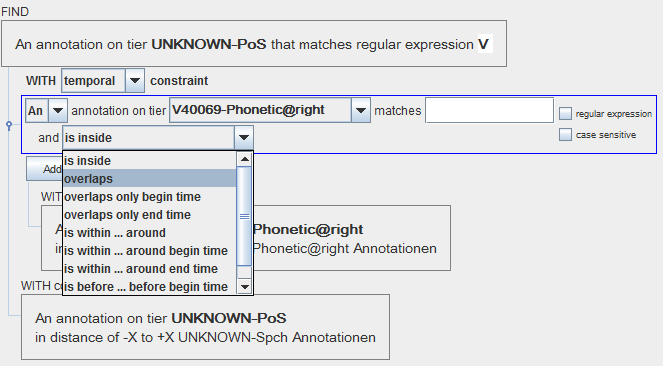


Figure 10 ELAN search interface with proximity modifiers



* 1. Nederlandse Familienamenbank

<http://www.meertens.knaw.nl/nfb/>

1. CQL Examples
   1. Metadata Queries

basic search clause

dc.title adj "open access"

boolean

Organisation any University

and (dc.language=de or cmd.Country=Austria)

and (dc.title any Liebe or cmd.Author any Trakl)

* 1. Content Queries
  2. Sequential Tier Search
  3. Metadata Content Queries

1. Proposed Extensions
   1. dynamic Indices

index ::= ['cmd.']cmdIndex | ['ccs.']contentIndex

cmdIndex ::= cmdIndex '.' cmdComponent | cmdComponent

cmdComponent ::= {componentName} | {componentID}

contentIndex ::= wordLevelIndex | annotationIndex

wordLevelIndex ::= 'word' | 'w' | 'pos' | 'p' | 'lemma'  
| 'l'| 'thesaurus' | 't' | {...}

annotationIndex ::= annotationPath ['.' annotationAttr]

annotationPath ::= annotationPath '.' {annotationElement} |   
{annotationElement}

Examples:

cmd.Session.Actor.Name

Collection.Project.Title

ccs.TierType.PoS

ccs.TierName.V40069-Lemma

isocat.PoS

* 1. Context Set: CMDI - Component Metadata Infrastructure
  2. Context Set: CCS - CLARIN Content Search
  3. New Boolean Operator: IN
  4. CCS response Schema: ResultSet, Resource, ResourceFragment, DataView

1. Mapping to other query languages
   1. SRU -> XPath

Actually it is translating to the cmd-dialect of XPath operating on CMDI.

More live examples under: <http://clarin.aac.ac.at/MDService2/docs/htmlpage/queries>

! Needs to be checked. May not be up2date!

|  |  |
| --- | --- |
| simple search : {term} | //\*[ft:query(.,{term})] |
| Peter | //\*[ft:query(.,'Peter')] |
| {cmdComponent} | //{cmdComponent} |
| Actor | //Actor |
|  |  |
| searchClause:= {cmdIndex} {rel} {term} | //{cmdIndex}[\. {rel} '{term}'] |
| Actors.Actor.Sex=f | //Actors/Actor/Sex[.='f'] |
| {cmdIndex} any {term} | //{cmdIndex}[contains(. '{term}')] |
| Organisation.Name any University | //Organisation/Name[contains(.,'University')] |
| AND | //CMD[.//{Q1}][.//{Q2}] |
| Organisation.Name = University and Actor.gender=m | //CMD [.//Organisation/Name [contains(.,'University')]] [.//Actor.gender='m'] |
| AND NOT | //CMD[.//{Q1}][not(.//{Q2})] |
| Organisation.Name any University and\_not Actor.gender=m | //CMD [.//Organisation/Name [contains(.,'University')]] [not(.//Actor.gender='m')] |
| OR | //CMD[.//{Q1} or .//{Q2}] |
| Organisation.Name any University or Actor.gender=m | //CMD[.//Organisation/Name[contains(.,'University') or .//Actor.gender='m'] |
| query expansion ([SemanticMapping](http://trac.clarin.eu/wiki/SemanticMapping)):= {datcat} {rel} {term} | //({cmdIndex1}|{cmdIndex2}|{cmdIndexN})[\. {rel} '{term}'] |
| dc:title any Peter | //(olac-title | teiHeader//titleStmt/title | teiHeader//monogr/title )[contains(.,'Peter')] |
| term | // |
|  | / |

