Requirements Specification
Web Services and Workflow Systems

2010-01-12   Version: 2

Editors: Adam Funk, Nuria Bel, Santi Bel, Marco Büchler, Dan Cristea, Fabienne Fritzinger, Erhard Hinrichs, Marie Hinrichs, Radu Ion, Marc Kemps-Snijders, Yana Panchenko, Helmut Schmid, Peter Wittenburg, Uwe Quasthoff, Thomas Zastrow

www.clarin.eu
Requirement Specification Web Services and Workflow systems

CLARIN-EB-1/2010

EC FP7 project no. FP7-RI-2122230

Deliverable: D2.R6 - Deadline: T24

Responsible: Peter Wittenburg

Contributing Partners: MPI, UTuebingen, RACAI, UFSD, UPF, DFKI, U Wroclaw

Contributing Members: IDS, MPDL, U Vienna, U Stuttgart, DANS

© all rights reserved by MPI for Psycholinguistics on behalf of CLARIN
Scope of the Document

This document describes the goals and requirements of web services and workflow systems that could be used by all CLARIN members and beyond, i.e. a functioning system could be used by other communities as well. Stepwise all CLARIN centers would need to introduce these requirements in their operational environment to come to a proper landscape of resources, services and tools where various instances can and will be created/operated at various places.

This document will be discussed in the appropriate working groups and in the Executive Board. It will be subject of regular adaptations dependent on the progress in CLARIN.

Update Information

This version is an update of the first version presented in 2009. To simplify the comparison with the earlier version we mainly introduced a new chapter 7 called “State and Evaluation”. In this chapter we will refer to other chapters when we are updating essential information. For small corrections that do not modify basic assumptions and statements we have changed the text.

CLARIN References

- Centers Types  CLARIN-2008-1  February 2009
- Persistent and Unique Identifiers  CLARIN-2008-2  February 2009
- Centers  CLARIN-2008-3  February 2009
- Language Resource and Technology Federation  CLARIN-2008-4  February 2009
- Metadata Infrastructure for Language Resource and Technology  CLARIN-2008-5  February 2009
- Report on web services  CLARIN-2008-6  March 2009
- Integration LR into web service infrastructure  CLARIN-2009-D5R-3a  December 2009
Executive Summary

In this document we want to specify the requirements for the service oriented architecture which CLARIN is envisaging. This includes all aspects that have to with interoperability. While work package 5 will deal with these issues in detail from the linguistic point of view, work package 2 is focusing on the computer science aspects. Of course these two aspects are overlapping, therefore we expect that once work package 5 has come out with further requirements specifications and standards definitions a new version of this document will need to be written. The following topics are discussed in this document:

- the relevant web service standards and the experiences of the involved partners
  - SOAP and REST must be supported and are described using WSDL or WADL
  - CLARIN MetaData Initiative (CMDI) framework will be used to describe web services
- the workflow systems that have been suggested so far
  - CLARIN claims no preference to any workflow system
- the work that has already been done within the community with respect to web services and workflow systems
  - A number of ready to use web services are presented in this document
  - GATE, UIMA, JBPM and Taverna are actively used in the community
- the requirements that are relevant for the LRT community, in particular with respect to metadata, provenance information, authentication in distributed web services solutions, the import/export aspects which cover the structure/format and the encoding aspects and aspects that are related to IPR issues and business models
  - Requirements are listed in chapter 8
- an analysis of the current demo projects within CLARIN
  - A European demo project is envisaged that must span multiple collaborating partners. It was scheduled for the second phase.
  - An analysis of the progress revealed that a combined metadata/content search covering resources from various partners covers many aspects and can be realized as a prototype demonstrating the power of the CLARIN infrastructure.
- standards for interoperability
  - just recently another meeting about standardization in our field has taken place bringing together outstanding experts from different initiatives such as CLARIN, FLARENET, ELRA, TEI and ISO TC37 resulting in wide agreements;
  - WP5 will report on this and other meetings that are relevant to improve syntactic and semantic interoperability

Thus, the framework for working on the pillars of a service oriented infrastructure has been largely specified and all activities need to adhere to the requirements specifications which have been specified so far and which are referred to in this document. Still with respect to a number of technical and linguistic questions additional prototypes need to be worked out, adapted to the requirements and to be extended to cover the European dimension to be able to describe the requirements for a service oriented architecture in a more comprehensive manner.
1. Introduction

This document describes the state of discussion on web services and workflow which have taken place at various meetings and workshops in the CLARIN context. In general it can be stated that considerable progress has been made in this area but that it is less mature than for example the metadata field. While these meetings and workshops about web services and workflow systems had a harmonizing and abstracting aspect which can be seen as a top-down approach it has been clear from the start that the grounding of the work must take place at a concrete level, i.e. a bottom up approach. The motivation for this lies in two reasons: (1) practical work is needed to improve understanding of the problems in particular in the area of standardization of formats and the encoding of linguistic phenomena that need to be dealt with and (2) a group of young experts is needed that can bring together sufficient knowledge and expertise from the area of linguistics and IT who can participate in the work during the prototype and construction phase. The results of this document therefore represent a joint collaboration of the core CLARIN partners who are actively involved in this area.

Other activities that are of relevance to the web services and workflow working group are dealt with in other packages and working groups. Workshops and personal interactions also ensure that there is a continuous exchange of ideas, approaches and solutions, particularly in the areas of centers for storage and computation, metadata to describe tools and services, single sign-on principles, approaches for semantic interoperability, analysis of use cases and IPR and business models. Some of these areas have been settled in CLARIN and can be seen as constraints for the work in the web services and workflow area, others such as IPR issues are still subject to active debate and little concrete results can be drawn from these yet. They will however remain to be closely monitored and implications for the area of web services and workflow will be taken into account when results materialize.
2. Terminology

2.1. Definitions

**AAI** [Stanica 2006]
Authentication and Authorization infrastructure
An infrastructure that provides Authentication and Authorization Services. The minimum service components include Identity and Privilege Management with respect to users and resources.

**Archive** [CITER]
repository dedicated to the long-term preservation of the associated data

**IP** [Stanica 2006]
Identity provider
An entity in an AAI that performs Identity Management.

**Metadata** [Gunter 2004]
structured information that describes, explains, locates, and otherwise makes it easier to retrieve and use an information resource.

**Metadata registry** [Gunter 2004]
registry
a formal system for the documentation of the element sets, descriptions, semantics, and syntax of one or more metadata schemes

**Provenance data**
Information that provides a traceable record of the origin and source of a resource

**Resource** [Berners-Lee 2005]
The term "resource" is used in a general sense for whatever might be identified by a URI. Familiar examples include an electronic document, an image, a source of information with a consistent purpose (e.g., "today's weather report for Los Angeles"), a service (e.g., an HTTP-to-SMS gateway), and a collection of other resources. A resource is not necessarily accessible via the Internet; e.g., human beings, corporations, and bound books in a library can also be resources. Likewise, abstract concepts can be resources, such as the operators and operands of a mathematical equation, the types of a relationship (e.g., "parent" or "employee"), or numeric values (e.g., zero, one, and infinity).

**Repository** [CITER]
facility that provides reliable access to managed digital resources

**SOA** [Mackenzie 2006]
Service Oriented architecture
A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

**SP** [Stanica 2006]
Service provider
An entity that provides access to a service based on federated authentication.

**Web service** [Brown 2004]
A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format.
Workflow

Workflow is a term used to describe the tasks, procedural steps, organizations or people involved, required input and output information, and tools needed for each step in a business process.

2.2 Acronyms

<table>
<thead>
<tr>
<th>Reference</th>
<th>Abbreviation of</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CGN]</td>
<td>Corpus Gesproken Nederlands</td>
<td><a href="http://lands.let.kun.nl/cgn/">http://lands.let.kun.nl/cgn/</a></td>
</tr>
<tr>
<td>[CLARIN]</td>
<td>Common Language and Technology Infrastructure</td>
<td><a href="http://www.clarin.eu">http://www.clarin.eu</a></td>
</tr>
<tr>
<td>[DC]</td>
<td>Dublin Core</td>
<td><a href="http://dublincore.org/">http://dublincore.org/</a></td>
</tr>
<tr>
<td>[DCAM]</td>
<td>Data Category Registry</td>
<td><a href="http://dublincore.org/documents/abstract-model/">http://dublincore.org/documents/abstract-model/</a></td>
</tr>
<tr>
<td>[DEISA]</td>
<td>Distributed European Infrastructure for Supercomputing Applications</td>
<td><a href="http://www.deisa.eu/">http://www.deisa.eu/</a></td>
</tr>
<tr>
<td>[DFKITR]</td>
<td>Deutsche Forschungszentrum für Künstliche Intelligenz Tool Registry</td>
<td><a href="http://registry.dfki.de/">http://registry.dfki.de/</a></td>
</tr>
<tr>
<td>[DFN]</td>
<td>Deutsches Forschungsnetz</td>
<td><a href="http://www.dfn.de/">http://www.dfn.de/</a></td>
</tr>
<tr>
<td>[DS-SPIN]</td>
<td>Deutsche Sprachressourcen-Infrastruktur</td>
<td><a href="http://www.sfs.uni-tuebingen.de/dspin/">http://www.sfs.uni-tuebingen.de/dspin/</a></td>
</tr>
<tr>
<td>[EII]</td>
<td>European Grid Initiative</td>
<td><a href="http://web.eu-eii.eu/">http://web.eu-eii.eu/</a></td>
</tr>
<tr>
<td>[ESF]</td>
<td>European Science Foundation Second Learner Study</td>
<td><a href="http://books.google.de/books?id=s92tXMX4tpC&amp;pg=PA1&amp;lpg=PA1&amp;dq=esf+Second+learner+perdue&amp;sourc=bl&amp;ots=WKi3GUQQP6&amp;sig=n7QSWy3SIXvD06nMfA2Y7GBbm9w&amp;hl=de&amp;sa=X&amp;ei=book_result&amp;resnum=3&amp;ct=result#PPP1,M1">http://books.google.de/books?id=s92tXMX4tpC&amp;pg=PA1&amp;lpg=PA1&amp;dq=esf+Second+learner+perdue&amp;sourc=bl&amp;ots=WKi3GUQQP6&amp;sig=n7QSWy3SIXvD06nMfA2Y7GBbm9w&amp;hl=de&amp;sa=X&amp;ei=book_result&amp;resnum=3&amp;ct=result#PPP1,M1</a></td>
</tr>
<tr>
<td>[HS]</td>
<td>Handle System</td>
<td><a href="http://www.handle.net/">http://www.handle.net/</a></td>
</tr>
<tr>
<td>[INTERA]</td>
<td>Integrated European language data Repository Area</td>
<td><a href="http://www.mpi.nl/intera/">http://www.mpi.nl/intera/</a></td>
</tr>
<tr>
<td>Acronym</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>LAF</td>
<td>Linguistic Annotation Framework</td>
<td></td>
</tr>
<tr>
<td>LMF</td>
<td>Lexical Markup Framework</td>
<td></td>
</tr>
<tr>
<td>MAF</td>
<td>Morhosentactic Annotation Framework</td>
<td></td>
</tr>
<tr>
<td>MILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLSR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPLESAML</td>
<td>SimpleSAMLphp</td>
<td></td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
<td></td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
<td></td>
</tr>
<tr>
<td>WADL</td>
<td>Web Application Description Language</td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
<td></td>
</tr>
</tbody>
</table>
### 2.2 Related Documents

<table>
<thead>
<tr>
<th>Document ID</th>
<th>Title</th>
<th>Publication Date</th>
<th>Link</th>
</tr>
</thead>
</table>
3. Goals and activities

CLARIN is devoted to build a persistent integrated and interoperable infrastructure that will facilitates the access and the combination of language resources and tools/web services for those researchers that are working with language material in some form, in particular of course the humanities and social sciences.

To achieve this goal CLARIN started a number of different activities. At the integration level all technologies that allow users to make resources visible and to bring them together are in focus: federation technologies, persistent identifiers, metadata, registries, workspaces and portals. At the interoperability level we are faced with the problems of exploiting the virtually integrated resources and tools/services, i.e. typically issues such as structural and semantic interoperability need to be solved to allow users for example to build and execute short and long chains of operations which we will call workflow chains. All integration aspects which are largely independent of the internal structure and semantics of resources and tools have been dealt with already by other requirements specification documents and are currently being worked out in prototypical form. They are normative for all CLARIN activities.

This requirements specification document deals with the interoperability aspects as they occur in service oriented architectures. Here we want to give two examples for such scenarios:

- a structured search operating across a virtual collection of heterogeneously tagged and encoded annotations for example requires to have a mapping between the related tag and encoding sets;
- the creation of a workflow chain of operations requires that the input resource(s) need to provide the format and encoding that the following operation can work on.

In this document we want to specify the requirements for the service oriented architecture which CLARIN is envisaging. This includes all aspects that have to with interoperability. While work package 5 will deal with these issues in detail from the linguistic point of view, work package 2 is focusing on the computer science aspects. Of course these two aspects are overlapping, therefore we expect that once work package 5 has come out with further requirements specifications and standards definitions a new version of this document will need to be written. In general we can state that the following topics need to be discussed by this document:

- the relevant web service standards and the experiences of the involved partners
- the workflow systems that have been suggested so far
- the work that has already been done within the community with respect to web services and workflow systems
- the requirements that are relevant for the LRT community, in particular with respect to metadata, provenance information, authentication in distributed web services solutions, the import/export aspects which cover the structure/format and the encoding aspects and aspects that are related to IPR issues and business models
- an analysis of the current demo projects within CLARIN

This document needs to be specific where CLARIN has already taken decisions and needs to be flexible enough in those areas where further discussions are required. In particular the harmonization of structures has been raised and there is the strategic view that we need to come to pivot formats for various processing steps, but yet we cannot claim that this problem has been solved. Similarly the issue of semantic interoperability at the level of linguistic encodings (tag sets etc) has been introduced recently at a broad scale. A sub-community has already dealt with the issue of defining linguistic categories for some period of time already which resulted for example in the EAGLES recommendations [Calzolari and Monachin, 1996]. However, other communities are using other categories for reasons that partly have to do with the differences between languages. Only 4 years ago the new idea of an open registry for maintaining such categories was launched which resulted in the ISO 12620 standard and the ISOcat software and where CLARIN expects that many well-defined tag sets will be registered and defined properly. Other people can make use of them and create references from their own schemas making the ISO DCR to a kind of reference set of categories. However, this is all very new and we do not know yet whether this will bring us ahead in semantic interoperability as expected.
4. Current state web services and workflow systems

4.1 Web services standards

4.1.1 UDDI/ebXML

The Universal Description, Discovery and Integration [UDDI] project was advocated as a universal method for dynamic discovery and invocation of web services. It was initiated by Arriba, Microsoft and IBM in recognition of the need for a global registry for discovering web services. Microsoft, IBM and SAP all have operated public implementations of UDDI and Universal Business Registry that represents a master directory of publicly available e-commerce services. By the end of 2006 all of these had been shut down. Private and community UDDI nodes remain in operation.

UDDI conceptually consists of three types of sections where information is stored. White pages contain basic business details information such as name, address, and business identifiers such as tax id. This makes it possible to locate services using organizational characteristics. Yellow pages contain information on web services using taxonomic classifications. These classification criteria, such as 'annotation tools', may also be used to locate web services of interest. Green pages finally describe technical aspects of a web service such as location and service bindings. UDDI also provides as web service API for publishing, searching, retrieving and replicating this information.

Electronic business XML [ebXML] is a joint effort from United Nations/CEFACT and OASIS to create a single global XML framework solution. It is intended to facilitate trade by providing a specification that allows organizations to express their business processes in a manner that is understandable by other organizations thereby allowing process integration. The primary focus of ebXML is therefore on e-business. It describes a data model for e-business objects (including services), messaging for transactions and a registry for eBusiness objects.

Both UDDI and ebXML are maintained by OASIS (Organization for the Advancement of Structured Information Standards). However, it was reported in the web services workshop that both suggestions are not widely used in the research community. Furthermore, as they are defined right now, the direct use of these web services registries is in e-Business and e-Commerce.

For registration of web services in the final CLARIN registry it has become clear at the Munich workshop\(^1\) that neither UDDI or EbXML are inadequate to describe the web service characteristics for this domain. As a result it has been decided that all web services and tools will be described using the Clarin MetaData Initiative which provides a flexible metadata framework with references to well defined data categories in the DCR.

4.1.2 WSDL

Web Services Description Language [WSDL] is an XML format that is used to describe web service interfaces. It describes the operations (network end points or ports) and data formats (messages) in an abstract manner. This results in a set of reusable bindings which are subsequently bound to concrete network protocols and message formats. WSDL is often used in combination with SOAP (Simple Object Access Protocol) and XML schema to define web services over the web. WSDL version 2.0 can be used to describe both REST and SOAP web services. WSDL is maintained by W3C.

A WSDL file contains the XML description of each of the web service exposed functions. The description of one function contains the types of its arguments formalized with XML Schemas (types can be either simple or complex) and the type of the returned value. The description of the service contains the end-point (the physical address of the web server hosting the service) and the URI of the service which is a unique identifier with which the service is associated (the HTTP request is formed by issuing a POST or GET HTTP operation to the end-point asking for a function along with the URI). In the following example (taken from the WSDL 2.0 primer of W3C, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/#basic-example)

\(^1\) Munich, November 2008
The function (or method since it's part of an interface— the new way in which WSDL 2.0 is grouping a set of operations of a web service) `opCheckAvailability` has as parameters the XML Schema complex type `tCheckAvailability` that is composed of two dates and one string and that it returns one double real number.

What WSDL does not specify is a semantics of the operation(s) it describes. The user of a web service must know the significance of the result one function returns. For instance, a WSDL file describes two functions f1 and f2, both of which require two float numbers as arguments and are returning float. Now, let’s suppose that the first one is taking the sum of its arguments and the second one divides its arguments. If the user does not recognize the type of operation either by reading the function name or by reading the documentation the creator of the function has been kind to offer, he will never know what makes the f1 and f2 different.

One other underspecified element of WSDL is that the particular format of a return value is not known. In other words, the user of a web service must know in advance what form the input parameters must have (for instance if a function is expecting a string, that string has a specific format like the format of a date for instance and will not work with a different format) and what format the output of the function adopts. These issues are discussed in this document in what follows regarding the interoperability of different CLARIN processing components.

4.1.3 XML-RPC

XML-RPC is a remote procedure calling protocol that works over the internet. An XML-RPC message is a HTTP-POST request. The body of the request is in XML. A procedure is executed on a remote server and the response is also formatted in XML.[Winer 1999]. It was created in 1998 by Dave Winer of UserLand Software and Microsoft.[Box, 2001]. As new functionality was introduced, the standard evolved into what is now SOAP. XML/RPC can be preferred to SOAP because of its simplicity, minimalism and ease of use.
There is nothing in XML/RPC that WSDL/SOAP cannot handle. Thus, in the interest of completeness, CLARIN should support the XML/RPC paradigm of defining web services, especially because it seems simpler to write these web services in conjunction with the REST methodology (see the next sections).

### 4.1.4 SOAP
SOAP (Simple Object Access Protocol) is a XML-based communication protocol for accessing a web service, created to communicate over HTTP (which is today supported by all internet browsers and services). SOAP is platform and language independent, simple and extensible. SOAP may also be used over HTTPS (which is the same protocol as HTTP at the application level, but uses an encrypted transport protocol underneath) with either simple or mutual authentication.

A SOAP message is an XML document containing:
- an *Envelope* element that identifies the XML document as a SOAP message and constitutes the root element; it contains the *namespace* attribute (which defines the envelope as a SOAP envelope) and the *encodingStyle* attribute (defining the data types used in the document);
- an optional *Header* element, containing application-specific information (like authentication, payment, etc) about the SOAP message;
- a *Body* element, that contains the actual SOAP message (call and response information);
- an optional *Fault* element containing errors and status information; it must be a child of the Body element and it can appear only once in a SOAP message.

Although using SOAP over HTTP allows for easier communication through proxies and firewalls than previous remote execution technology, the technique has the disadvantage of using an application level protocol (HTTP) as a transport protocol (critics have argued that abusing a protocol by using it in a different purpose may conduct in sub-optimal behaviour). The pros and cons of SOAP are listed in Table 1.

### 4.1.5 REST
REST is an architectural style for distributed hypermedia systems such as the World Wide Web. The term was introduced in 2000 in the doctoral dissertation of Roy Fielding [Fielding 2000], who also participated in the IETF [IETF] working groups on URI, HTTP and HTML. The systems which follow REST principles are called RESTful.

Shortly, the basic REST principles are:
- Application state and functionality are abstracted into resources; all types of documents can be used as representations for resources (XML, XHTML, HTML, PNG, ...)
- Every resource is uniquely addressable using a universal syntax for use in hypermedia links (URI – Uniform Resource Identifier)
- All resources share a uniform interface for the transfer of state between client and resource, consisting of a constrained set of well-defined operations (represented by the GET, POST, PUT and DELETE methods) and a constrained set of content types (optionally supporting code on demand);
- The transfer protocol is client-server, stateless, cacheable and layered.

A RESTful web service is a simple web service implemented using HTTP and the principles of REST.
### SOAP

<table>
<thead>
<tr>
<th><strong>Pros</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language, platform, and transport independent</td>
<td>Conceptually more difficult, more “heavy-weight” than REST</td>
</tr>
<tr>
<td>Designed to handle distributed computing environments</td>
<td>More verbose</td>
</tr>
<tr>
<td>Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS-*), and tooling from vendors</td>
<td>Harder to develop, requires tools</td>
</tr>
<tr>
<td>Built-in error handling (faults)</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Pros and Cons of SOAP

### REST

<table>
<thead>
<tr>
<th><strong>Pros</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language and platform independent</td>
<td>Assumes a point-to-point communication model— not usable for distributed computing environment where messages may go through one or more intermediaries</td>
</tr>
<tr>
<td>Much simpler to develop than SOAP</td>
<td>Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophisticated requirements are harder to develop (“roll your own”)</td>
</tr>
<tr>
<td>Small learning curve, less reliance on tools</td>
<td>Tied to the HTTP transport model</td>
</tr>
<tr>
<td>Concise, no need for additional messaging layer</td>
<td></td>
</tr>
<tr>
<td>Closer in design and philosophy to the Web</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Pros and Cons of REST

The Web Application Description Language (WADL) [Hadley 2009] is designed to provide a machine processable XML description of HTTP-based Web applications. It intends to provide a simple alternative to WSDL for use with XML/HTTP applications, typically REST web services. WADL can be thought of as the REST equivalent of Web Services Description Language version 1.1. Version 2.0 of WSDL can be used to describe REST Web services, thus competing with WADL.

**WADL vs WSDL:** WADL and WSDL 2.0 HTTP binding are similar but do have some differences. In short, WADL is simple and has limited scope. By design, WADL is limited to describing HTTP applications and does not address features such as security. On the other hand, the WSDL 2.0 HTTP binding is more feature rich, at the cost of increased complexity, yet still lacks a true resource-centric model [Takase 2008].

### 4.2 Pre-CLARIN Work in the LRT community

#### 4.2.1 GATE

GATE (General Architecture for Text Engineering) is one of the most widely-used systems of its type and is a comprehensive infrastructure for language processing software development. It was first released in 1996, and then completely re-written in 2002, reaching its version 5.0 in June 2009.

The key features of GATE are:

- Component-based development, which aims to reduce the systems integration overhead in collaborative research.
- Automatic performance measurement of Language Engineering (LE) components, which promotes quantitative comparative evaluation.
- Clear distinction between low-level tasks such as data storage, data visualisation, and components loading and the high-level language processing tasks.
- Clean separation between data structures and algorithms that process human language.
- Consistent use of standard mechanisms for components to process linguistic data.
• Use of open standards such as Unicode and XML.
• Insulation from idiosyncratic data formats, by performing automatic format conversion and enabling uniform access to linguistic data.
• Provision of a basic set of LE components that can be extended and/or replaced by users as required.

As of 2008, GATE has been developed into a suite of related tools; GATE Developer\(^2\), GATE Embedded, GATE Cloud and GATE Teamware\(^3\)

**GATE Developer**
GATE Developer is an open source desktop application written in JAVA that provides a user interface for professional linguists and text engineers to bring together a wide variety of text analysis tools and apply them to a document or set of documents. GATE Developer incorporates more than 80 separate text tools as plug-ins\(^4\). Some have been developed in-house, others have been written specifically for GATE and others have been ported from stand-alone open-source tools.

GATE also has a command line interface and standard set of java libraries to allow developers to embed GATE-derived applications (custom collections of processing tools and rule sets) into other applications. The most commonly used GATE Developer application is ANNIE, a standardized collection of tools for information extraction, which comprises: tokeniser, gazetteer, sentence splitter, regex sentence splitter, part of speech tagger, semantic tagger, orthographic coreference and pronominal coreference. GATE Developer also has tools for performance benchmarking, text alignment, machine learning, automatically creating ontologies, viewing and editing existing annotations. GATE Developer is an open source \(^5\) offered under the LGPL licensing framework and runs on Windows, Mac, and Linux.

**GATE Embedded**
GATE Embedded is an object library optimised for inclusion in diverse applications that gives access to all the services used by GATE developer and more.

**GATE Cloud**
GATE Cloud is a software tool aimed at parallel execution of automatic [semantic] annotation processes. It is intended to process batches comprising large numbers of documents in a robust, long-running process. Its main characteristics are:

• Scalability: uses several parallel execution threads
• Flexibility: a set of parameters can be used to configure its behaviour.
• Robustness: designed to run for extended periods of time.

Low operating effort: designed as a batch process that does not require operator oversight or manual interventions.

**GATE Teamware**
Text Annotation is an essential part of the NLP systems and often required as a way of attaching metadata to documents. There are numerous text annotation environments in existence but almost none provides:

• Collaborative work over the internet.
• Flexibility to support different annotation tasks
• Employment of different user roles (from the cheap labour to highly skilled work).
• Service orientation (NLP tasks are realized as Web services)
• Support for complete annotation lifecycle (with clearly demarcated start and finish)

GATE Teamware does all of that and can be briefly described as a software suite and a methodology for the implementation and support of annotation factories. It is intended to provide a framework for commercial annotation services, supplied either as in-house units or as outsourced specialist activities.

\(^2\) [http://gate.ac.uk/sale/tao](http://gate.ac.uk/sale/tao)
\(^3\) [http://gate.ac.uk/teamware](http://gate.ac.uk/teamware)
\(^4\) [http://gate.ac.uk/sale/tao/splitch9.html#chap:misc-creole](http://gate.ac.uk/sale/tao/splitch9.html#chap:misc-creole)
\(^5\) [http://gate.ac.uk/download/index.html#latest](http://gate.ac.uk/download/index.html#latest)
It can be seen as a great complement of the GATE Developer because:

- It structures intervention by different actors (human and machine) into clearly defined roles and provides the means to manage them in a unified fashion,
- It adds User Interfaces (UIs) oriented on other necessary staff roles to GATE’s developer-oriented facilities,
- It is methodological instead of purely technological.

GATE Teamware splits project-related tasks into logical roles, including Manager (can setup and oversee a project), Annotator (can manually add or correct annotations on a document-by-document basis), Curator (a super annotator who can review existing annotations and approve documents) and Administrator (can oversee the entire Teamware interface).

Developed as a web based application with a workflow management system in its heart, it allows users to easily set up an annotation project and manage it over the Internet. This allows for a broadly distributed workforce to be applied to a single project.

GATE Teamware provides two kinds of user interfaces:
1. Annotation Editor implemented as JAVA Web Start delivers functional components to the desktop.
2. Management interface allows project set-up, including:
   - Choosing or uploading a document collection (a “corpus”)
   - Choosing or uploading a schema to constrain manual annotations
   - Choosing or setting up re-usable project templates
   - Applying pre-processing (automatic annotations) to a corpus
   - Selecting project participants and assigning projects to specific users
   - Monitoring progress and various project statistics
   - Viewing, searching and comparing annotated documents.

Both types of interfaces are shown in the figures below.

![Figure 1: Annotation Editor](image-url)
4.2.2 ALPE and Schema Recognition

ALPE (Automatic Linguistic Processing Environment) is a component based metadata framework which facilitates integration of LT tools of different origins sharing XML input-output specifications. At the base of ALPE is a hierarchy of XML annotation schemas, partially ordered by subsumption relations. On another level, the directed edges of this metadata hierarchy are associated NLP modules that are able to produce the transformations incurred by the pairs of corresponding subsumed schemas (in all cases a module adds information to the input file). Finally, to be used extensively, a hierarchy of schemas should be conformant to a universe of standards, which would orchestrate the employment of metadata (element names and attribute values) for a large community of users. However, since standardization usually comes late, in many cases, when no standards are still available, researchers need to invent their own formalisms, including metadata notations, in order to go on with their work. For this reason, nodes in ALPE could be surrounded by clouds of semantically equivalent notations. This feature of the framework should be taken as an opening towards the use of a huge diversity of notations, only few of them being already standardized.

ALPE offers now a number of functionalities, among which: is able to identify the annotated format of an input file (thus, allowing the classification of the file inside the hierarchy, either by finding the node the format of the file corresponds to, or proposing a new node and placing it in the right place in the hierarchy); can simplify the XML annotations in a file (i.e. transforming a file conformant to a certain node to one conformant to one of the node’s ancestors from the hierarchy); can merge the annotations in two or more files (if applied to the same hub document); can compute the path that links two nodes of a hierarchy (which correspond to the processing flow that allows the transformation of a file conforming to the annotation described by the start node to one observing the annotation of a destination node); finally, if all steps of a computed flow are available as modules (either locally or as Web services), ALPE can apply the flow to an input file in order to obtain the expected output.

In order to facilitate interoperability, we study now the possibility to detect the semantics of an annotated file and to map it towards a node of the hierarchy. Seen this way, the ALPE hierarchy becomes a constellation of "pivot formats", which could be described more formally, for instance, as given by the Lexical Markup Framework. We believe that this feature could become feasible in the near future, which would allowing also for the automatic computation of wrappers (format transformation programs), accommodating thus all kind of notations for the LT resources and making existing modules to collaborate in workflow chains even if their input-output specifications do not fit exactly. Moreover, the ALPE philosophy makes possible to think some
issues which are central in CLARIN, as multilinguality, cost and IPR, in correlations with solution finding for complex text processing problems on a solid theoretical ground, which is also mirrored closely by immediate and easy implementations.

4.2.3 Leipzig Linguistic Services

In 2004 web service access to digital text and lexical data as well as NLP algorithms and tools was established at the NLP department of the university of Leipzig [ASV09a, ASV09b, Bue09]. The Leipzig Linguistic Services (LLS) include:

- a very large frequency sorted dictionary of German word forms including POS information, sample sentences and co-occurrences,
- monolingual corpora of standard size for currently 48 different languages,
- a tool for sentence boundary detection,
- graph based clustering,
- co-occurrence statistics,
- synonyms and similar words computed on word’s co-occurrence profiles, and
- automatic terminology extraction.

Since then, about 70 million requests (2007: 27 million, 2008: 37 million excluding November and December) on the LLS were made. As of 2004, a total of 18 services have been installed covering 12 languages.

Architecture

The LLS infrastructure defines different tasks:

1. Editing and configuring a service,
2. Generating and deploying a service on the server and
3. Users accessing the Linguistic Services.

Editing and configuring services: Caused by the requirement for easy integration of services, in LLS the insertion of new data for NLP and Text Mining algorithms as services can be done by adding a new dataset in the table of a relational database. This dataset includes information like the class implementing the access to the data, a service name, a list of input fields of the service like word, limit or minimum significance, authorization level information and a service description.

Generating and deploying a service on the server: Within the LLS infrastructure automatic generation and deployment though a completely automatic process based on Apache ANT. This process also generates UI clients and makes these available to the end user.

Users accessing the Linguistic Services: SOAP is being used as standardized application protocol. Currently, there are well known client implementations in Java, .NET, Perl, Python and Delphi. Sometimes, however, interoperability problems arise.

Technical Experience

In this section, we summarize the main results of the last 5 years. Most of those results are based on a two years release corpus of access log files containing about 43.3 million datasets. 200,000 of them are failures and errors caused by a high load on the server. For 22.8 million of the 43.1 million non-failure requests a non-empty result could be delivered. For 20.3 million requests, however, an empty result had to be provided.

There are several reasons why almost 50% of all queries lead to empty results:

- **Spell checking errors**: One sort of an empty result is caused by spell checking errors. Users presumably have requested words of chat texts or different kinds of forums that do not adhere to standard spelling.
- **Encoding errors**: Another category of empty results are words requested with a damaged encoding.
- **Artefacts/fragments of web pages**: Some users are forwarding to the web services all kinds of information they can get from web sites. Thereby, they are also extracting RGB colour descriptions like ff0000, or actions from action forms.
- **Topics lacking in corpora**: Some frequently requested topics like fashion, gymnastics, linguistics and business language are under-represented in our public German newspaper corpus.
- **Services with insufficient data**: In comparison to the services Frequencies and Sentences, services like Baseform or Synonyms are based on less data and are not available for every language.
LLS is a web service for linguistic data, NLP and Text Mining processing tools running stable since 5 years. By looking in detail at 43 million query logs, we have been able to gain some insights into who are the users of these services, what they are used for, and some of the problems arising.

While LLS is a local service only, based on a number of data and tools available at our site [ASV09b], the next step to go is to offer these services in a unified environment for language resources (a language resource infrastructure) like DSpin [DSPIN] and CLARIN that enables a wide application of existing language resources (LR) from different organizations.

4.2.4 RACAI Services

Tokenizing, Tagging, Lemmatizing, Chunking and Linking Free Running Texts

TTL [Ion, 2007] is a text preprocessing module developed in Perl. Its functions are: Named Entity Recognition (by means of regular expressions defined over sequences of characters), sentence splitting, tokenization, POS tagging, lemmatization and chunking. The NER function is included as a preprocessing stage to sentence splitting because end of sentence markers may constitute parts of an NE string (i.e. a period may be a part of an abbreviation). POS tagging is achieved through the HMM tagging technology. The POS tagger of TTL follows the description of HMM tagger given in [Brants, 2000] but it extends it in several ways allowing for tiered tagging, more accurate processing of unknown words and tagging of named entities (which are practically labelled by the NER module before actual POS tagging). The TTL’s tag-set is the MSD\(^6\) with its smaller (subsuming) subset CTAG. (TTL tagging methodology follows the tiered tagging approach [Tufiş, 1999] where MSDs are recovered from an initial CTAG annotation). Lemmatization is achieved after POS tagging by lexicon lookup (generally, a word form and its POS tag uniquely identify the lemma) and in the case of word forms that are not in the lexicon, by a statistical lemmatizer which automatically learns lemmatization rules from the lexicon (for details see [Ion, 2007]). Finally, chunking is implemented with regular expressions over sequences of POS tags. It is not recursive and it does not perform attachments (PPs to NPs or VP for instance).

The TTL web service\(^7\) offers the following remote procedures (actual names taken from the WSDL file):

1. SentenceSplitter which takes as parameters the language of the text to process (currently either “en” or “ro”) and a SGML entity encoded text and returns another string which is a list of sentences separated by carriage return/line feed sequence (“\r\n”);
2. Tokenizer which has as parameters the language code and a sentence and returns a list of tokens separated by “\r\n” each token possibly carrying its NE tag (added to the token with the tab character “\t”) given by the NER module of the SentenceSplitter in the case the token is a NE (i.e. a real or integer number, a roman number, percents, abbreviations, dates, clock times, etc.);
3. Tagger which takes the language code and a tokenized sentence from Tokenizer and returns a MSD tagged sentence which is a string with triples of token, “\t”, MSD separated by “\r\n”;
4. Lemmatizer uses the POS tagged sentence along with the language code and returns a lemmatized sentence which resembles the one from the Tagger’s output except that the token annotation is enriched with its lemma which is separated again by a “\t” from the MSD tag;
5. Chunker is the final operation of TTL and, beside the language code, it takes a lemmatized sentence and returns the same sentence with chunk information added after the lemma annotation;
6. XCES is a helper function which calls all the previously mentioned operations in order and returns an XML representation of the result.

In principle, TTL operations are to be pipe-lined from 1 to 5, Chunker being the last function call, Lemmatizer the last but one function call, and so on and so forth till the SentenceSplitter operation which takes the actual text as parameter. Since TTL operates with SGML entities and not UTF-8 representation of the text, the user is required to transform the input text from UTF-8 to SGML by calling UTF8toSGML helper function of the TTL web service.

To get a feel of how the TTL web service is invoked, here is a short example written in Perl using the SOAP::Lite package. The example shows the processing of the English sentence:

“This is a simple example of a web service remote execution.”.

\(^6\) http://nl.ijs.si/ME/V2/msd/  
\(^7\) WSDL located at: http://ws.racai.ro/ttlws.wsdl
use SOAP::Lite;
my( $soap ) = SOAP::Lite->new()
   ->uri( 'http://ws.racai.ro/pdk/ttlws' )
   ->proxy( 'http://ws.racai.ro/' );
print( $soap->Chunker( "en", $soap->Lemmatizer( "en", $soap->Tagger( "en", $soap->Tokenizer( "en", $soap->SentenceSplitter( "en", "This is a simple example of a web service remote execution." ) ) ) ) )
   ->result()
); #end print

We have set the URI of the TTL web service which is its universal identifier onto the Internet with the uri method of the newly created SOAP object. Then, we have specified the physical URL of the web server which hosts the service with the proxy method. We are now ready to call all the TTL's public procedures. By writing $soap->Chunker(...) for instance, the SOAP::Lite package does all the hard work for us: it encodes the method call and its input parameters into a SOAP message, it sends the message to the web server web server and receives from it another SOAP message which encodes the procedure's return value. It parses the response SOAP message and extracts the result which then it presents to us with the result() method call.

So, the result of running this program looks like this:

This  Pd3-s  this
is  Vmip3s  be  Vp#1
a  Ti-s  a  Np#1
simple  Afp  simple  Np#1,Ap#1
example  Ncns  example  Np#1
of  Sp  of  Pp#1
a  Ti-s  a  Pp#1,Np#2
web  Ncns  web  Pp#1,Np#2
service  Ncns  service  Pp#1,Np#2
remote  Afp  remote  Pp#1,Np#2,Ap#2
execution  Ncns  execution  Pp#1,Np#2
.  PERIOD  .