* 1. SRU -> DDC

This is taken from trac-wiki/QueryLanguage and is only tentative (not implemented/tested yet)

|  |  |  |
| --- | --- | --- |
| *description* | *SRU-CQL* | *DDC* |
| **word-level:** |  |  |
| any word-form | [ccs].w |  |
| just that word-form | [ccs].w={word-form} | @{word-form}  $w={word-form} |
| lemma | [ccs].l={lemma}  [ccs].lemma={lemma} | $l={lemma}  %{lemma} |
| pos-tag | [ccs].pos={pos}  [ccs].p={pos} | $p={pos}  [{pos}] |
| morphological features | [ccs].morph={list of morph features} | [{list of morph features}] |
| thesaurus | [ccs].thes={superconcept} | { {list of morph features} } |
| multiple criteria | {index1} =/var=X {term1} and {index2} =/var=X {term2} | {index1}={term1} with {index2}={term2} |
| **patterns:** |  |  |
| word starts with | [ccs].w = {word-start}\*  [ccs].w = ^{word-start}  [ccs].w =^ {word-start} | {word-start}\* |
| word ends with | [ccs].w = \*{word-end}  [ccs].w = {word-end}^  [ccs].w ^= {word-end} | \*{word-end} |
| contains | [ccs].w any {word-part}  [ccs].w = \*{word-part}\* | /.\*{word-part}.\*/ |
| **boolean-operators:** |  |  |
| and | and, &, && | && |
| or | or, |, || | || |
| and not | not, ! | ! |  |
| **distance-operators:** |  |  |
| exact sequence, phrase | "{phrase}" | "{phrase}" |
| maximum distance ordered | prox/unit=word/distance < {max-distance}  prox/w/<{max-distance} | #{max-distance} |
| exact distance ordered | prox/unit=word/distance = {distance}  prox/w/{distance} | *?:* NEAR({Q1};{Q2};{distance}) && ! NEAR({Q1};{Q2};{distance - 1}) |
| maximum distance unordered | prox/unit=word/bidirectional/distance = {distance}  prox/w/bi/>={distance} | NEAR({Q1};{Q2};{max-distance}) |
| window | ? see [#OpenIssues](http://trac.clarin.eu/wiki/QueryLanguage#OpenIssues) | near({Q1};{Q2};{Q3};{max-distance}) |
| term within annotation | ccs.{annotationIndex} any {term} | {term} #within {annotationIndex} |
| **bibliographic-metadata:** |  |  |
| bib-field | {index} any {term} | #has\_field[{index},{term}] |
| date-range | dc.date < {date\_from} and dc.date > {date\_to} | #less\_by\_date[{date\_from}, {date\_to}] |
| **further query options:** |  |  |
| case-sensitive | ? | {corpus option} |
| sort clause | {whole-query} sortBy {index-list} | #greater\_by[{bib-field}]  #less\_by[{bib-field}] |
| restrict to subcorpus | ? | {query} :{defined-subcorpus-list,} |

* 1. SRU -> CQP
  2. SRU -> manatee
  3. Other potential protocols / query languages

There are other existing proposals/protocols for search services which should be at least kept notice of.