Information was added from left to right: tokens, MSD tags, lemmas and chunking information. Regarding the chunks, every token has a list of the chunks it belongs to. The order of the chunks in the list signifies chunk inclusion (e.g. the token 'service' belongs to the prepositional phrase no. 1 and to the noun phrase no. 2 and the noun phrase is included in the prepositional phrase).

XCES is a another function of TTL which turns the vertical text format exemplified above into an XML encoding, compliant with the XCES specifications (http://www.cs.vassar.edu/XCES/). For our recurrent example, the result of invoking the XCES function is shown below:

<seg lang="en">
  <w id="example.1">
    <w lemma="this" ana="Pd3-s">This</w>
    <w lemma="be" ana="Vmip3s" chunk="Vp#1">is</w>
    <w lemma="a" ana="Ti-s" chunk="Np#1">a</w>
    <w lemma="simple" ana="Afp" chunk="Np#1,Ap#1">simple</w>
    <w lemma="example" ana="Ncns" chunk="Np#1">example</w>
    <w lemma="of" ana="Sp" chunk="Pp#1">of</w>
    <w lemma="a" ana="Ti-s" chunk="Pp#1,Np#2">a</w>
    <w lemma="web" ana="Ncns" chunk="Pp#1,Np#2">web</w>
    <w lemma="service" ana="Ncns" chunk="Pp#1,Np#2">service</w>
    <w lemma="remote" ana="Afp" chunk="Pp#1,Np#2,Ap#2">remote</w>
    <w lemma="execution" ana="Ncns" chunk="Pp#1,Np#2">execution</w>
  </w>
</seg>
LexPar

LexPar [Ion 2007] is a word linker. A link between two words in a sentence is a syntactic-semantic approximation of a dependency relation as described in [Mel'čuk, 1988] in that relation orientation and labeling are omitted. A link structure of a sentence (called a linkage) is constructed with a Lexical Attraction Model [Yuret, 1998]. We have improved the convergence properties of a LAM with a syntactic filter that rejects links that are not syntactically valid (i.e., a link between an adverb and a determiner).

The LexPar web service\(^8\) provides only one function: LinkSentence. This function generates the linkage of the tokenized, tagged and chunked sentence. The input parameters of this function are the XCES encoding of the sentence to be processed and the language code. LinkSentence returns the XML encoding enriched with the linkage information. The output in the case of our example is given below:

\[
\begin{align*}
\text{<seg lang="en">} \\
\text{<s id="example.1">} \\
\text{<w lemma="this" ana="Pd3-s" head="1"> This</w>} \\
\text{<w lemma="be" ana="Vmip3s" chunk="Vp#1" head="5"> is</w>} \\
\text{<w lemma="a" ana="Ti-s" chunk="Np#1" head="5"> a</w>} \\
\text{<w lemma="simple" ana="Afp" chunk="Np#1,Ap#1" head="5"> simple</w>} \\
\text{<w lemma="example" ana="Ncns" chunk="Np#1" head="1"> example</w> }
\end{align*}
\]

We can see that for all but one token there is a head attribute. This attribute has an integer as its value which indicates the position in the sentence (0 based numbering) to which the token is linked (the naming of the attribute does not imply that the token with the head information is actually the head of the relation). The token without this attribute (in our example the verb be) is the root of the linkage. The linkage of the sentence is a connected graph (and it is always planar and acyclic graph). The cases in which the graph is not connected might appear because the syntactic filter occasionally rejects good links which otherwise (in the vast majority of cases) are not correct.

TextProcessing

The TextProcessing web service\(^9\) has only one function - process(string input, string language) that takes three arguments: the first is the text to be processed, the second is the language (currently, only processing for „en“ and „ro“ are available).

The TextProcessing web service is similar to the TTL web service but faster (with the expense of a lower accuracy on the intermediary steps). The web service does tokenization, sentence splitting, POS-tagging and produces annotations with morpho-syntactical descriptions in one step. The web service will return for the English sentence “This is a simple example of a web service remote execution.” the annotated results:

This|this|DMS|Pd3-s is|be|VERB3|Vmip3s a|=a|TS|TI-s simple|simple|ADJE|Afp example|example|NN|Ncns of|of|PREP|Sp a|=a|TS|TI-s web|web|NN|Ncns service|service|NN|Ncns remote|remote|ADJE|Afp execution|execution| execution|execution|execution|execution|execution|execution|execution| execution|execution| .|PERIOD|PERIOD

In the sample client interface\(^10\) the user has the possibility to send archived files to be processed and she receives also archived results. Archived content is used in order to minimize bandwidth consumption.

WordNet Browser

The Wordnet browser\(^11,12\) is a web service that allows browsing through aligned wordnets. For now, only the Princeton 2.0 and the Romanian wordnets are available but the web service can be easily extended with wordnets of different languages or even with wordnets of different versions. Currently, Romanian Wordnet has 55000 synsets. As the figures for Romanian wordnet are continuously changing, one should check the latest statistics at [http://nlp.racai.ro/wnbrowser/](http://nlp.racai.ro/wnbrowser/).

---

8 WSDL located at: [http://ws.racai.ro/lxpws.wsdl](http://ws.racai.ro/lxpws.wsdl)
10 Sample client located at: [http://nlp.racai.ro/WebServices/TextProcessing.aspx](http://nlp.racai.ro/WebServices/TextProcessing.aspx)
The data of both wordnets is stored in a database with records like:

```
<SYNSET><ID>ENG20-12977363-n</ID><POS>n</POS> <SYNONYM><LITERAL>cvintillion</LITERAL><SENSE>1</SENSE></SYNONYM>
<DEF>un milion de cvadrilioane</DEF>
<ILR>ENG20-12969974-n<TYPE>hypernym</TYPE> </ILR><DOMAIN>number</DOMAIN>
<SUMO>PositiveInteger</SUMO><SENTIWN><P>0.0</P><N>0.0</N><O>1</O></SENTIWN></SYNSET>
```

Each record is indexed by literal and synset id. This simple representation allows the client of the webservice to search for both literals and synsets.

As the example above shows, the Romanian wordnet contains not only the Princeton WordNet specific data but also the IRST DOMAIN [Bentivogli et al, 2004], SUMO [Niles & Pease, 2001] and SentiWordnet [Esuli & Sebastiani, 2006] annotations. Currently these annotations can be visualized on the web browser available at [http://nlp.racai.ro/wnbrowser/](http://nlp.racai.ro/wnbrowser/).

A common usage scenario for the current wordnet web service is to translate a word to and from Romanian/English: (i) the client applications queries the web service for all the synsets ids of a given literal in the target language; (ii) the client queries for all the synsets of the corresponding ids in the source language; (iii) the client application extracts the literals from the source language synsets. The literals from the last processing stage are the translation of the given literal.

For example, one can first check what wordnet sources are available using the function:

```java
String[] GetSources()
```

The web service will return the array of strings "wn20-en" and "wn20-ro". If we want the information on the Romanian word “biografie” the first step would be to call the function:

```java
String GetLiteral(string source, string literal, int level)
```
like

```java
getLiteral("wn20-ro", "biografie", 0);
```

If the argument “level” is bigger than 0, then the synset relations will be recursively expanded with the related synsets for \( n \) levels. The result for this function is:

```
<Result source="wn20-ro" literal="biografie">
<SYNSET><ID>ENG20-06113482-n</ID><POS>n</POS>
<SYNONYM>
<LITERAL>biografie</LITERAL><SENSE>1</SENSE></SYNONYM>
<LITERAL>viață</LITERAL><SENSE>16</SENSE></SYNONYM>
<DEF>Expunere (scrisă și comentată) a vieții unei persoane.</DEF><BCS>1</BCS>
<ILR>ENG20-06111883-n<TYPE>hypernym</TYPE></ILR><DOMAIN>telecommunication</DOMAIN>
<SUMO>Biography</SUMO><SENTIWN><P>0.0</P><N>0.0</N><O>1</O></SENTIWN></SYNSET>
</Result>
```

With the function

```java
String GetSynset(string source, string ili, int level)
```
we can get the equivalent English synset:

```java
getSynset("wn20-en","ENG20-06113482-n",0);
```
with the results:

```
<SYNSET><ID>ENG20-06113482-n</ID><POS>n</POS>
<SYNONYM>
<LITERAL>biography</LITERAL><SENSE>1</SENSE></SYNONYM>
<LITERAL>life</LITERAL><SENSE>8</SENSE></SYNONYM>
<LITERAL>life story</LITERAL><SENSE>1</SENSE></SYNONYM>
<LITERAL>life history</LITERAL><SENSE>1</SENSE></SYNONYM>
<DEF>an account of the series of events making up a person's life</DEF><BCS>1</BCS>
```

WG2.6 Requirements
In the future versions of this service there will be added new functions to allow queries on aligned wordnets using different criteria like synset relations, domains, SUMO information, etc.

Language Identification
The Language Identification web service is derived from a standalone application that was initially aimed at autonomously collecting web data for English and Romanian. It was also meant to check whether all the paragraphs/sentences in a given, presumably, monolingual corpus where indeed written in the respective language. For instance, this is how we cleaned-up the Romanian part of the 22-language parallel corpus JRC-Acquis (Steinberger et al., 2006). Currently, the web service distinguishes among the 22 languages of the European Union, present in the JRC-Acquis parallel corpus. The function takes as its sole parameter a string consisting of a fragment of text and returns a string in which one can find the language code for that text and a confidence score for the classification. The Language Identification service can be easily extended with arbitrary new languages (as we already did for two endangered languages) because the implementation is language independent, trainable on language specific data. The training module is not available as a web service, but a user interested in having a new languages included in the Language Identification engine may contact the administrator of the web service platform (ws-admin@racai.ro) and send training data. The new languages will be eventually added in short time.

Most developers construct their applications for language identification using N-gram or Markov chains approaches. We have taken a different approach. Given training texts in different languages (about 1.5Mb of text for each language), a training module estimates the frequencies of the prefixes (the first 3 characters) and the suffixes (4 characters endings) by analysing all the words in the texts, for each language. Thus, for every language two models are constructed. The models will contain the weights (percentages) of prefixes and suffixes in the texts representing a language. In the prediction phase, for a new text, two models are built on the fly in a similar manner. These models are then compared with the stored models representing each language for which the application was trained.

Factored Statistical Machine Translation (FSMT)
The FSMT web service is based on the Moses decoder - the open source factorized translation system. Factored translation models (Koehn and Huang, 2007) integrate additional information into the word form model. The additional info can be represented by several factors like lemma, part-of-speech, morphological description, etc. The additional information can help wordform translation or reordering the translated word forms.

Configuration of the Moses decoder is based on alternating translation and generation steps: (i) translating lemmas; (ii) generating morpho-syntactical descriptions from lemmas; (iii) translating morpho-syntactical descriptions; (iv) generating the surface form based on lemma and morpho-syntactical description. The translation web service is available for 4 language pairs: English-Romanian, English-Greek, English-Slovene and Romanian-English.

The web service has only one function: translate(string languagePair, string input). The translation web service requires factorized input. The factorized input (having additional information for each word form, like lemma, POS, morfo-syntactical descriptions) can be obtained using the TextProcessing web service to annotate user input (available only for English and Romanian).

The corpus used for training the translations systems for Romanian and English is a compilation of the Jrc-Acquis corpus and several other parallel resources available at ICIA (aligned RoWordnet, journalistic parallel corpus, etc.). The corpus contains 800,000 translation units with more than 30 millions tokens. The corpus was annotated with lemma, POS tags and morfo-syntactical descriptions.

13 WSDL located at: http://www.racai.ro/WebServices/LangId.asmx?WSDL
14 Sample client located at: http://www.racai.ro/WebServices/LangId.aspx
15 WSDL located at: http://www.racai.ro/WebServices/FactoredTranslation.asmx?WSDL
16 Sample client located at: http://www.racai.ro/WebServices/FactoredTranslation.aspx
The corpus used for training the translation systems for English-Greek and English-Slovene is the SEnAC corpus – SEE-ERA.net Administrative Corpus (Tufis et al, 2008). The SEnAC corpus (64,352 translation units and about 1.5 million tokens per language) is richly annotated, thus allowing a great variation of factored translation models (4 factors available: wordform, lemma, part-of-speech and morpho-syntactical description).

Web Services in Testing
In addition a number of new web services have been developed which have now entered the testing phase:

1. Search Acquis, a search engine web service
2. JRC Acquis Question Classifier (JRCACQCWebService) for Romanian
3. JRC Query Generator for Romanian
4. Lexical Chainer, returns lexical/semantic paths in the WordNet hierarchy between two dictionary word forms (lemmas).
5. Paragraph Aligner, aligns English-Romanian parallel texts at the paragraph level, but it can also be used for sentence alignment

These are described in more detail in Appendix E.

4.2.6 UPF web services

Statistical Web Services
The Statistical Web Services (WS)\textsuperscript{17} family performs statistical tasks on a specific corpus. This corpus must be provided by the user using the Upload Web services. Each one of the corpora will be assigned to an unique and persistent identifier which has to be used to reference it by one of the statistical-task web services. The describeCorpusByFile\textsuperscript{17} web service provides statistical information for each file of the corpus.

The following statistical information is provided:

- "Herdan" index of lexical richness.
- "type-token ratio" index of lexical richness.
- corpus vocabulary extension, that is, the number of different words in the corpus.
- vocabulary extension of the sub-corpus formed by all the precedents files and the current file
- corpus extension, that is, the number of words in the corpus.
- cummulated vocabulary extension, that is, the sum of the corpus extension of all precedent files and the current file

The describeCorpusByLength\textsuperscript{20} web service treats the corpus as if it was one document and splits it into N sub-corpora of length defined by an input parameter (length). Next, the Herdan index of lexical richness is computed on each of the sub corpora. The kwic\textsuperscript{21} web service provides regular expression search on the whole corpus. It returns the context (N words at the left side and N at the right side) for each result and the file of the corpus where the result is located.

The Ngrams\textsuperscript{22} web service implements the N-grams statistic package using the Pedersen libraries described in http://www.d.umn.edu/~tpederse/nsp.html. Returns all the n-grams of the corpus with its number of occurrences and its coefficient value.

Upload Web Services
The Upload web service uploads to a UPF server one corpus UTF-8 encoded (text)file located in a public url (received as parameter) and generates an identifier to refer to that corpus in local scope. The web service caller is issued a corpus identifier while the corpus is uploaded in the background. With this corpus identifier

\footnotesize{\textsuperscript{17} Location: http://gilmere.upf.edu/WS/pedersen/v1/invoke
\textsuperscript{18} WSDL located at: http://gilmere.upf.edu/WS/pedersen/v1/wsd
\textsuperscript{19} Location: http://gilmere.upf.edu/WS/jaguar/jaguar/invoke_method_params?method=Herdan&service=jaguar
\textsuperscript{20} Location: http://gilmere.upf.edu/WS/pedersen/v1/invoke_method_params?method=DescribeCorpusByLength&service=v1
\textsuperscript{21} Location: http://gilmere.upf.edu/WS/pedersen/v1/invoke_method_params?method=Kwic3&service=v1
\textsuperscript{22} Location: http://gilmere.upf.edu/WS/pedersen/v1/invoke_method_params?method=Ngrams&service=v1}
the status of the upload process can be monitored. Comparable functionality is offered for uploading zipped corpora consisting of multiple files.

**AAILE Web Services**
The AAILE\(^{23,24}\) service groups concordances according to the syntactic contexts the keyword occurs in. Given a lemma, a corpus and a predefined set of admissible syntactic contexts (expressed in terms of regular expressions) the system (i) looks for all occurrences of the lemma in corpus, (ii) constructs the corresponding vectors (taking into account the set of regular expressions) and (iii) groups the vectors. The idea is that when looking for occurrences in corpus, the lexicographers receive an organized set of examples. The system is restricted to nouns and adjectives.

**CQP Web Services**
The CQP\(^{25,26}\) web service allows querying the IULA corpus which is indexed in CQP. Current version is version 4. This web service includes 4 functions (in the latest version):

- CqpQueryResults GetPendingResults(int ticket_no)
- string ResultsAvailable(string ticket_no)
- string QueryCqpExpression(string cqp_expression, string corpus_name)
- string GetPendingResultsXml(string ticket_no)

ResultsAvailable and CqpQueryResults allow asynchronous queries. Thus, the cqp query sent by the user with QueryCqpExpressions is stored in a dB at the server side which returns a ticket number. The system triggers the queries in dB and marks them as available as soon as they are executed and stores the results obtained. ResultsAvailable queries the dB checking whether the returned ticket number is ready or not. When the ticket is available, CqpQueryResults sends the available ticket and gets the stored results. More information can be found at [http://cqp-ws.rubyforge.org/](http://cqp-ws.rubyforge.org/).

The CQP BagOfWords\(^{27,28}\) service, currently version 3, creates a CQP web service query and returns a data matrix which collects the words that go with the lemma. (The ultimate goal of the service is to provide the lexicographer with a structured set of concordances. Thus, the system classifies occurrences according to the words that go with the key word. This clustering is performed in CQP BagOfWordsClustering. Here, the system generates the corresponding matrix for the set of occurrences in corpus). This web service includes 4 functions (in the last version):

- int CqpQuery(string cqp_wsdl_url, string word, string domain, int window_size, int cut_size)
- string GetPendingResultsFileUrl(int ticket_no)
- bool ResultsAvailable(int ticket_no)
- string CqpXmlInput(string cqp_xml_url, int window_size)

bool_ResultsAvailable and int_CqpQuery allow for asynchronous querying. Thus, the cqp query sent by the user with int_CqpQuery is stored in a dB at the server side which returns a ticket number. The system triggers the queries in dB and marks them as available as soon as they are executed and stores the results obtained. bool_ResultsAvailable queries the dB checking whether the returned ticket number is ready or not. When the ticket is available, string_GetPendingResultsFileUrl sends the available ticket and gets the stored results. More information in [http://cqp-bagofwords.rubyforge.org/](http://cqp-bagofwords.rubyforge.org/). CqpXmlInput is another version of CqpQuery, where the occurrences are not taken from a CQP indexed corpus but a xml file. This file must be accessible to be downloaded. An example of this XML file can be found at [http://gilmere.upf.edu/WS/bag_of_words/system/cqp.xml](http://gilmere.upf.edu/WS/bag_of_words/system/cqp.xml)

The CQP BagOfWordsClustering\(^{29}\) service performs clustering on the matrix returned by CQP BagOfWords

---


\(^{25}\) Location: [http://gilmere.upf.edu/WS/cqp/v4/invoke](http://gilmere.upf.edu/WS/cqp/v4/invoke)


Common Tools Web Services
A common XSLT Transformer REST service is supplied that performs XSL transformations on given XML and XSL content. A Driver for testing can be found at: http://gilmere.upf.edu/WS/xslt/v1/transformation

4.2.7. Heart Of Gold
Heart of Gold is a middleware architecture for creating and combining markup produced by multiple natural language processing components in multilingual environments. It was initially developed for a special sort of multi-dimensional annotation, namely application-oriented, XML- and XSLT-based online integration of various shallow NLP components with a deep HPSG parser PET for increased robustness in the hybrid natural language processing paradigm (Callmeier et al., 2004). The middleware, however, can also be used for various other online and offline tasks related to multi-dimensional markup creation and integration. These comprise automatic corpus annotation, incorporation of multi-dimensional markup into a single XML representation, and NLP component cascades interleaved with XSL annotation transformation. The middleware provides XML-RPC interfaces for simple, networking-enabled and programming language-independent application and component integration [Schaefer 2006].

The Heart of Gold has been developed under the wings of the EU-funded project DeepThought and the BMBF-funded projects Quetal, HyLaP and and in the context of the DELPH-IN collaboration.

The core middleware architecture (and also the PET system) is available under an LGPL open source license. Some of the components for which adapters are provided are only available for research purposes or have their own licenses different from LGPL.

4.2.8 ISO/LIRICS API's
LIRICS (Linguistic Infrastructure for Interoperable Resources and Systems) [LIRICS] was a European project within the e-Content programme which aimed to provide ISO ratified standards for language technology to enable the exchange and reuse of multilingual language resources and facilitate implementation of these standards for end-users by providing an open-source implementation platform, related web services and test suites building on legacy formats, tools and data.

LIRICS delivered a portfolio of standards for ratification by ISO including:
- a meta model for lexical representation;
- meta models and data categories for morpho-syntactic and syntactic annotation;
- reference data categories for semantic annotation;
- test suites in nine European languages;
- an open-source implementation platform, compatible with major legacy systems and tools;

The API's delivered for the LIRICS project include API's for accessing the Data Category Registry (DCR), Morphosyntactic Annotation Framework (MAF),Syntactic Annotation Framework (SynAF) and Lexical Markup Framework (LMF).

The DCR (ISO 12620:2009) provides a framework for defining data categories compliant with the ISO/IEC 11179 family of standards. According to this model, each data category is assigned a unique administrative identifier, together with information on the status or decision-making process associated with the data category. In addition, data category specifications in the DCR contain linguistic descriptions, such as data category definitions, statements of associated value domains, and examples. Data category specifications can be associated with a variety of data element names and with language-specific versions of definitions, names, value domains and other attributes. As part of the LIRICS project an initial design of a browse and search web service interface to the DCR has been implemented. This interface has been further developed in the current 12620 DCR implementation, ISOcat. The full (REST) API specification is available at http://www.isocat.org/rest/help.html. A SOAP implementation is planned. The Data Category Registry API is

---

31 Location: http://gilmere.upf.edu/WS/xslt/v1/transformation
32 http://heartofgold.dfki.de/
currently used by tools such as LEXUS\(^{33}\) and ELAN\(^{34}\) to provide references to data categories for annotated multimedia files and lexica respectively.

The MAF (ISO/DIS 24615:2009) API\(^{35}\) distinguishes two levels of APIs; Service APIs for interactions with a processor and Linguistic Content APIs for manipulating the linguistic content returned by a processor. The MAF service API distinguishes 2 methods; one to get the languages supported by the MAF processor and one that performs the morphosyntactic annotation of the raw text. The MAF content model is supplied through a RelaxNG schema. The MAF API can be found at: http://lirics.loria.fr/doc_pub/del5.1.C-v2-maf-api.pdf.\(^{35}\)

The SYNAF (ISO/DIS 24615:2009) API\(^{36}\) distinguishes between two levels of APIs:
- Service API for interactions with a processor,
- Linguistic Content API for manipulating linguistic content returned by a processor.

The SYNAF API can be found at: http://lirics.loria.fr/doc_pub/del5.1.D-v2-synaf-api.pdf. LMF (ISO 24613:2008) is the ISO standard for Natural Language Processing lexicons and Machine Readable Dictionaries. The LMF API\(^{37}\) developed in LIRICS provides a browse and search interface for accessing LMF resources. The LMF API can be found at: http://lirics.loria.fr/doc_pub/del5.1.B-v2-lmf-api.pdf and is part of the Lexus implementation.

4.2.9 TextGrid

TextGrid\(^{38}\), which is part of the D-Grid initiative\(^{39}\) in Germany, focuses on the creation of a community grid for the collaborative editing, annotation, analysis, and publication of primary and secondary texts. The architecture of TextGrid enhances a Globus-based\(^{40}\) grid infrastructure with a secure middleware layer and an open, WebService-based layer of specialised tools for working with texts. The user interface (TextGridLab) is based on the Eclipse RichClient Platform.

In order to keep this infrastructure as modular as possible and to foster interoperability, TextGrid tools (at least the streaming components) are implemented as WebServices. These tools/services act as building blocks of specialized functionality. Atomic functionalities such as tokenization, lemmatizing, or collation can be wrapped into individual services or organized by user-defined workflows, to be re-used by other services or plugged into an application environment. Depending on the extent to which a service is integrated, it might be fully embedded in the grid environment and entered into a topic map-based service registry\(^{41}\).

In order to foster interoperability as well as to keep the initial implementation effort as low as possible, integration into the service environment can be performed in several steps\(^{42}\). The first step of integration is rather loose, with growing requirements as well as growing possibilities with each additional step. For example, authentication is necessary from the third step on. Each initiative can choose how far they advance into the TextGrid service network.

Workflow-enabled

The service can be mashed up with other services in a workflow. The workflow editor in the TextGridLab may embed the service in a comprehensive batch process. Basically any programming language and host environment may be used for the services, yet web service standards form the basis for interoperability. Ideally, the following standards should be supported resp. implemented:

\(^{33}\) http://www.mpi.nl/lexus
\(^{34}\) http://www.mpi.nl/elan
\(^{38}\) http://www.textgrid.de/
\(^{39}\) http://www.globus.org/
\(^{40}\) To be implemented in TextGrid II.
• WSDL 2.0, SOAP 1.2 (document-literal style) or REST
• Character encoding: UTF-8

Registry-entry
The installation of a service registry is planned for the future. This registry allows the documentation of services and their interfaces. The more generic a service is designed, the better for its reuse. For example, a lemmatizer for a specific language should avoid prescribing a specific XML/TEI schema; rather, it can take any stream of text and return the text with respective annotations for the lemmatized words, ideally enabled to parameterize the output markup or at least employ the same tags as other lemmatizers in the service network do.

The following general guidelines are advised to the service producer:
• Provide sufficient documentation with your service.
• Learn from other services and follow conventions.
• Test your service exhaustively, and provide feedback-mechanisms for users to report bugs.

User interface
A graphical user interface is available and it can be embedded (with other services) in user environments. Initially, the primary user environment is an Eclipse-based client (i.e. the TextGridLab), though web-based or other clients may follow. Actual user environments may be tailored to a specific user group (e.g. discipline, project), thus each service/tool may be embedded in various user environments. The user interface (resp. each user interface, e.g. Eclipse-based, web-based, etc.) embeds nicely into the respective environment and follows its conventions. Documentation can be found at the respective user interface project.

Grid access
The service carries all the necessary information to be allowed into the grid. This includes information for authentication and logging. TextGrid is Shibboleth-enabled [SHIBBOLETH] and will interconnect with the national scientific Shibboleth federation DFN-AAI [DFN] once productive. For the time being, users can register at TextGrid directly to be granted access. Grid access entails the possibilities to join projects, deposit and share (private) digital objects, and similar activities. In the future, licensing policies may build on authentication information.

Interoperability
With the measures mentioned above, the service implements all requirements for interoperability with other services in the TextGrid e-Humanities service network. This includes TextGrid services, as well as services from Interedition43 and other sources.

4.3 Current state Workflow Systems

4.3.1. TRIANA, TAVERNA and KEPLER
In order to have a basis for considering the feasibility of Workflows involving language technology modules according to CLARIN assumptions a survey was conducted at UPF on selected existing workflow tools(Kepler, Taverna and Triana).

To start the assessment of the difficulties of workflows in the Language Technology domain a workflow for testing purposes was defined that incorporated all the features that could be required: interoperability defined by standards, supportive workflow editors in addition to the following:
- CLARIN workflows must interact with web services. REST and SOAP without distinction
- IF / THEN / ELSE clauses must be supported
- Information returned from web services are used as part of conditional statements
- Data manipulation of web services results: it is very important to be able to access to specified data inside complex objects coming from web services.

43 http://interedition.huugensinstituut.nl
- Constants and variables declaration for internal workflow use. Even complex types could be important. Scope of variables is an issue as well (Globally declared variables can cause problems and may be overridden at a local scope)
- LOOP clauses:
  - For each
  - While
  - For to
- Loops in parallel. Web services can be slow and it is not needed to do it sequentially. Parallelization features are required. Parallelization in loops does not involve big changes and the benefit of doing it could be big.
- Every operation, even if it is small should be a web service
- For optimum performance, very common tasks could be web services hosted in the CLARIN user computer (but still as a web service).
- Exception handling (special behaviour when an error is raised by a web service). Since the Registry will have a lot of information about alternative web services, mirrors, etc, CLARIN should handle exception handling automatically in many cases.

To test and simulate a scenario where remote services from different providers were involved, a workflow was defined which can be summarized as follows:

1) for a given input text, we want to know the language it is written in using the GetLanguage Web Service.
2) for each word in the input text, we want to get the KWIC (key word in context, that is: concordances taken from corpus) using the CQPquery Web Service in UPF. Depending on the language returned in the previous step, a different corpus will be used (thus, if language is English the corpus to be used is EN, if language is Spanish the corpus is ES, and so on.).

Although the scenario in itself is not very interesting, it proved useful for the testing purposes. This apparently trivial scenario involves some tasks that posed too many problems:

1) an input text is sent to GetLanguage web service which returns a complex output type with a 'language label' and a 'confidence rate'. We need to manage this output complex data type in order to identify the 'language label' and the 'confidence rate'. Workflow editors include some local tools that allow us to manage complex data. For the sake of simplicity, we obviate the 'confidence rate' and only get the 'language label'.

2) Once the language label is obtained, each word input needs to be sent to the CQPquery web service in order to get the KWIC. This web service needs two inputs: the word to be searched and the corpus where the word is to be searched. The corpus involved depends on the language, thus we have three different corpora according to weather the language is English, Spanish or Catalan. This apparently simple scenario involves some difficult tasks:
   - the CQPquery web service needs to be triggered for each word in input string. Thus we need some kind of 'for each' loop.
   - the CQPquery requires two inputs: the word to be searched and the corpus. In our case, we have a list of words and a single 'corpus label'. In other words, the iteration process not only needs to read the whole word list but also needs to `generate` the corpus label.
   - there are format requirements, thus input words in CQPquery web service are [word="inputword"]). In this case we need some format conversion.
   - finally, some conditional control is needed in order to set the corpus according to the language label. In this case we needed an `if then, else if`.

3) In order to avoid blocking of client processes, Web Services were implemented in an asynchronous way. Thus, once the word(s) and corpus are sent to CQPquery web service, the remote server returns a ticket number and stores the query in a dB. Queries in dB are executed and when they are ready they are labeled as 'true'. A second process named ResultsAvailable sends a ticket number to the server and checks weather it is ready. When the 'ready' ticket is received, a third process named GetPendingResults sends the ticket to the server and returns the results. Note that not only a conditional control but also a retry loop was needed which, in case the ticket is not ready, retries the query. An alternative approach would be to introduce a callback mechanism where the service CQP query service invokes a callback method on the client.
The overall process can be summarized as follows:

```
(language,confidence) = ws.GetLanguage(input_text)
If language = English Then corpus = EN
If language = Spanish Then corpus = ES
If language = Catalan Then corpus = CA

for each word in input_text
    ticket = ws.CQPquery(word, corpus)

status = ws.AvailableResults(ticket)
If status true then go to KWIC = ws.GetResults(ticket)
If status false then go back to ws.AvailableResults(ticket)
```

When trying to edit this workflow using the work flow editors that have been reviewed (Triana, Taverna and Kepler) many problems were encountered:

- None of the work flow systems made it possible to implement the “IF language THEN corpus” condition in a trivial way. Only Taverna offers a ‘friendly’ solution, in this case its Java Bean editor could be used. (All tools deal well with Boolean branching conditions, but this is not the case.)
- The entire experiment could not be run in Triana as server errors were encountered which could not be solved.
- It was not possible to deal with the iteration (FOR EACH) process over the input text in Kepler. Taverna allowed correct iterations over the input list, in this case the ‘Dot cross’ strategy was used which worked correctly.
- It was not possible to deal with the “IF true THEN” control condition used by the asynchronous Web Services. Although Kepler includes a local IF/THEN construct that could be used, the ‘retry’ action when the returned vale is false could not be implemented.
- Taverna does not recognize Boolean types. Web Services had to be modified in such a way that they only returned string values in order to overcome this problem and be able to test Taverna.
- Format conversion is not trivial. Kepler and Taverna include some local tools that deal with format conversion but they did not fit our needs completely. Again, the use of the Java Bean editor in Taverna proved useful. An external conversion solution is always possible of course.

**Triana**

Triana is one of two test bed applications developed for GridLab, a large EU funded project. The aim of GridLab is to develop a simple and robust grid application toolkit (GAT) enabling applications to exploit the power of the GRID.

**Technical overview:**

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Data flow based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Proprietary language.</td>
</tr>
<tr>
<td>Concurrence</td>
<td>It has a good level of concurrency control with merge and block components.</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Being a data flow driven software Triana has a very good level of parallelism.</td>
</tr>
<tr>
<td>Loops</td>
<td>Triana has a very good level of loop control. It can be done in several ways depending on the user decision and it accepts dynamic variables to control the exit condition. Any logic operation with variables can be used for the exit condition.</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Conditions</td>
<td>The conditional components are not as powerful as the loop components and there is only a very basic IF clause.</td>
</tr>
<tr>
<td>exception handling</td>
<td>Exception handling is good enough. Every component may have an error output (called error node). Actions can be programmed in the workflow when an error appears.</td>
</tr>
<tr>
<td>Other</td>
<td>Triana does not include a way to insert small algorithms in the workflow (this must be done by a web service or a local compiled component).</td>
</tr>
</tbody>
</table>

**Descriptive information:**

<table>
<thead>
<tr>
<th>Developers</th>
<th>Developed by Cardiff University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Astronomy and life sciences.</td>
</tr>
<tr>
<td>Maturity</td>
<td>Projects using Triana: GridOneD, GEO 600, BiodiversityWorld, DIPSO, FAEHIM, GEMMS, GENIUS Grid portal, Data-Mining Grid</td>
</tr>
<tr>
<td></td>
<td>The last publication reported on the Triana's web page is 2007.</td>
</tr>
<tr>
<td>future plans</td>
<td>-- no information available --</td>
</tr>
<tr>
<td>Community</td>
<td>Information taken from NesCForge:</td>
</tr>
<tr>
<td></td>
<td>Public mailing lists: 6 lists with no activity</td>
</tr>
<tr>
<td></td>
<td>Public Forums: 2 public forums with no activity</td>
</tr>
<tr>
<td></td>
<td>First release: beta version 3.0 date: 2004-08-12</td>
</tr>
<tr>
<td></td>
<td>Last release: version 3.2.2 date: 2007-04-24</td>
</tr>
<tr>
<td>eHumanities</td>
<td>The DataMiningGrid project is a shared cost Strategic Targeted Research Project (STREP) granted by the European Commission (grant no. IST-2004-004475). It was part of the Sixth Framework Programme of the Information Society Technologies Programme (IST)</td>
</tr>
<tr>
<td></td>
<td>Based on Triana</td>
</tr>
<tr>
<td></td>
<td>170 downloads from sourceforge</td>
</tr>
<tr>
<td>Integration</td>
<td>Triana runs on windows and linux.</td>
</tr>
<tr>
<td>documentation</td>
<td>Enough documentation when starting. However most of the functionalities and tools are not documented.</td>
</tr>
</tbody>
</table>

**The GUI**

a) Installation & documentation:
- Easy installation and well documented.
- The GUI includes help and tutorial (with some missing images and not completed).
- Problems when looking for help/support on the web.
- Not easy to start working with it.

b) Editing workflows:
The task of editing workflows is a bit verbose. Triana includes *Local workflows* (a large library of "local tasks") and *Distributed workflows*. Distributed components within Triana include grid-oriented components (GRAM44 & GRMS45) and service-oriented components (web services and P2P). Service-oriented components use a GAP (Grid Application Prototype Interface) Interface which provides job submission and file-transfer operations within Triana.

44Grid Resource Allocation Management del Globus project
45Grid(Lab) Resource Management del GridLab project
Once a web service is imported, it appears as a tool in the tool tree alongside the other tools and can be connected into a Triana workflow in exactly the same manner as other ‘local’ tools.

Triana includes a graphical editor where selected processors (used in TRIANA to denote a component perform a unit of work) are dragged into the editor window. Input/output information is displayed in a pop-up window when the mouse is over the processor. As soon as processors are added to the editor, the system establishes the links between them.

c) Input / output:
In Triana input renderers are ‘tools’ and they are listed in the tool tree for ‘Triana Tools’ together with any other processor. The user needs to choose the relevant input/output tool in order to read/write inputs/outputs. As Triana was created to support astronomy and live sciences, most of the input/output tools are irrelevant to humanities. In order to deal with complex inputs, the user needs to generate static type classes and create custom tools.

d) Searching:
Triana includes a searching facility. However we were not able to search in the UDDI repository.

Taverna
Taverna is a free software workbench for designing and executing workflows, created by the myGrid project, and funded through OMII-UK.

Technical overview:

<table>
<thead>
<tr>
<th>paradigm</th>
<th>Data flow based</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Proprietary language, the Simple Conceptual Unified Flow Language or SCUFL. (Taverna 2 no longer uses SCUFL.)</td>
</tr>
<tr>
<td>concurrence</td>
<td>The concurrence offered by Taverna is basic and based on merge 2 branches when both are finished. It is enough for many workflows but some advanced ones are outside this approach.</td>
</tr>
<tr>
<td>parallelism</td>
<td>Parallelism is a primary concept in Taverna since it is Data Flow driven and almost all tasks are performed in parallel when possible.</td>
</tr>
<tr>
<td>loops</td>
<td>Taverna has not a while component that allows for iterations but there is the possibility to use list of items as input in the operations and set the option “iterate”. It will execute the operation once per item. It is allowed as well to set several input lists and the input will be taken one item of each list per processing iteration or as a Cartesian product per iteration. In most of the cases this system of iterations is enough but should have more powerful loop control like while statement.</td>
</tr>
<tr>
<td>exception handling</td>
<td>Taverna has a basic exception handling mechanism included. It supports retry of invocation with configurable timeout and number of retries, and user-defined alternatives for processors failing constantly.</td>
</tr>
<tr>
<td>other</td>
<td>All operations in Taverna must be encapsulated as a web services or local services programmed in Java. There is no possibility of simple data manipulation in the workflow. It is very useful for connection between processes. Some times it is required to manipulate a little bit an output of a web service for being the input of another one, just a simple mathematical operation for instance.</td>
</tr>
</tbody>
</table>

Descriptive information:

<table>
<thead>
<tr>
<th>developers</th>
<th>Created by the myGrid project, and funded through OMII-UK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original myGrid Partners: EMBL-EBI University of Manchester, University of Newcastle, University of Nottingham, University of Sheffield, University of Southampton IT Innovation Centre</td>
</tr>
<tr>
<td>domain</td>
<td>e-biology</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
| maturity | Figures taken from sourceforge (at 28 July 2008):
  - downloadable files: 49671 total downloads to date
  - taverna -1.7.1: 1513 downloads to date

Many 'e-biology' projects can be accessed using Taverna: Seqhound, BioMoby, BioMart, BioTeam iNquiry, Utopis, BioMart, EMBOSS.