OpenSearch

<http://www.opensearch.org/>   
interesting article comparing OpenSearch and SRU/SRW (2010-07): <http://dlib.org/dlib/july10/hammond/07hammond.html>

As one of the 7 documents the Search WS TC actually already provides a binding of the APD to OpenSearch (2008-06-30)

Lucene

an Apache project: pure Java scalable full-text search engine, operates on a flat model of documents having fields; underlying Apache solr (used for the second version of CLARIN’s VLO-FacetedSearch)

Query syntax: <http://lucene.apache.org/java/3_0_0/queryparsersyntax.html>

YQL

Yahoo Query Language  
<http://developer.yahoo.com/yql/>

1. From Repository to ResourceFragment View

Repository

Collection

Subcollection

Resource

Fragment

1. Related Formats

we should provide the xml at least here with syntax highlighting.

links to schemas and docs

* 1. SRU: searchRetrieve()
  2. SRU: scan()
  3. ZeeRex explain record

<sru:explainResponse xmlns:sru="http://www.loc.gov/zing/srw/">

...

<zr:explain xmlns:zr="http://explain.z3950.org/dtd/2.1/">

<zr:serverInfo protocol="SRU" version="1.2" transport="http"

method="GET POST SOAP">

<zr:host>myserver.com</zr:host>

<zr:port>80</zr:port>

<zr:database>cgi/mysru</zr:database>

</zr:serverInfo>

<zr:databaseInfo>

<title lang="en" primary="true">SRU Test Database</title>

</zr:databaseInfo>

<zr:indexInfo>

<zr:set name="dc" identifier="info:srw/cql-context-set/1/dc-v1.1"/>

<zr:index>

<zr:map><zr:name set="dc">title</zr:name></zr:map>

</zr:index>

</zr:indexInfo>

<zr:schemaInfo>

<zr:schema name="dc" identifier="info:srw/schema/1/dc-v1.1">

<zr:title>Simple Dublin Core</zr:title>

</zr:schema>

</zr:schemaInfo>

<zr:configInfo>

<zr:default type="numberOfRecords">1</zr:default>

<zr:setting type="maximumRecords">50</zr:setting>

<zr:supports type="proximity"/>

</zr:configInfo>

</zr:explain>

</sru:recordData>

</sru:record>

</sru:explainResponse>

* 1. CMD
  2. Annotation file EAF-format

Here you can find the corresponding IMDI-File:

<http://catalog.clarin.eu/ds/imdi_browser/viewcontroller?request=view&nodeid=MPI425714%23&row=9160>  
<http://corpus1.mpi.nl/CGN/COREX6/data/meta/imdi_3.0_eaf/sessions/fv400279.imdi>

An example annotation (EAF-format):

<ANNOTATION\_DOCUMENT AUTHOR="unspecified" DATE="2006-09-05T17:52:18+01:00" FORMAT="2.2" VERSION="2.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.mpi.nl/tools/elan/EAFv2.2.xsd">

<HEADER MEDIA\_FILE="" TIME\_UNITS="milliseconds">

<MEDIA\_DESCRIPTOR MEDIA\_URL="file:/data/corpora/CGN/COREX6/data/audio/wav/comp-h/vl/fv400279.wav" MIME\_TYPE="audio/x-wav"/>

</HEADER>

<TIME\_ORDER>

<TIME\_SLOT TIME\_SLOT\_ID="ts1" TIME\_VALUE="950"/>

<TIME\_SLOT TIME\_SLOT\_ID="ts2" TIME\_VALUE="1210"/>

<TIER DEFAULT\_LOCALE="nl" LINGUISTIC\_TYPE\_REF="Spch"

PARTICIPANT="V40069" TIER\_ID="V40069-Spch">

<ANNOTATION>

<ALIGNABLE\_ANNOTATION ANNOTATION\_ID="fv400279.1"

TIME\_SLOT\_REF1="ts1" TIME\_SLOT\_REF2="ts42">

<ANNOTATION\_VALUE>wat daar begonnen is op dat laatste avondmaal dat wordt nu nog altijd gedaan en wel ho over heel de wereld.</ANNOTATION\_VALUE>