Taverna (and myExperiment) is present in many conferences and international events. For 2008, myExperiment participated in: I Intelligent Systems for Molecular Biology Conference (ISMB 2008) in Toronto, 9th annual Bioinformatics Open Source Conference BOSC 2008, e-Science All Hands Meeting in Edinburgh, ...

| future plans | Taverna 1.7.1 was launched in 2007.
Taverna 2 was launched in 2008. A plug-in is already available and seems to work properly.
Taverna includes a 'check for updates' service and a plug-in service, to keep the tool updated. |

| community | Mailing Lists: Different mailing lists with 2580 users, moderate usage.
Taverna has a social web site named myExperiment with over 964 users, 82 groups, 301 workflows, 101 files and 15 packs. MyExperiment 'makes it really easy to find, use and share scientific workflows and other files, and to build communities'. |

| eHumanities | text-mining |

| integration | Taverna runs on any modern PC or Mac, running any recent version of Windows, Linux, OSX and most UNIX like operating systems as long as Java version 5 |

| documentation | Good documentation. However, some examples in the documentation are not in the distribution. Although the GUI is really friendly and 'easy' to use, it does not include any help. |

The GUI:

a) Installation and documentation
Taverna GUI is nice and friendly. It is easy to install, execute and start with. Taverna is in general well documented.

b) Workflow editor
The editor is friendly and nice and includes drag-and-drop facilities and visual facilities. Taverna has two spaces: the 'editor' (Advanced Model Explorer) and the visualization. It also includes and interactive graphical (experimental) editor. The 'editor' panel contains a tree with five initial branches for processors, inputs, outputs, data connections and coordination links. Basically, the process of editing workflows goes as follows:

- The user selects the desired processors. When processors are drag-and-dropped into the 'editor' panel they are automatically placed under the processors node.
- The user has to define the inputs and outputs of the workflow and link the processors.
- The graphical panel displays the diagram as we are editing the workflow. The system allows for different visualizations and the diagram can be saved.

c) Inputs
The edition of inputs (and outputs) in Taverna is different from that of Triana and Kepler. In Taverna, inputs are not longer listed together with 'processors'. They have a different status. The user creates as many inputs as needed under the branch 'inputs' in the 'editor' panel. The 'input' branch is populated with the created inputs. The user names the input and edits the metadata assigned to it. Metadata includes (i) free text...
description (used to give instructions to other users about how to populate the inputs) and (ii) tagging of the input (or output) with MIME types. In case of outputs, MIME types are crucial as they determine the selection of renderers within the result browser (see below).

Taverna includes tools to deal with complex inputs in the workflow. When a web service has a complex input (an XML document) it is hard for the user to provide the required data without having previous knowledge of the structure. Taverna includes an 'XML splitter' processor which deals with complex inputs. XML splitters show the child inputs to the user. Child inputs can be 'edited' as simple inputs and the values are assigned in the standard manner. When a processor has a complex input, the system automatically offers the possibility to include an XML splitter to deal with.

d) Assigning values.
Contrary to Triana and Kepler, in Taverna input values do not have to be prespecified. That is, the assignation of values to inputs is not done during workflow construction but is done during workflow execution. When a workflow is invoked, a pop-up window is displayed to assign input values. The user can enter values in different ways: typing, reading from a file, reading from a directory or from a previously saved 'file input definition'.

The assignation of values in Taverna is 'independent' from the workflow definition. In Triana, for example, if we want a workflow to read from a file, we need to select a specific 'read from file' tool. Similarly, if an XML output is produced, Taverna includes an XML viewer which automatically displays the data and allows the user to save it. In Triana, the user has to choose beforehand whether she/he wants to see the data (no syntax displayed) or save them.

The system allows specifying default values. In this case, default values are assigned during the edition process. When running workflows requiring inputs, these can be saved. This allows to re-run the workflow using the same inputs.

e) Outputs
Taverna includes different renderers to display results (results may have different formats). Provided the correct MIME types are specified for the workflow output, the renderer selection mechanism will select, by default, an appropriate renderer component. Thus, the user can see plotters, graphics, XML files and text files without having to specify anything.

The list of MIME types includes:

- text/plain=Plain Text
- text/xml=XML Text
- text/html=HTML Text
- text/rtf=Rich Text Format
- text/x-graphviz=Graphviz Dot File
- image/png=PNG Image
- image/jpeg=JPEG Image
- image/gif=GIF Image
- application/zip=Zip File
- chemical/x-swissprot=SWISSPROT Flat File
- chemical/x-emb/ dl-nucleotide=EMBL Flat File
- chemical/x-pdb=PPD File
- chemical/seq-aa-genpept=Genpept Protein
- chemical/seq-na-genbank=Genbank Nucleotide
- chemical/x-pdb=Protein Data Bank Flat File
- chemical/x-ml-molfile

Taverna includes third party visualisation tools. This is the case of SeqVISTA graphical tool (which displays some chemical... types) and Jmol (an open-source Java viewer for chemical structures in 3D). Taverna allows the user to inspect and save intermediate inputs and outputs of individual processors both during and after a workflow invocation.
Outputs can be saved in different manners.

- **Iterations**: When a processor expects single sequence and receives a list of sequences iteration takes place automatically. Specific iterations behaviors can be defined.

- **Fault tolerance**: Taverna includes the following fault tolerance settings:
  - the ability to retry after failures
  - user can set the number of retries
  - user can set the time between retries
  - the ability to define alternative processors when all retries have been exceeded
  - the ability to define a processor as 'critical', in this case the workflow stops

- **Discovery, metadata-management and grimoires registry**: Ta is the component within myGrid responsible for semantic service search. Feta is composed of two components, namely Feta Client and Feta Engine. The Feta Client is a GUI-plug-in to Taverna which is used to search for services descriptions of which are provided by the Feta Engine.

Taverna includes a good discovery tool, user can search by: name & description and, most interesting, by tasks, method and resources used, input and output, and type. Multiple queries can be issued at the same time to save the communication time. Multiple searching criteria can be combined in one query.

Grimoires is a UDDIv2 compliant service registry. It was originally developed for the myGrid Project (www.mygrid.org.uk). Currently, it is a managed program project of Open Middleware Infrastructure Institute (www.omii.ac.uk). Grimoires provides metadata annotation/discovery and WSDL registration/discovery functions that are not supported by UDDI.

f) **Interesting features**:
- Taverna includes the 'Resource usage report' functionality which shows the various external resources used by the current workflow. Such functionality is useful for documentation purposes.
- The workflow diagram includes different visualization options.
- For large workflows some non-interesting parts can be marked as 'boring'. Boring processors are hidid from diagram.
- myExperiment plug-in allows access to workflows in myExperiment. This allows for both browsing and direct invocation.
- Provenance tools: The Taverna Log Book is a plugin for Taverna that allows users to automatically log their experiments in a database and browse, reload, rerun and maintain past workflows.
- LSID (Live Science Indentifiers) for editable metadata associated to a workflow. Any time the user edits a definition, the system connects to whatever LSID authority is configured in the mygrid.properties file and asks for a new LSID suitable for a workflow definition. This then provides a globally unique identifier for the workflow definition.
- Taverna offers the ability to save 'input configuration' to re-run the experiment with the same data input.
- Ability to define alternate processors in case of failure
- Taverna allows the user to inspect and save intermediate inputs and outputs of individual processors both during and after a workflow invocation.
- Good discovery tool (services can be discovered by exploiting their semantic descriptions. This is done by Tavern Feta plug-in).
- Taverna includes a BeanShell editor tool (this is easier than working with Java and does not need to compile and use external tools).
- Breakpoints allow users to stop the workflow and edit data before going on.

**Kepler**
Kepler is a cross-project collaboration led by the Kepler/CORE team (UC Davis, UC Santa Barbara, and UC San Diego). The software builds upon the Ptolemy II framework, developed at the University of California, Berkeley. Ptolemy II is a software framework designed for modeling, design, and simulation of concurrent, real-time, embedded systems.
Technical overview:

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Data flow based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Proprietary language. Workflows can only be created and executed using Kepler tool. Kepler uses the proprietary Modelling Markup Language (MoML).</td>
</tr>
<tr>
<td>Concurrence</td>
<td>The concurrence control is based on Merge as well that is enough for many workflows but not for some more sophisticated.</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Parallelism is full executed as normal in Kepler.</td>
</tr>
<tr>
<td>Loops</td>
<td>Kepler has a lot in common with Taverna, it has no loops since it is data flow driven but in Kepler is not used the internal iteration system that Taverna has. In this way, any kind of iteration should be described using tricky ways. In addition, dynamic variables cannot be used as loop exit conditions.</td>
</tr>
<tr>
<td>Conditions</td>
<td>Good conditional components. We find select, switch, comparator, logic function, equal, isPresent and some more.</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>Exception handling is not available in Kepler. Although the user can program raising exceptions in the required situations, there is not a way of catching exceptions when a web service is missing, timeouts, etc…</td>
</tr>
</tbody>
</table>

Descriptive information:

<table>
<thead>
<tr>
<th>Developers</th>
<th>Developed by the members of the Ptolemy project at UC Berkeley.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Molecular biology, ecology, geosciences, chemistry and oceanography.</td>
</tr>
<tr>
<td>Maturity</td>
<td>First Kepler alpha 6 version released on April 29, 2005 Last release (Kepler 1.0.0 ) May 2008 Kepler is nightly updated.</td>
</tr>
<tr>
<td>Future Plans</td>
<td>kepler-users mailing lists with moderate usage. kepler-dev mailing list or technical discussions with active usage. Kepler includes a kepler repository wich can be accessed from the web. The tool allows searching the repository. The organisation of the reporitory and search capabilities is not developed.</td>
</tr>
<tr>
<td>eHumanities</td>
<td>Text-mining. MultiChek (the Multivalment – Chesire- Kepler VRE project) aims at developing a collaborative engineer environment able to provide new methods of creating, sharing, disseminating and reusing scholarly information.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Good documentation for both users and developers. Help manuals are good. It includes flash demos and examples.</td>
</tr>
</tbody>
</table>

The GUI
Easy installation and well documented. The Workstation includes user manual, examples and a ‘Kepler Actor’ reference file. The graphical tool is good and has nice functionalities; however we faced some refreshing problems. (When editing and instantiating Java components, the application had to be shut down and restart in order to refresh the whole workflow).
Editing workflows
In Kepler processors are called Actors. The system includes a large standard library of local actors. The way 'remote actors' are included in workflows differs from that of Taverna and Triana. In this case, the user needs to choose the correct actor 'type' from the local library. Once the relevant actor is dragged into the editor window, the user populates it with information concerning the wsdl uri and the method name.

Thus, for example, Kepler includes WebServiceActor and WsWithComplexTypes. These actors invoke the Web service and broadcast the response through their output ports. But whereas the first deals with web service operation with simple types, the later deals with web services with complex types. This means that the user needs to know whether the remote service requires simple or complex input types.

Note that in Triana and Taverna the way to 'load' remote processors from web services is quite different: the user enters the WSDL URI and the system populates the local library with the external processors. Once the remote processors are in the local library, the user 'drag&drops' them into the editor. In Kepler, the user selects the appropriate Actor and, once this is 'dropped' in the editor area, populates the Actor with WSDL information.

Again, the way GUls deal with processors requiring complex input types is different. In Triana, the user needs to select the appropriate input renderer. In Taverna, the system automatically offers the possibility to include the local tool XMLsplitter which deals with complex input types.

Kepler allows adding local Java Actors (the equivalent Java Beans in Taverna). Java Actors are edited and compiled outside Kepler before they can be 'imported' and used in the workstation. For non-expert users, this is not an easy scenario.

In Kepler, every workflow requires a director. The user selects the relevant Director which directs the execution of the workflow. The list of directors includes: Synchronous Dataflow (SDF), Process Networks (PN), Dynamic Dataflow (DDF), Continuous Time (CT) and Discrete Events (DE). For a non expert user, choosing the appropriate Director is a non-trivial task.

4.3.2 UIMA

UIIMA general information.
The Unstructured Information Management Architecture (UIIMA) is an architecture and software framework for the analysis of unstructured data. It provides support for creation, discovery, composition and deployment of multi-modal text analysis services (e.g. language identifiers, tokenizers, summarizers, categorizers, parsers, named-entity detectors and any other kinds of analytics that can operate on text documents, audio, video) and integrating them with UIM applications.

The Apache UIIMA framework\cite{uima-web} is an Apache licensed, open source implementation of the UIIMA Architecture. It consists of Java framework, of a matching C++ framework (also allows analytics to be written in Perl, Python, and TCL), and a Asynchronous Scaleout (UIIMA AS) framework (enables UIIMA clients to utilize UIIMA analytics as services running in separate processes on the same or different machines.

The UIIMA OASIS specification\cite{uima-std}, released in March 2009, has been approved as an OASIS standard which is intended to define platform-independent data representations and interfaces for software components or services that analyze unstructured information. However, the current Apache UIIMA framework implementation does not fully comply with it.

Data representation, modeling and interoperability in UIIMA.
All UIIMA analytics work on common data representation object that represents a stand-off annotation model (called CAS - Common Analysis Structure). A CAS conforms to the build-in feature structures and to user-defined Type System(s). Type System is a collection of inter-related type definitions and it models CAS. Each

\cite{uima-web}  http://incubator.apache.org/uima/documentation.html
\cite{uima-web}  http://incubator.apache.org/uima/
\cite{uima-std}  http://docs.oasis-open.org/uima/v1.0/uima-v1.0.html
type definition declares its name, its supertype and its attributes with their respective value domain. Value domains may be other types, including build-in types, String, or primitive types such as Boolean, Byte, Short, Integer, Long, Float, and Double, as well as Arrays of these types. For interoperability between text analysis services the common Type System(s) should be developed. It can be mapped to the existing metadata standards. The Type System is specified in a XML file called Type System Descriptor. For an example of Type System Descriptor see Appendix A.

To model a text analysis engine in UIMA one should:

- define a Type System that the CAS on which text analysis engine will work must conform to;
- declare types from the Type System that the text analysis engine requires as the input CAS;
- declare types from the Type System that the text analysis engine will produce in the output CAS;
- optionally define Subject of Analysis (Sofa) the text analysis engine will work on by specifying their names;
- optionally define language(s) of Sofa(s);
- optionally define input parameters (parameter names and valid values in terms of primitive types or if they are multi-valued);
- optionally define external resources necessary for text analysis engine to work with.

This metadata about analytic makes up analytic descriptor and is provided in XML format as well. For an example of an analytic descriptor see Appendix B. All text analysis services must implement a predefined framework interface. Interoperability of text analysis services is achieved through a shared Type System. They can be composed one after another in a chain if types defined in the output specification of one text analysis engine match types defined in the input specification of another text analysis engine.

**XML Metadata Interchange (XMI).**

UIMA uses the XML\(^\text{45}\) interchange format to represent UIMA-compliant analysis results. The use of XMI enables applications outside UIMA to more easily produce or consume these UIMA analysis results. UIMA provides support to serialize and deserialize CAS objects (UIMA object holding analytic analysis results) to and from standard XML format, and to convert UIMA Type System to and from core models (UML meta-models). An example of CAS serialized to XMI is presented in Appendix C. An example of Ecore model that corresponds to a UIMA Type System is presented in Appendix D. The adoption of XMI as exchange format within Clarin can be beneficial because it’s standardized, has tooling support, enables separation of content and representation, supports extensibility.

**UIMA text analysis engine as web services.**

UIMA supports several remote deploying options: as SOAP web services, as based on internal protocol Vinci web services, as Rest services, or as UIMA AS services. SOAP and Vinci deployment presupposes that the services will be called by clients running UIMA. Rest service is intended for UIMA unaware clients which are interested only in a result of processing a text, so that they can provide plain text as input and get as output result of processing in the simple inline xml format. UIMA AS provides a service wrapper that creates a shared UIMA service from a UIMA text analysis engine. UIMA AS services are asynchronous, enable load balancing and service instances can be dynamically added or removed at runtime. The UIMA AS design is based on JMS (Java Message Service) and uses Apache ActiveMQ to support messaging between components. UIMA AS also exposes numerous performance parameters via JMX (Java Management Extensions).

**UIMA from the Clarin perspective.**

UIMA is for working with text analysis services, multipliers, collection readers, i.e. with what is called Linguistic Tools in Clarin. UIMA is very suitable as framework for developing, discovering, composing and deploying this kind of services. On the other hand UIMA doesn't provide an interface for working with what is called Linguistic Resources in Clarin. Although such resources as corpora, video or audio stream can be modeled by Type System and read into CAS for some further manipulation, some other Clarin Resources that will provide querying a database kind of service have a completely different nature and UIMA doesn't provide interface, modeling and deploying environment for such kind of services.

---

\(^\text{45}\) XML Metadata Interchange(XMI) is an OMG standard for exchanging metadata information via XML
4.3.3 JBPM

JBoss JBPM is a powerful and open source business process management (BPM) system implemented in Java and sponsored by JBoss, one of the leaders in Java middleware open source software. JBPM enables the creation of business processes that coordinate between people, applications and services. Its architecture is very modular and flexible which allows easy process development and provides a scalable process engine.

JBPM differentiates itself from other BPM projects in the following aspects:

- It is easily embeddable into any Java application. BPM Systems typically require a separate server to be installed which makes it hard to integrate into the Java software development cycle. Embedding JBPM in a Java application means adding the JBPM library to the application classpath. The JBPM tables can be put in any database next to the tables of the host application.
- It has an easy-to-read, compact and feature complete JPDL (Java Process Definition Language), backed by nice visual editor. JBPM also supports other process languages, which will be discussed next.
- It has a very flexible transaction management mechanism which fits very well with the existing database layer in the host application.

JBPM provides business process management (BPM) and service orchestration in a multi-process language platform. Process language describes how people and/or systems work together, or more technically speaking, it describes an execution flow as graphical activities that can be wait states. JBPM supports several process languages, as shown in Figure 3.

![Figure 3: Process languages in JBPM](image)

The Process Virtual Machine (PVM) is a simple Java library for building and executing process graphs. It serves as a basis for various process languages. PVM runs in all Java environments, from standalone swing application, a web application on a servlet container like Tomcat, or an enterprise application on an application server like JBoss. It can work with or without relational database persistence. Native support for any process language can be build on top of the Process Virtual Machine. The runtime behaviour of each activity in the process graph is delegated to a Java interface. Process languages are a set of activity types. An activity implements the runtime behavior and corresponds to one activity type. So building a process language on the PVM is as easy as creating a set of activity implementations.

At the moment, JBPM supports the following process languages: JPDL, BPEL, Pageflow and in the very near future, with the forthcoming version 4.0 it will support also XPDL. JPDL is a process language implementation based on the Process Virtual Machine that exposes the PVM concepts natively. It defines a number of concrete process language constructs, an XML schema and a process archive format to store processes. It is an intuitive process language which describes workflows graphically in terms of tasks, wait states for asynchronous communication, timers, automated actions, etc. It also has very powerful and extensible control flow mechanism which glues these operations together.

---

The term workflow is used to denote a series of related interactions between people and a computer system. Where workflow is more often used in a technical context, business process has a broader scope. It implies also the non-technical aspects like analysis, organisational impact, from the viewpoint of a manager or process owner. So, BPM System is in its core a workflow system, but enriched with new services that provide more control and visibility over the process, like real-time monitoring or a business rule engine.

---

WG2.6 Requirements
In JPDL, process definitions are packaged as zip archives, containing: process definition XML file, layout description XML file and process image JPEG file. A process archive is uploaded to the process engine for execution. The process engine traverses a process graph, executes defined actions, maintains process state, and logs all process events.

JPDL in JBPM is open ended, which is a big difference from fixed process languages in traditional BPM systems. It provides easily extendable base with clean API for runtime behaviour. Unlike BPEL, which is strongly related to Web Services and Web services invocation (in BPEL, every activity has to be implemented as a Web service) JPDL is more of a component framework, allowing direct invocations of the Java handlers (actions) and supporting human tasks. Inside Java handler (action) any service can be invoked, therefore service orchestration is fully supported in JPDL. More about this you can read in section: Service Orchestration.

JBPM components
The main components of JBPM are: JBPM embeddable core (which contains workflow engine and repository used for storing process definitions, runtime execution data and history), Graphical designer and Web console application

The JBPM core library
The JBPM core is the plain java (J2SE) library, containing workflow engine for managing process and execution of process instances. It has a long list of nice features including:

- Database portability: JBPM uses well known Object Relation mapping framework called Hibernate internally for its persistence. Hibernate resolves the SQL dialect differences between the different databases, thus allowing JBPM to use any database, currently in the market.
- Identity component: The design of JBPM allows an easy integration with organisational structure of any system that is embedded to. That means there will be loose coupling with existing user management. Also, if application does not have user management, default JBPM identity component can be used for workflow task assignments.
- Job Executor: a component for monitoring and executing jobs in a standard (non enterprise) Java environment. Jobs are used for timers and asynchronous messages. In an enterprise environment, Java Message Service (JMS) Service can be used for that purpose and JBPM can reuse existing JMS based implementation.
- Business Activity Monitoring (BAM): a component that supports real-time monitoring (information about currently executing processes) and collection of historical data (information about processes that have been completed). This component is a precondition for employing some kind of business intelligence.

The graphical process designer
The JBPM JPDL Designer is a graphical tool for authoring business processes, implemented as an Eclipse plugin. The most important feature of the graphical designer tool is that it includes support for both the process designer (analyst, manager or researcher) as well as the developer. This enables a smooth transition from modelling to the practical implementation. The example of the process created in JBPM Designer is shown in Figure 4.
The latest version can be obtained through Eclipse update mechanism.

The JBPM console web application
The JBPM console web application serves two purposes. First, it serves as a central user interface for interacting with runtime tasks generated by the process executions. Secondly, it is an administration and monitoring console that allows an inspection and modification of runtime instances. As a part of distribution JBPM provides JSF based console. The new version 4.0 will have completely rewritten console application, implemented in GWT. However, it is not difficult to implement web application that uses JBPM API. GATE Teamware [Reference] has console web application for task orchestration, process monitoring and administration. One of the application screens is shown in Figure 5.

Overview of JBPM architecture
In this section we will list the main JPDL constructs and analyse the most important points of the runtime execution.

The main JPDL constructs
The following lists the main JPDL constructs.

Figure 4: Process created with JBPM designer

Figure 5: Task instance list in GATE Teamware
**Process definition** is based on a directed graph. The graph is composed of nodes, transitions, one start state, and one end state. The type of each node defines the runtime behaviour for the node. While a process definition executes, the below mentioned entities come into play.

- **Process instance** represents one execution of a process definition.
- **Token** is one path of execution. A token is the runtime concept that maintains a pointer to a node in the graph. When a process instance is created, a token is created for the main path of execution. This token is called the process instance’s root token and is positioned in the process definition’s start state.
- **Transitions** connect nodes and direct the flow of execution
- **Signal** instructs a token to continue graph execution by transition out of a node.
- **Nodes** are responsible for the continuation of a graph execution. When a token enters a node, the custom Java code defined in node is executed. This actually is completely correct only for “plain nodes”. However there are several implementations of nodes. Each has a slightly different. They will be explained below.
- **Actions** are instances of Java code executed when events occur in a process execution. The most common event types are “entering a node,” “leaving a node,” and “taking a transition.”

JPDL defines the following types of nodes:
- Start Node – Start of the process
- Task Node – Human Activity
- State – or how it is usually called: “Wait State”, does not propagate execution like plain nodes. Instead it waits for an external signal to trigger execution.
- Node (plain node) – Execution of Custom Code
- Decision Node – Decision based on process variables
- Fork – Splits execution along multiple paths
- Join – Combines multiple execution paths
- SuperState – Aggregation of multiple nodes
- End Node – Completion of the process

**Execution context**

An execution context can be seen as something similar to a HTTP session. It contains a set of named variables, stored as a map. Any node can access content of the execution context and read/write context
variables. The execution state is persisted in the database - when the process is in the wait state, its context with variables is stored in the database; when the process is resumed, context variables are extracted from the database and context is recreated.

This scenario is shown in Figure 6. Please note that “submit web sale” and “sales review” are wait states.

Service Orchestration
Here we will discuss the JPDL support for service orchestration, mainly:
- custom code execution (service invocation),
- decision logic,
- parallel execution
- loop support
- timers
- exception handling

Service invocation
A starting point for the service orchestration is a Node. A typical Node executes an action handler – a class implementing the \texttt{ActionHandler} interface. A Node can be configured to execute any Java code, e.g. web service call, both synchronously (as a “plain node”) and asynchronously (as a “wait state”). If a node execution is invoked asynchronously, the current state of the process is stored in the database. These stored instances can be picked up by the JBPM Executor for continuation.

Decision logic
A Decision node is another type of node that supports custom implementations, by taking \texttt{DecisionHandler} interface as a base. Hence, the custom code can provide any complex decision rules. The other way is to provide an expression condition in one of supported scripting languages like: Groovy, MVEL, BSF, etc.

Parallel execution
A parallel execution in JPDL is implemented using the combination of Fork and Join nodes. All the paths connecting Fork and Join nodes will be executed in parallel. A thing to keep in mind is that the number of Forks has to match the number of Joins.

Loop support
Loop construct is one of the things widely used in services orchestration, but it is missing in JPDL. However it is relatively straightforward to implement to loop types by writing a pair of custom action handlers: A sequential loop can be easily implemented using Decision node which evaluates loop process variable. A parallel loop can be implemented using Fork/Join with the runtime calculation of the number of transition paths.

Timers
The easiest way to specify a timer is by adding a timer element to the node. When a timer expires, an action (custom java handler) can be executed or a transition can be taken.

Exception handling
Process execution can encounter exceptions. JPDL allows association of an exception handler with every node. An exception handler is a special kind of action handler, which is invoked when exception occurs during the node execution.

Example
As an example, how all these concepts can be utilized, we will take a process called “Automatic Annotation” implemented in GATE Teamware. The process should do the following:

- Take a corpus of documents as an input
- If corpus is not provided, select it inside the manual task
- Create number of paths, so it matches number of documents (Fork node)
- Execute specified numbers of services (automatic annotator) in loop (iterator), for each document (path)
4.4 Concluding remarks

This chapter has provided an overview of some of the important work that has been carried out in the LRT domain and that is being contributed to CLARIN by its members. It also describes web services and workflow technologies that have been worked out so far. It is clear that large amounts of time and effort are associated with this work and that CLARIN cannot redo most of them. The web services and workflow domain itself however remains largely unorganized in which only small incidental networks and collaborations exist. In particular there is no coherent view on standards yet that can cater for linguistic interoperability.

For CLARIN it is important (a) to play in active role in bringing together this community and attempt to incorporate as much of this work, services, tools and expertise as possible without raising too many boundaries for participation and (b) to push standardisation. The intention of CLARIN is to establish a structured network of strong centers on top of these collaborations through which resources, tools and services are made available in a stable and highly available manner. It is the task of these centers to push forward these harmonisation and standardisation efforts.

As a start, a temporary LRT inventory\(^1\) has been set up listing all the resources, tools and services which have been made available by the CLARIN members. This is in addition to what is already contained in the IMDI, OLAC and ELRA catalogues. The services described in this chapter, are for a large part registered there. For the registration of web services in the final CLARIN registry it has become clear at the Munich workshop\(^2\) that neither UDDI nor EbXML are adequate to describe the web service characteristics for this domain in particular when we think of (semi) automatic profile matching for example to find useful tools for a specific task to be carried out in a given resource. As a result it has been decided that all web services and tools will be described using the Clarin MetaData Initiative (CMDI) framework which provides a flexible metadata framework with references to well defined metadata categories in the DCR. Formal specifications to these services may be provided through WSDL or WADL for SOAP and REST respectively and must be however be supplemented by additional data categories which express further constraints on the input/output characteristics of the web services which are not expressed in WSDL or WADL. Useful documentation and

---

\(^1\) http://www.clarin.eu/inventory

\(^2\) Munich, November 2008
guidelines must be provided for user reference. The services provided by RACAI and UPF provide an excellent basis for test cases in subsequent CLARIN work.

Leipzig Linguistic Services is currently the only service provider who is capable of handling high volume requests. This should become a concern if CLARIN is to be used by large numbers of researchers. Scalability of services must be addressed by the service providers when high request volumes are expected for a service. These services are expected to be made available in a unified environment for language resources (a language resource infrastructure) like CLARIN [CLARIN] that enables a wide application of existing language resources (LR) from different organisations. CLARIN, however, needs to address the execution scenario in a principle way for two reasons:

- CLARIN wants to motivate humanities researchers to carry out a large number of operations which are provided by CLARIN which could easily exceed the processing capacities of CLARIN centers.
- CLARIN knows about processes such as training stochastic engines for processing large amounts of texts or speech signals that require computational capacities exceeding the possibility of CLARIN centers.

Therefore, the CLARIN infrastructure needs to be amended with a "Preservation and Execution Infrastructure" which will hopefully be offered by the big computer centers soon.

For workflow systems two main fractions are represented in the LRT community: those using GATE and those using UIMA. CLARIN claims no preference to either approach but should rather accommodate for both. ALPE provides an interesting approach to integrate LT tools of different origin, but it will take some time for this system to stabilize. The suggestion to automatically detect formats and other resource properties and by doing so automatically select available software and processing steps is a very interesting one from a theoretical perspective. CLARIN needs to check whether and when these suggestions can be included in good practice.

Workflow systems, such as Triana, Kepler and Taverna, present approaches from other domains and may be of interest to parts of the LRT community, yet they lack firm grounding within the LRT community. Some of the outcomes of projects associated with these systems, such as the success of myExperiment.org may serve as examples on how to make these tools available to a wider community. JBPM is one of the major workflow systems in the IT industry, has a wide IT support base and is used in mission critical business systems. The Teamware approach demonstrates an interesting direction in which automated and human tasks are combined into workflows that span multiple participants. It is thus extending the workflow approach beyond the traditional domain of text analysis processing and highlights the possibility to create complex tasks where researchers collaboratively work on research questions in a completely distributed yet well defined manner. Details on how to deal with different workflow systems in the CLARIN environment will need to be worked out in subsequent workshops focusing on workflow issues.

Of course discussing interoperability issues at the format and linguistic encoding level will be an ongoing issue as well. Further workshops are scheduled already for example to bring various major stakeholders from various "communities" such as ACL, ELRA, CLARIN, FLARENET and ISO together. But this work will be continued in work package 5.
5. Web Services and Workflow Systems Requirements

5.1 Introduction

The ultimate goal of CLARIN is the construction of a shared distributed infrastructure that aims at making language resources and technologies available to the humanities and social sciences research communities at large. The nature of the project is therefore primarily to turn existing, fragmented technology and resources into accessible stable services that any user can share or adapt and repurpose.

A network of powerful centers is needed that can offer a wide range of stable and highly available services. These range from archiving services allowing for storage of data resources with a guarantee of long-term accessibility to ontological services providing access to widely accepted and widely defined domain concepts. These centers will also need to provide portals for registering all types of resources (data, tools and services) such that they are both human and machine interpretable. One of the guiding principles in CLARIN is that metadata for all resources, tools and services is provided through standoff metadata descriptions, that is, all metadata is provided in separate XML documents that are open to others to allow providers to harvest the descriptions through accepted protocols such as OAI-PMH. Standards need to be worked out that are flexible enough to cope with resource type and semantic differences to allow the construction of more advanced services through the creation of workflow processes. Automatic profile matching will become an important mechanism to support non-technical users in the construction of workflow processes suitable for their research questions. Metadata is to be specified as described for the Clarin MetaData Infrastructure (CMDI).

CLARIN can build upon a rich history of national and European initiatives in this domain. A large number of services and tools have been created by various organizations in the past which present valuable assets for analysis and enrichment of data resources. The open nature of CLARIN must make it possible to integrate these services and tools as much as possible into the CLARIN landscape without preference for specific tools or technologies. Thus, not only multiple tokenizers, part of speech taggers or tree banks will be expected to coexist within the CLARIN infrastructure but also multiple workflow or pipeline systems, such as GATE, UIMA or JBPM. An open approach is necessary since neither the potential user community, nor the use case scenarios, nor the daily work of humanities and social science researchers can be adequately predicted. If CLARIN is to expand beyond the currently represented communities then new varieties of tools and resources can be expected. The CLARIN infrastructure needs to provide a flexible solution for this.

In a heterogeneous environment such as CLARIN, interoperability will help to enlarge the domain in which resources, tools and services can be applied thus assisting in new ways of combining tools and resources for complex tasks. The current domain is already characterized by a large degree of both syntactic and semantic fragmentation, which needs to be addressed at the infrastructure level to at least make these differences visible. As mentioned, semantic and syntactic properties of resources, and input/output specifications of services must be properly described through relevant data categories in CMDI to allow for example for profile matching. Convergence of data formats into a smaller number of recommended formats will further make it possible to reduce the amount of time and effort needed to convert and prepare existing resources for further processing. It is expected that work package 5 will be providing requirements specifications and standards recommendations.

The next sections address specific requirements which are deemed relevant to the CLARIN infrastructure; workspaces, metadata, provenance data, AAI, I/O, IPR issues and semantic interoperability.

5.2 Relevant topics

First a number of topics are mentioned that can be seen as defining constraints for the CLARIN web services and workflow work in the construction phase.

5.1.1 Workspaces

Primary and secondary data will be stored in accepted CLARIN repositories and archives. Users will access this data and extend this in manners that are appropriate to the research questions at hand. Therefore CLARIN must ensure that users have access to temporal workspaces where they can carry out their operations and store created temporary data in a temporary fashion, i.e. an infrastructure of powerful compute
and storage nodes need to be available that are organized in a grid like manner [CLARIN 2008-6]. End users will typically use this workspace for day to day work. It must be easily possible to push results forward from private workspaces to shared repositories. However, the rules of operation in workspaces need to be the same to not create irritations with laymen as users and to not have to support different software versions. Sharing resources (and tools/web-services) by making them visible and accessible is one of the core pillars of CLARIN. All resources within a user’s private workspace must therefore also be associated with at least the minimal amount of required metadata information (see below). This metadata can initially be obtained through the user uploading the resource into his/her private workspace. Further access rights to resources may then be specified using an access management system.

Storing resources and metadata in workspaces is transient by nature, i.e. these may only exist for a certain period. This transient nature may be enforced through organizational policies and procedures on workspaces to limit the amount of time data can reside there and the amount of data that may be stored there. As a result, resource identifiers and metadata identifiers are not persistent identifiers. Once resources are stored in archives resources and metadata descriptions are assigned their final PIDs. Although this aspect will be largely ignored in the discussions of this document, this aspect must be taken into account when references to resources or metadata descriptions are made.

Resources may be further processed through processing chains where the result of each processing step is consumed by a subsequent step. The intermediate and end results of these processing chains are stored in the user’s workspace first and must be accessible throughout the processing chain.

Obviously we need centers that can offer workspaces and computational capacities to allow arbitrary researchers to execute the offered services. It is obvious that there are currently no structural solutions for this, since this is coupled with large investments if we think of a cyberstructure scenario with many users. In the CLARIN operational phase this issue needs to be solved probably in collaboration with the high performance computer centers in Europe for example. This collaboration will also need to come to solutions of how to quickly transfer large amounts of data to the processes or to install and execute software at remote places. Big companies are implementing cloud computing techniques - basically a revival of the central computer idea where a smart dispatcher software distributes requests to a large number of cores. CLARIN will seek to establish a close collaboration with initiatives such as DEISA [DEISA] and EGI [EGI]- both working on grid-based technologies - to come to convincing solutions.

5.1.2 Metadata infrastructure

In a number of meetings CLARIN experts discussed the more flexible CLARIN metadata infrastructure (CMDI) which comprises a number of aspects: (1) the elements that are necessary to describe language resources and tools/services; (2) the component framework specifications and (3) an infrastructure including all aspects such as editors, portals, search engines, browsers etc. These discussions which are based on almost 10 years of experience in the field and beyond from initiatives such as Dublin Core [DC], OLAC [OLAC], IMDI [IMDI], DFKI Tools Registry [DFKITR], TEI header elements [TEI] and LREP [Broeder2008] resulted in a requirements specification document [CLARIN-2008-4] which is the agreed basis for all metadata related activities in the CLARIN realm. The most relevant specifications are summarized in the Metadata Short Guide.

In this context it is not the task to repeat all statements of the requirements specification document. However, it is relevant to look into the details of the description of resources and tools/services and to see in how far they can be used to serve in the area of web services. Two points need to be mentioned in this context:

- the authors made a careful comparison of the suggested elements with the suggestions included in UDDI for example

---

53 Berlin, October 2008
54 Athens, April 2009
55 Oxford, February 2009
56 Tübingen, April 2009
58 UDDI is addressing an unspecified community and therefore must provide fairly complex and generic mechanisms while CMDI can be much more restricted towards the direct needs and terminology used in the LRT domain. Therefore,
• the currently suggested element set can be extended due to the flexible component structure dependent on the needs of the infrastructure to be set up

The elements that have been specified so far can be used to describe media resources, text resources, annotations, lexica, lists, tools and services. A number of main structural parts were worked out that should be provided to describe various aspects of the resources and tools/services: metadata self, creation, access, content, participants and resources. These descriptions cover administrative, functional and inspectional elements, the details are available through the CLARIN site. In this context only those elements are of direct relevance that describe functional aspects that are needed for example to do profile matching, i.e. to check whether a resource can be processed by a tool or web service as indicated in the following figure.