</ALIGNABLE\_ANNOTATION>

</ANNOTATION>

<ANNOTATION>

…

</TIER>

<TIER DEFAULT\_LOCALE="nl" LINGUISTIC\_TYPE\_REF="Words" PARENT\_REF="V40069-Spch" PARTICIPANT="V40069" TIER\_ID="V40069-Words">

<ANNOTATION>

<ALIGNABLE\_ANNOTATION ANNOTATION\_ID="fv400279.1.1" TIME\_SLOT\_REF1="ts1" TIME\_SLOT\_REF2="ts2">

<ANNOTATION\_VALUE>wat</ANNOTATION\_VALUE>

</ALIGNABLE\_ANNOTATION>

</ANNOTATION>

<TIER DEFAULT\_LOCALE="nl" LINGUISTIC\_TYPE\_REF="PoS"

PARENT\_REF="V40069-Words" PARTICIPANT="V40069" TIER\_ID="V40069-PoS">

<ANNOTATION>

<REF\_ANNOTATION ANNOTATION\_ID="fv400279.1.1-pos"

ANNOTATION\_REF="fv400279.1.1">

<ANNOTATION\_VALUE>VNW(vb,pron,stan,vol,3o,ev)

</ANNOTATION\_VALUE>

</REF\_ANNOTATION>

</ANNOTATION>

….

* 1. TCF

ADD Examples of TCF!

1. Remarks on GUI, displaying/viewing

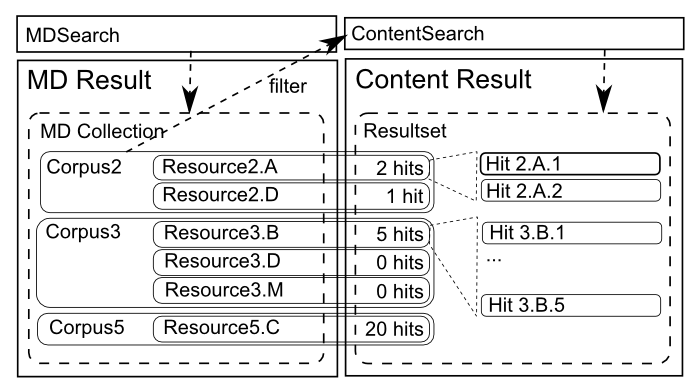
not really the topic of this paper, but related, thus some general consideration

Very probably this will be a separate paper

* 1. Search

Query Input

Result



* 1. Resource Viewer
* Generic Interface for a loadable type-specific UI-Component(!)
* Multiviews
* Interactivity

defined interaction mechanisms between the calling and called element.

Bound to platforms: js/jQuery, HTML5, Flash, Wicket, Java-applets

1. CLARIN Component Metadata Infrastructure <http://www.clarin.eu/cmdi> [↑](#footnote-ref-1)
2. for standard conformance status see s.9 in <http://clarin.aac.ac.at/MDService2/static/CMDRSB_20110123.pdf> [↑](#footnote-ref-2)
3. <http://www.oclc.org/uk/en/worldcat/default.htm> [↑](#footnote-ref-3)
4. <http://search.theeuropeanlibrary.org/portal/en/index.html> [↑](#footnote-ref-4)
5. <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=search-ws> [↑](#footnote-ref-5)
6. <http://www.loc.gov/standards/sru/oasis> [↑](#footnote-ref-6)
7. <http://www.loc.gov/standards/sru/index.html> [↑](#footnote-ref-7)
8. APD draft <http://www.loc.gov/standards/sru/oasis/current/apd.doc> version 2010-09-17 [↑](#footnote-ref-8)
9. <http://www.loc.gov/standards/sru/specs/cql.html#baseprofile> [↑](#footnote-ref-9)
10. <http://www.loc.gov/standards/sru/specs/diagnostics.html> [↑](#footnote-ref-10)
11. <http://www.loc.gov/standards/sru/resources/cql-context-set-v1-2.html> [↑](#footnote-ref-11)
12. <http://www.loc.gov/standards/sru/resources/sort-context-set.html> [↑](#footnote-ref-12)
13. CQL draft: <http://www.loc.gov/standards/sru/oasis/current/cql.doc> version: 2010-09-20 [↑](#footnote-ref-13)
14. <http://www.loc.gov/standards/sru/resources/cql-context-set-v1-2.html> [↑](#footnote-ref-14)
15. <http://www.loc.gov/standards/sru/specs/explain.html> [↑](#footnote-ref-15)
16. <http://explain.z3950.org/> [↑](#footnote-ref-16)
17. <http://www.loc.gov/standards/sru/oasis/current/sru-2-0.doc> [↑](#footnote-ref-17)
18. <http://www.loc.gov/standards/sru/specs/common.html#extraData>h [↑](#footnote-ref-18)
19. <http://www.loc.gov/standards/sru/resources/dc-context-set.html> [↑](#footnote-ref-19)
20. <http://clarin.aac.ac.at/MDService2/terms/htmlpage/?q=all&repository=2&maxdepth=8> [↑](#footnote-ref-20)
21. <http://www.lat-mpi.eu/tools/elan/> [↑](#footnote-ref-21)
22. Actor in CompReg: [http://catalog.clarin.eu/ds/ComponentRegistry/?item=clarin.eu:cr1:c\_1271859438158](http://catalog.clarin.eu/ds/ComponentRegistry/?item=clarin.eu:cr1:c_1271859438158#app=5f9c&7239-selectedIndex=1) [↑](#footnote-ref-22)
23. Bird, Libermann (2000): A Formal Framework for Linguistic Annotations <ftp://ftp.cis.upenn.edu/pub/sb/sact/bird/annotation.pdf> [↑](#footnote-ref-23)
24. <http://www.isocat.org/rest/dc/2462> [↑](#footnote-ref-24)
25. Discussed by Ray Denenberg in DLib article January 09 in the chapter: 4.2 Proximity: <http://www.dlib.org/dlib/january09/denenberg/01denenberg.html> [↑](#footnote-ref-25)
26. <http://explain.z3950.org/overview/#2.1> [↑](#footnote-ref-26)
27. <http://demo.spraakdata.gu.se/clarin/cmd/model/stats?operation=getCollections&collection=&maxdepth=3> [↑](#footnote-ref-27)
28. <http://clarin.aac.ac.at/MDService2/collections/xml?repository=2> [↑](#footnote-ref-28)
29. <http://clarin.aac.ac.at/MDService2/?q=&squery=&collection=oai:acl.sr.language-archives.org&columns=&startItem=1&maximumItems=10&repository=1>  
    (In the UI you can also switch to XML view, to see, that the result "is close to" a SRU-result.) [↑](#footnote-ref-29)
30. <http://corpus1.aac.ac.at/fackel/> [↑](#footnote-ref-30)
31. <http://www.sketchengine.co.uk/> [↑](#footnote-ref-31)
32. Language resource management - Persistent identification and sustainable access (PISA)  
    <http://www.iso.org/iso/catalogue_detail.htm?csnumber=37333> [↑](#footnote-ref-32)
33. <http://www.loc.gov/standards/sru/specs/search-retrieve.html#params> [↑](#footnote-ref-33)
34. <http://www.indexdata.com/pazpar2> [↑](#footnote-ref-34)
35. <http://www.isocat.org> [↑](#footnote-ref-35)
36. <http://clarin.ids-mannheim.de/vcr/app/public> [↑](#footnote-ref-36)
37. [htt//www.meertens.knaw.nl/cms/en/databases](http://www.meertens.knaw.nl/cms/en/databases) [↑](#footnote-ref-37)