This figure indicates the principle of profile matching. A resource can be consumed by a succeeding processing step if the functional characteristics of the resource description map with those that are specified for the input of the tool or web service. The tool or web service will create additional metadata so that for the next processing step the same argument holds.

Figure 8: Principle of profile matching.

In the following table we want to briefly indicate the elements that have currently been identified as being most relevant in this process.

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceFormat</td>
<td>Specifies the annotation format that is used since often the mime type will not be sufficient for machine processing.</td>
<td>this is at the heart of the IE problem, if mime type etc don’t work than this field could be used</td>
</tr>
<tr>
<td>AnnotationType</td>
<td>An indication whether the annotation was created inline or in a stand-off fashion.</td>
<td>could be relevant for some tools although in future all tools should operate on both types</td>
</tr>
<tr>
<td>ApplicationDomain</td>
<td>An indication of the application domain for which the resource or tool/service is intended.</td>
<td>could be relevant as a rough indicator</td>
</tr>
<tr>
<td>CharacterEncoding</td>
<td>Name of the character encoding used in the resource or accepted by the tool/service.</td>
<td>is obvious</td>
</tr>
<tr>
<td>CreationTool</td>
<td>An indication of the tool with help of which the resource or the annotations in the resource were created.</td>
<td>this is even more detailed than the &quot;AnnotationFormat&quot; descriptor</td>
</tr>
</tbody>
</table>

it was chosen to check the completeness of CMDI based on the UDDI definitions, but to use more simple constructions. Since UDDI is an OASIS standard it was agreed however, to create UDDI descriptions from CMDI based records. The relation is basically similar as between domain metadata descriptions and DC for data resources where gateways are already in use for years.

59 http://www.clarin.eu/view_datcats
<table>
<thead>
<tr>
<th>DeploymentTool</th>
<th>An indication of a specific tool that may be used for the deployment of the resource.</th>
<th>not really relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>LanguageID</td>
<td>Identifier of the language as defined by ISO 639 that is included in the resource or supported by the tool/service.</td>
<td>check whether the language is supported</td>
</tr>
<tr>
<td>LanguageName</td>
<td>A human understandable name of the language that is used in the resource or supported by the tool/service.</td>
<td>check whether the language is supported - although ID should be used</td>
</tr>
<tr>
<td>LanguageScript</td>
<td>Indication of the writing system used to represent the language in form of a four letter code as it is defined in ISO-15924.</td>
<td>of no relevance I guess, since this has to do with visualization</td>
</tr>
<tr>
<td>MediaType</td>
<td>Specification of the media type of the resource or the media types the tool/service is suitable for.</td>
<td>it is a rough indicator on which media the tool is operating</td>
</tr>
<tr>
<td>MimeType</td>
<td>Specification of the mime-type of the resource which is a formalized specifier for the format included or a mime-type that the tool/service is accepting.</td>
<td>this is at the heart of the I/E problem, in many cases the mime type should be sufficient for format matching</td>
</tr>
<tr>
<td>Modalities</td>
<td>A listing of all modalities that are contained in the recording such that they can be subject of analysis or that are supported by the tool/service.</td>
<td>check whether the tool supports certain modalities</td>
</tr>
<tr>
<td>Resolution</td>
<td>Specification of the spatial resolution of images or movies.</td>
<td>some programs might have restrictions</td>
</tr>
<tr>
<td>Samplerate</td>
<td>Specification of the sample rate that is used for the recording.</td>
<td>some programs might have restrictions</td>
</tr>
<tr>
<td>Tagset</td>
<td>Specifies the tag set used in the annotation of the resource or a used by the tool/service or it contains a URL that points to the information about the tag set.</td>
<td>could be relevant to check the content of the annotation</td>
</tr>
<tr>
<td>TagsetLanguage</td>
<td>Indicates the language of the tag set itself, expressed in the two-letter language codes of iso639.</td>
<td>could be required by some tools</td>
</tr>
<tr>
<td>AnnotationLevelType</td>
<td>Specifies the types of annotation tiers provided by the resource.</td>
<td>could be interesting to quickly check whether the resource includes appropriate information</td>
</tr>
</tbody>
</table>

Table 3: List of relevant data categories for profile matching

Thus we have a set of elements that can specify in increasing detail which format and encoding is used:

Media-Type - Mime-Type - ResourceFormat - CreationTool - Tagset (in case of texts)

Whenever specific parameters were used when creating the resources the provenance information needs to be used (see below). At the Tübingen meeting\(^\text{60}\) where almost all active developers were present this set was checked against the requirements and until now it seemed to be covering all needs. This is however an extensible list which means that any new data categories that need to be taken into account may be added to this list at a later stage.

For all activities CMDI is the basis, i.e. it will be used to search or browse for tools and services and for workflow operations as indicated. Metadata descriptions need to be self-standing stand-off objects with an own persistent identifier (PID) according to the CLARIN requirements [CLARIN-2008-4] to allow people to register them and to carry out manipulations with them such as creating workflows. In the same way as for resources the metadata descriptions for all tools and web services need to be open and to be made available by the tool/web service providers to allow arbitrary service providers to harvest the descriptions via accepted protocols such as OAI-PMH [PMH].

The various web-services projects being carried out mostly via national funds in CLARIN need to be used to evaluate the element set and its vocabularies to allow final conclusions for the construction phase.

5.1.3 Provenance solutions

Provenance refers to the origin or source or the history the creation process of a resource. For scientific research provenance data must be documented in a sufficient manner to allow reproducibility of the results. Such provenance data is of primary importance to CLARIN as data becomes available through orchestration of services. It seems clear that in order to reproduce these processing chains all relevant data must be described.

\(^\text{60}\) Tübingen, April 2009
Although a number of provenance systems, for example PASOA and KARMA may be used to assist in capturing provenance data there is currently not yet a specification for the provenance data within CLARIN. The minimal set of provenance data for workflows will at least contain the process type and the initial input variables specifications and values. For each individual service or tool at least the following information must be recorded:

- Service Identifier. The Service Identifier is a PID referring to the metadata record of the service.
- Binding and Port information. These refer to the message format, protocol and endpoints of the service. Services may expose multiple of these.
- Operation Identifier. The Operation Identifier refers to the operation that was performed to create the resulting resource.
- Parameter Information. Parameter information contains all relevant information related to the parameters used. For parameters that refer to resources the metadata PID for the resource may be used. This metadata contains all relevant information on actual data and functional characteristics of the resource to allow re-invocation and reconstruction of the profile matching process at a later stage.

A pointer to the provenance data is included in the CLARIN metadata description data structure, as shown below, through the JournalFileProxy class.

A resource can be a simple resource (terminal) or a remote collection (MD) description.

5.1.4 IPR Issues and Business Models
IPR issues at the level of web-services cannot be ignored by CLARIN. In particular we see two major aspects:

- web-services need to access resources and to do so need to show who the requester is to determine whether access permissions were granted to this requester. IPR related issues will determine authorization for resources that a web service accesses on behalf of a specific user.
- CLARIN will include web services that may also be associated with restrictions determined by IPR related issues.

Both aspects need to be dealt with at run-time since workflows could be shared by various people, i.e. when users are entering the CLARIN web-services domain via a web-application they need to authenticate according to the federation principles. The web applications and services need to ensure that user credentials are passed through between them. Some services will create CLARIN infrastructure resources such as
registries and therefore have intrinsic credentials. In 5.1.5 there is a description of the SLCS approach that will help to overcome the current bottlenecks. Also the Grid frameworks have worked on solutions to tackle these problems. However, their main approach to use a Grid-Account cannot be applied in a broad infrastructure such as CLARIN.

It may be even the case that users that access resources via web-services such as for example a lexicon that is requested to serve certain attributes of a lexical entry or that are making use of certain algorithms such as a finite-state-transducer for morphological analysis packed into a web-service need to pay some fee for the operation. CLARIN cannot exclude these scenarios as well since some resources will be offered for example by companies or research institutions that need to earn some return on investment.

In the preparatory phase CLARIN will not tackle these issues with the intention to come to a professional solution. However, we will make some developments in collaboration with the Dutch grid project and carry out some tests. These issues will be specified in more detail in the construction phase when the results of the tests have become clear and when hopefully other communities such as EGI or DEISA have improved modules to offer which we can more easily integrate.

5.1.5 AAI solutions

Context

The general CLARIN AAI infrastructure for access to resources and applications is to be based on federated identity based on SAML 2.0 such as is implemented in middleware as Shibboleth [Shibboleth] and SimpleSAMLPhp [SimpleSAML]. This infrastructure assumes that for every user there exists a record at his home organization containing all user attributes that can be relevant for any authorization decision. Furthermore such a home organization will run a so called identity provider (IdP) middleware, that is able to authenticate the user and create SSO sessions. The IdP can issue SAML 2.0 assertions in response to queries about a user’s attributes. A resource or application is controlled by a so called service provider (SP) that will enforce an access policy of (1) forcing yet unauthenticated users to authenticate with their IdP to create a web container session and (2) querying the user’s IdP for the user attributes necessary for the authorization decision. The authorization decision lies solely with the SP. This procedure has been extensively described in [CLARIN-2008-1] and [CLARIN-2008-4] and must be the frame of reference for all AAI solutions chosen for web services and workflow systems to avoid interoperability problems and duplicated work.

Web services

There exist extensive specifications for web service security (WSS) with different possible solutions. In view of the above we will concentrate on the possibility to use WSS solutions with SAML assertions and certificates.

Workflow systems

Web services will be used (mainly) in complicated scenarios of workflow specification and execution. It is expected that those scenarios will stretch the AAI requirements to the fullest and should therefore be the main inspiration for the analysis. An example of a workflow scenario is shown in Figure 10.

The user authenticates with a web application that allows him to execute a previously specified workflow. We assume that the web application controls a workflow engine that (1) has knowledge about the authenticated user and has to perform all the work. (2) The workflow engine has the responsibility to contact and direct all the web services needed in the workflow. It has to do that such that the authority of the user is “delegated” to the different WSs because a web service may have to access resources that are only available to the user or a web service may itself be a composite web service and so need to act as a workflow engine itself.

This scenario leads to the following requirements for an AAI system.

- The web application controlling the workflow engine functions as a SP and allows federated login. This is relatively easy to realize.
- The workflow engine can send messages to other web services that assert, with sufficient certainty that the workflow engine acts on behalf of the user.
- Every web service is then itself capable of performing the same action again: delegating the authority of the user.
The last two requirements can be achieved in a number of ways.

1. As a first option it is possible to enforce an “always trust the web service” rule. This means that any registered web service should be trusted if it claims to act on behalf of a specific user. Web services identify each other by means of server certificates but the user identity itself is not proven in any way except when the user logs into the workflow engine controlling application. This can be a solution for a relatively limited number of web services, but it is clear that this is not a scalable solution.

2. Secondly it is possible to embody the identity (and thus the authority) of the user in a user certificate that is either uploaded by the user to the workflow engine or, and that is more easy and appropriate, is generated by a Short Lived Certificate service. This certificate is then propagated from web service to web service.

3. Finally it is possible to use SAML assertions especially the Relayed-Trust (RT) SAML assertion. In this scenario the workflow engine will use the original authentication assertion it obtained from the user’s IdP to build a RT SAML assertion that is specific for itself and the web service it needs to
access, and use that to access the WS. Every WS needs to build its own RT assertion for every other web service it needs to access (see Figure 11).

The above scenario assumes that the workflow engine is directly controlled by the web application the user logs in to. However it could just as well be implemented as a separate web service making it necessary for the web application to construct its own RT assertion and use it to call the workflow engine web service.

5.1.6 I/O Aspects

Metadata aspects

As with resources, metadata descriptions of services are described in stand-off metadata files. Service metadata can be decomposed into functional and non-functional metadata. Functional metadata describes the information about an operation, non-functional data describes information about the service. The latter for example includes publishing information on organizations, individuals and licensing schemes. Functional metadata is of relevance for example for profile matching (see 5.1.2). Figure 5 indicates the principle of profile matching. A resource can be consumed by a succeeding processing step if the functional characteristics of the resource description match with those that are specified for the input of the tool or web service. In order to be able to automatically check the applicability of a sequence of web service operations to a given resource it is necessary to have a formal description of (i) the resource properties, (ii) the input requirements of a service operation and (iii) the output characteristics of a service operation. Table 3 summarizes the data categories that are considered relevant to the profile matching process.

Services consist of operations which have input and output parameters. To a large extent functional metadata is described using WSDL or WADL files which describe services, operations and input/output parameters. These files may be stored as part of the service’s metadata specification to allow access to the depositor’s original interface specification for clients who need these, for example for client stub generation. At the parameter level however these specifications lack the level of detail that is needed to allow for profile matching using the data categories from table 3. The Language Identifier service from RACAI, for example, (described elsewhere in this document) consists of a single operation IdentifyLanguage which requires the text for which the language is to be identified to be UTF-8 encoded. This information is only available through the service’s documentation, but is not represented at the level of the WSDL file. Other examples of these types of preconditions are tag sets used within resources or languages of resources that are expected by the service. Some services, typically those that take part of processing pipelines require preprocessing steps to have been taken place before the resource can be processed. An example of these are part of speech taggers which require a resource to have at least been tokenized beforehand. These preconditions need to be expressed in terms of the data categories presented in table 3. Services which produce this type of output must advertise this on the output parameter producing this result through the same data category.

To describe these types of characteristics basically two options present themselves:

- Annotate WSDL/WADL files to describe these characteristics. This is comparable to approaches taken on WSDL-S\(^\text{61}\) or SAWSDL\(^\text{62}\) where WSDL files are further annotated to provide additional layers of information.
- Describe the data category pre/postconditions in a separate document. The most appropriate place for this is the service’s meta data document. All relevant interface specification information may be copied into CMDI and parameter specifications are extended with appropriate data category characteristics.

\(^\text{61}\) http://www.w3.org/Submission/WSDL-S/
\(^\text{62}\) http://www.w3.org/TR/sawSDL/
The Language Identifier service also reveals another requirement that must be taken into account here. The parameter for the `IdentifyLanguage` operation is a complex XML type, consisting of the text and two boolean parameters which indicate whether the language is a rare language and/or a modern language. The latter are configuration parameters for controlling the service behaviour.

Profile matching is achieved by matching the resource’s functional metadata characteristics against service operation input/output parameter functional metadata which are expressed in terms of data categories presented in table 3.

The profile matching mechanism developed within the German D-Spin demo study takes a similar, though in some essential details different approach. It assumes that services consist of a single (REST) operation which takes zero or more XML documents as arguments. Only a limited number of document types are accepted; `TextCorpus`, `Lexicon` or `CorpusQuery`. Each document is associated with a set of attribute-value pairs which encode the functional metadata properties of the document. Information such as language, tag sets used or annotation level are encoded as inline metadata. In this respect the D-Spin demo study approach deviates substantially from the manner in which CLARIN metadata is constructed, which is stand-off meta data. An example of a `TextCorpus` document containing only raw text, which is indicated through the top-level attribute `text="yes"` is given below:

```
<TextCorpus text="yes">
  <text>Peter isst Pizza</text>
</TextCorpus>
```

The `text="yes"` attribute-value pair may be expressed through the data categories from table 3 as:

- `/CharacterEncoding` = „UTF-8“: Name of the character encoding used in the resource or accepted by the tool/service.
- `/MimeType` = „text/xml“: Specification of the mime-type of the resource which is a formalized specifier for the format included or a mime-type that the tool/service is accepting.
- `/ResourceFormat` = „TextCorpus“: Specifies the annotation format that is used since often the mime type will not be sufficient for machine processing.
- `/AnnotationLevelType` = „text“: Specifies the types of annotation tiers provided by the resource.

In the D-Spin demo study service metadata is expressed in an input/output specification (IO-Spec) in the following form:

```
<IO-Spec>
  <service>Language Guesser</service>
  <input>
```

---

**Figure 12: Service data model.**
The input specification contains a TextCorpus element. Looking at table 3 this corresponds to the ResourceFormat = "TextCorpus" from the CLARIN data category list. Please note that the D-Spin demo study has taken the option to describe the IO-Spec separately rather than annotate the WSDLor WADL file.

Within the D-Spin demo study, profile matching is achieved by comparing a resource's document type and functional attribute-value pairs against the input specification of a service or by comparing the output specification of a service against the input specification of another service for pipeline processing.

The choices that were made within the D-SPIN study were motivated basically to make quick progress. Making the architecture CLARIN compliant, i.e. handling separate objects for metadata and provenance data can be achieved in the construction phase.

The need for richer semantic specifications of web services, flexible automation of service orchestration and well founded semantic reasoning about services has given rise to a number of other approaches which lie in the domain of Semantic Web services such as OWL for services (OWL-S), the Web Services Modeling Ontology (WSMO) and WSDL-S. These have in common that they employ an ontological approach, which is currently not envisaged here.

Interface harmonization

For certain types of web services it becomes advantageous to strive for uniform interface specifications to enable replacement of these uniform services in a pluggable manner. A potential candidate here are tokenizers, which all essentially perform the same task. Currently, the interface specifications for each of the tokenizers is proprietary in nature which makes it impossible to easily replace one tokenizer by another without additional considerations with respect to operation names and parameters used. Unification of interface specification for these types of services allows for pluggable services. These services may be easily swapped by the end user. For LMF, MAF an SYNAF systems a start with this has been made as a result of the LIRICS project. Further harmonization of these interfaces is to be achieved through further collaboration with tool builders.

5.1.7 Semantic interoperability

Workflow chains of operations as described in this document require in addition to syntactic interoperability, i.e. the knowledge about the structural aspects of a resource which is input to the subsequent operation, an understanding about how to interpret the content that can be found. The content can appear in various forms:

- in structured documents as tags indicating types such as in lexica the names of the lexical attributes or in annotations the names of the annotation levels
- as encodings within (values of) such attributes or annotation levels
- or as prose texts in arbitrary structural elements which even can be a text in a newspaper only tagged by formal characteristics such as "paragraph" or "chapter".

In the context of this document we only refer to the two first forms of semantic interoperability. The third aspect is being tackled by so-called upper ontologies, wordnets or the many domain specific ontologies that can help to relate arbitrary words with each other as they appear in prose texts. In the context of concatenated web services it is of course important that algorithms will find the required information for example about morphology and that they can interpret the encodings. Of course, we cannot expect that there is any agreement with respect to the terminology used.
In an "open scenario" as is given in CLARIN\(^3\), harmonization and mapping is a hard problem and is very much depending on the intentions of the researcher. We need to differentiate between three scenarios: (1) Tools that allow users to create new linguistic encodings should offer a standardized vocabulary so that harmonization can occur from the beginning. In the case that a researcher is convinced that the existing encodings are not appropriate he/she should be asked immediately to register his new data category and give it a proper definition. This will allow others to make use of it and to relate it to other existing categories dependent on the goals. (2) For all the legacy resources which are already out there it is more difficult to achieve interoperability. On purpose the experts have chosen their set of tags, but often the definitions are not made explicit. We can partly only guess their definitions and then create schemas and offer relations to registered categories. Often these relations will not be acceptable for other experts, thus a flexible framework needs to be made available. In most cases the used tags should not be registered, since their naming etc is completely idiosyncratic and would spoil any registry. (3) Sub-communities will continue with their frameworks which includes in some cases also some agreements about categories. As an example we can refer to a large community of child-language researchers worldwide that make use of the CHILDES/CHAT framework. CLARIN needs to respect these practices and look for methods to include them in a flexible way, since they have their own dynamics.

Currently we can see three initiatives working on methods to offer a framework for semantic interoperability at the tag level: TEI, GOLD and ISO TC37. TEI offers encoding principles for many areas in the humanities that can be embedded in recursive structures. GOLD developed an ontology which covers a number of linguistic concepts and their relation between each other - all expressed in RDF notation. ISO TC37 standardized a model for registering data categories (ISO 12620) that require to enter proper definitions and that has a number of features such as language sections all facilitating re-usage in different language environments. On purpose ISO does not include relations since they are often very much disagreed amongst the experts and often are dependent on practical intentions. A flexible framework for manipulating and sharing relations needs to be provided which is not subject of standardization. Currently, we see a trend towards harmonization between these three approaches, since TEI will make its categories referable, i.e. act as a data category registry and since the GOLD categories will be entered into the ISO DCR.

Yet these interoperability frameworks are new, i.e. we cannot refer to experiences and a number of questions such as for example semantic granularity and constraints are still subject to discussion. Also, adapting all types of legacy resources so that they can participate in interoperable scenarios in the sketched way is a rather time consuming effort. Thus CLARIN is aware of these problems so that other pragmatic approaches of achieving semantic interoperability need to be accepted as well.

5.2 Web Services in CLARIN

Invocation of web services is considered to be data driven, i.e. users will generally supply a resource they want to invoke a service on and are interested in (part) of the information that results from the service invocation. In CLARIN all resources are associated with separate metadata which may link to provenance data if resources are the result of previous processing steps. After service invocation a new resource is created which is again to be associated with appropriate metadata. Gathering of metadata and provenance data are crucial elements of service invocations to provide well described, traceable resources. The figure below shows a simplified overview of this process. Figure 16 provides a more detailed interaction diagram covering components and interactions involved.

The ability to generate metadata and provenance data for service requests relies on the following components to be part of the CLARIN infrastructure:

- Metadata generation component, generates metadata for the generated resources

---

\(^3\) The "open scenario" of CLARIN can be characterized in the following way: (1) The users that will make use of CLARIN services cannot be predicted, since there are so many potential users in particular in the humanities and social sciences. (2) The applications that they want to carry out cannot be predicted, i.e. only occasionally we can speak about typical production scenarios where a specific type of application is repeated frequently. (3) The chains of processes will be very heterogeneous with respect to their nature and their components, i.e. all user interfaces and tools need to offer this flexibility. Therefore the approach CLARIN needs to take is complementary to the approach of Virtual Research Environments where a specific type of scientific application is the source of motivation. (4) Metadata for example serves many different functions in such an open scenario and therefore must be registered in a stand-off manner.
- Provenance data generation component, generates provenance data for the service invocation
- Transformers, transforms data between different resource formats, tag sets and annotation level types.

The **metadata component** is responsible for gathering metadata that will describe and point to the resulting resource. The new metadata description will make use of information taken from the original resource description, which is then supplemented with additional information gathered during the service invocation. This will include all relevant information that may be included from the service’s metadata and processing information and it may contain both functional and non-functional metadata. In situations where multiple primary resources serve as input for a service operation the metadata for each of these may result in a combination of various metadata elements.

Some input parameters that are used to invoke services are different from others in the sense that they refer to data resources. At the process of service invocation the PID of the metadata description will be handed over. This will allow the metadata component to get the metadata description which has all information about the real resource including a pointer to it for processing purposes and a pointer to provenance data. The metadata component must also have access to the service’s metadata specification to obtain the output characteristics of the resulting resource. Parameters needed for invocation of workflow chain must be specified in a project description which is associated with the workflow chain. Therefore it is the responsibility of the metadata component to retrieve all information that is necessary to invoke and carry out the service and at the end to generate a new metadata description that refers to the resulting resource of the invocation and the updated provenance resource.

---

46 If the resource is located in a temporary workspace then the identifier is not a persistent one, so technically speaking it is not a PID here, but a transient identifier.
The **provenance data component** will capture all relevant information that is necessary to provide traceability of the resource. Provenance data will at least include the identifier of the service, the operation being invoked and the parameters used to invoke that service with. The trace information should be strong enough to be able to replay the entire service invocation at a later stage (reproducibility). For the moment it is assumed that the services which are used are persistently available, something which in practice will need to be looked at very carefully. Services tend to change over time, something which can be dealt with by including versioning mechanisms at the web service metadata registry. New releases of the service thus result in new versions of the original service's meta data specifications. Services may however also be decommissioned, which will no longer guarantee reproducibility of the results.

**Transformers** (see figures below) convert resources into formats that are acceptable as input parameters of service requests. The large variety in input/output formats is forming the greatest obstacle for allowing users to easily establishing processing chains, since this area is heavily underspecified. We need to distinguish the formal structure aspects from the linguistic encoding aspects. With respect to the structures we can see that powerful representational schemes are emerging which CLARIN should adopt if they turn out to be stable and accepted by the community. For lexica the Lexical Markup Framework has been standardized by ISO. For complex annotation structures quite a number of suggestions have been made from rather abstract ones to more detailed ones. Two communities play a role here; those that annotate streaming data such as audio, video or time series and those that annotate texts. The discussions in these areas have by no means settled yet. CLARIN will observe the further standardization and verification process in this area.

Of course still many other legacy formats are being used which brings up the question of how to do transformations. The principle is schematically indicated in the following two figures. The usage of pivot formats would reduce the number of transformers needed.

![Figure 14: Transformations using point-to-point connections](image-url)
Services usually produce information in a proprietary format (structures and encoding) which is at the heart of interoperability problems. It is obvious that using pivot formats will be the only feasible way to go for CLARIN knowing that there will not be just one but a few that will be supported. We cannot expect that all existing code is being rewritten and that all resources are re-formatted, but what we can expect is that there will be transformers that transform legacy formats into these recognized pivot formats. Another transformer will take care that a follow-up service can consume the newly created resource.

In addition to the structural aspects the transformation at the linguistic encoding level needs to be solved. Here we can again distinguish between two layers: (1) the terms that different linguists and their tools are using and (2) the pattern in which the encodings are written into an annotation layer. As an example we can refer to the output of tokenisers or the way syntax trees are encoded in brackets. Here we still lack a good overview requiring a deeper analysis as it is planned in work package 5. An eScience seminar of the Max Planck Society and a number of IEEE eScience conferences revealed that these problems are most difficult to solve for many disciplines. The heterogeneity is enormous and due to the scientific dynamics we will be confronted with new linguistic encoding types in the future as well.

The following figure illustrates the interaction between the service requester and the web service. As can be observed the metadata and provenance components are directly involved in this interaction and are called through a Web Service Wrapper. The Web Service Wrapper combines in the basic service interaction figure shown in figure 13. Transformers may be modeled as separate services or may in some cases be included in the Web Service Wrapper as well. The diagram below assumes the first.
The interaction pattern between the Service Requester and the native web service is as follows:

1. The Service Requester calls the Web Service Wrapper. This delegate transfers all incoming requests to the web service. The embedded web service is therefore never called directly.
2. For parameters that are associated with resources Metadata PIDs are supplied. This will allow the Web Service Wrapper to access the metadata of all provided resources and prepare an initial copy of the metadata to be used for the returned result (Step 12 to 15).
3. The resource's metadata is collected and prepared for further processing in steps 12 to 15.
4. The resource data is loaded through the resource link extracted from the metadata description.
5. The returned resources replace the content of the primary input for which the metadata PID was supplied.
6. The provenance data is prepared and stored. The minimal amount of provenance data consists of:
   a. The service identifier.
   b. The operation identifier.
   c. The parameter information. If a parameter refers to a resource's content then the metadata PID is supplied
7. An acknowledgement is sent.
8. The embedded web service is invoked.
9. The web service result is returned. If the request is an asynchronous one then the Web Service Wrapper must make a replyTo address available which will further process the response before returning it to the original web service caller

Figure 16: Interaction diagram for web service invocation.
10. The result from the request is stored as a separate resource to provide traceability. If a number of web services are called in sequence this provides the opportunity to store intermediate results for each of the actions. The final result is stored in the CLARIN infrastructure, most likely as part of the user’s private workspace.

11. An acknowledgement is sent.

12. The provenance data for the resource is stored.

13. An acknowledgement is sent.

14. The metadata for the resulting resource is stored, most likely as part of the user’s private workspace. This metadata may partly be obtained through (combination of) the original metadata of the parameter resource(s). Metadata, provenance data and resources are associated with temporary identifiers, if these are stored in the user’s private workspace, which may be converted to PIDs when this data is persistently stored in archives.

15. An acknowledgement is sent.

16. The result is returned to the original caller.

CLARIN will deal with large amounts of data and also with time consuming processes. In order to avoid timeouts in processes and improve the user acceptance, CLARIN will offer ways for calling web services asynchronously. This feature will increase the stability of the system and its reliability as well, for instance, asynchronous web services will give the user the possibility to run in parallel other tasks while the first process is being done or to consult the results while the system is waiting for the confirmation, etc.

In order to reach those objectives, CLARIN will take into account that some tasks need asynchronous interaction and will offer the proper resources or components for handling the interaction.

Architecture considerations
There are a number of alternatives to implement the Web Service Wrapper, two of which are highlighted here:

1. Local Service wrapper. Each embedded Web Service is encapsulated by its own wrapper which is composed of all functional components related to pre and post processing to make it a CLARIN compliant service. This wrapper has component to deal with PIDs, meta data, provenance etc. For the tool supplier this means that in order for his/her service to become available to the CLARIN infrastructure this wrapper must be constructed and the Web service Wrapper, rather than the native web service is to be registered.

2. Enterprise Service Bus. An Enterprise Service Bus is a software infrastructure to accomplish interconnectivity between services. If the behaviour of in particular the metadata component and provenance component are generic in nature then these components can be embedded in an enterprise service bus. A Service Requester will call web services through a service bus rather than call the native web services directly. A service bus may be part of the generic CLARIN infrastructure, or organizations may run their own.

In both cases generic CLARIN functionality is provided with the help of some software functions. In the case of the wrapper every service provider needs to encapsulate his service to make it a CLARIN compliant web service. In the case of the service bus implementation these generic software functions are installed at a few places from which the individual services are invoked, i.e. the service bus provides centralized functionality which has advantages due to simplified software deployment and maintenance. For a more detailed discussion in case of composite services see below.

From an architectural perspective transformers may be treated as separate web services or as components embedded in the Web Service Wrapper. The first option requires transformers to be described in the same manner as other web services, i.e. formal metadata must be provided as described in 5.1.4. (meta data aspects). Alternatively, transformers may also be associated directly with the Web Service Wrapper. The Web Service Wrapper is registered in the web service registry instead of the native web service. This latter strategy is pursued in the D-Spin demo project. Since we expect in the open CLARIN scenario that there will be many different transformers for various purposes that may be integrated in various contexts only the atomic provision of transformers will finally make sense. Again the D-Spin demo project decided to use this option since it offers a high compactness and thus advantages in fast implementation and testing.
5.3 Workflow systems in CLARIN

5.3.1 CLARIN Service Bus

The CLARIN infrastructure is about providing language resources by following well known SOA principles. The language resources are hosted in CLARIN participating centres all over Europe and include data resources, services and tools. Their comprehensive description will be stored in centralized metadata repositories that are aggregated via well-known protocols such as OAI-PMH to central registries an instance of which is called a CLARIN Registry. Every participating centre should be, as much as possible, independent in its choices of internal organisation and setup as long as it adheres to the agreements that are defined for a smooth interaction within the network.

This smooth interaction should follow relevant standards and span interaction people, tools and data in a distributed fashion.

This section will discuss the main requirements for a CLARIN infrastructure related to services, their orchestration and other important workflow aspects. On that subject enterprise service bus (ESB) technology will be briefly described along the explanation how it can become a backbone of the CLARIN infrastructure.

Currently, CLARIN provides various services for NLP, Text analysis, language translation, multimedia annotation etc. There are however several issues with directly exposing these services to the CLARIN users:

- Every service serves a particular purpose, while CLARIN users might want a composite service. For example, a user might require language translation, Part of Speech tagging and parsing along his route. This can be implemented in several steps – first translate a resource into the specified language, then perform POS tagging and at the end find subjects, predicates and objects in the text. This sequence of steps constitutes a new composite service that should be created easily based on user requirements and which could be registered and described by appropriate metadata.
- It is expected that the number of CLARIN services will grow each using different data models/ definitions, which makes usage and orchestration complex.
- Different users might require content/services delivery via different protocols and in different manner. Some might want the ability to access services using HTTP via SOAP or REST, some would prefer FTP or maybe something else. Also, some services are synchronous, while others are asynchronous…
- There will be several services that perform very similar operations; hence user should be able to select them according to his/her preference.
- All services need to employ a common security model with the possibly restricted access
- Service reliability and timely response are very important

A middleware that enables the service integration in an easy manner serving the above requirements is essential to delivering such systems. Here we will present how technologies like Enterprise Service Bus (ESB) and Workflow Management System WFMS can be used for building such middleware, which we will name CLARIN Service Bus (CSB).

Role of the Enterprise Service Bus (ESB)

ESB is a framework that helps to reduce the complexity and costs associated with distributed software systems. It is especially suitable for performing the following tasks:

- Service Level Agreement (SLA) and policy management
- Security management
- Protocol reconciliation
- Message transformation
- Orchestration (possibly, in conjunction with a BPM engine)

According to Wikipedia Enterprise Service Bus is a convenient catch-all term for a set of capabilities, which can be implemented in different ways. There is considerable debate on whether an ESB is a tangible product or an architectural style and on exactly how an ESB may be implemented.
• Integration
• Logging and instrumentation
• Provenance

An ESB can perform all these tasks in a consistent and centralized fashion, so that service providers do not have to take care of these for each service they are providing. Also, it solves the complexity problem by letting developers focus on the construction of native services and their associated business logic and not on integrating infrastructure logic which will help in providing a robust distributed system. This will make it easier to deploy services and achieve CLARIN compliance.

CLARIN infrastructure – global view at the architecture

As a result of the previous analysis, and adopting ESB model, one of the possible overall architecture for the CLARIN infrastructure is presented in Figure 17.

![Figure 17: CLARIN infrastructure](image)

The central place in the figure is taken by the CLARIN Service Bus (CSB), which implements the required services for exposed protocols by orchestrating both existing and future CLARIN services. It consists of three main layers:

- Wrappers – a layer responsible for accessing existing web services and bringing the result of execution (in the form of CLARIN data model) into the platform.
- Service implementation layer – a layer responsible for orchestration of existing services (accessible through converters) into required services.
- Consumer adapters – a layer responsible for delivering results of service invocations to different types of clients through different protocols/transport.

Services and messages

From an architectural point of view everything in the CLARIN Service Bus should be a service. This is not necessarily a Web Service, but CSB Service, which can be exposed through a variety of transport protocols. CSB services are different from CLARIN services, the first provide common services at the infrastructure level while the latter provide services that are typically used by end user applications, such as tokenisers or part of speech taggers. Services will exchange messages. A CSB message is similar to SOAP message and consists of several parts, including header, body, fault string, attachments, etc. Every part will
contain a collection of data objects that can be accessed by name. CSB services will have single method “doWork” and can be described using the following interface:

```
Message doWork (Message input);
```

Services are defined inside CLARIN Registry which includes information such as a service identifier and the input/output characteristics of the service. Reliability should be virtually guaranteed to the services built on the CSB. Additionally, the messages themselves will be normalized. This means that regardless of the initial form of the data sent between services, and the form of communication a particular system is designed to use, the CSB will transform those messages and adapt various protocols to work together.

**Implementation and deployment if CSB services**

CSB should define a service as an explicit pipeline that can contain an arbitrary set of actions (Figure 18), each of which implements either service logic or infrastructure support, for example data transformation or storing service messages in the message repository.

![Figure 18: CSB Service](image)

A CSB service developer, from some of the participating CLARIN Centres will decompose service functionality in a set of actions and then implement each one of them. Such a procedure of service implementation provides two levels of reuse – services themselves and actions can be reused across multiple services. A rich library of reusable actions should also be provided by the CSB implementation.

This should include at least:

- Metadata component
- Provenance component
- Transformers & Converters
- Workflow support
- Scripting
- Routing
- Web Service support

A service request can be delivered through a variety of transports. A service definition will include references to the set of transports and endpoints which that service listens on (this endpoint has to be unique for a given service). Besides that, service definitions should also allow for configuration of the number of threads which will be used for service request processing. A CSB service will contain:

- The action code
- The configuration files
- The library dependencies.

All this should be packaged in some sort of archive file. Service provider can deploy as many archives as he wants but also needs to be able to specify their deployment order, if the archives are mutually dependent. The idea behind the archive concept is that it is a single deployable unit. It should allow service provider to move
his service between servers simply by moving the archive file. This will be a big step towards reliability and load balancing. As an additional benefit this allows for fast distribution of the CSB services to other CLARIN partners who may use and test these in their own set ups. Configuration of the service listeners and service execution pipelines is done using straight XML format. CSB should however provide tooling to simplify the creation, configuration, deployment and maintenance of CSB services.

Service invocations in CSB
Services in CSB can be invoked either synchronously or asynchronously, the latter should be the default way. The service developer should be able to define the type of service execution through the metadata specification of the service at the CLARIN registry which is accessible from the CSB. On the other hand the service consumer should be able to define whether he expects a reply from the service or not, since services can also be divided into: “request/reply” and “one way” services. The CSB should make invocation of services easier, no matter what type of service, by providing ready-to-use components called Service invokers which expose a very simple interface, which are stored in the CLARIN registry as well. This means that CSB can be used to expose Web service endpoints for any of its services, either through SOAP or REST or other protocols. In the same way, CSB should provide support to dynamically generate web service clients which can call target services for either SOAP (by using service WSDL) or REST.

Fault tolerance
Fault tolerance should be one of the very important features of the CLARIN Service Bus. If we take “CLARINServiceA” for example and imagine that a participating CLARIN Centre has already deployed this service on Node1, what should happen if the same service gets deployed on some other node, let’s say Node2? Simply, during the process of adding this new binding to the CLARIN Registry, it will be found out that the service with the same name already exists on Node1, therefore a new endpoint reference (EPR) will be added for Node2. This will provide a fault tolerance mechanism: if Node1 goes down, “CLARINServiceA” Node2 will keep on working.

Channel fail-over and load balancing
To discuss channel fail-over and load balancing we can expand the previous scenario with “CLARINServiceA” deployed on Node1 and Node2 and configure “CLARINServiceA” to listen to more than 1 protocol (not just http, but also ftp, i.e.) on both Nodes 1 and Node2. When the CSB tries to deliver a message to our “CLARINServiceA” it would need to decide which endpoint reference (EPR) to use (there are now 4 choices (2 nodes x 2 protocols).

In that situation Policy configuration should be provided to support at least the following options:

- First Available. If a valid EPR is found it will be used unless it dies. This Policy should not provide any load balancing between the two service instances.
- Round Robin. Typical Load Balance Policy where EPRs are taken one by one, either by order or randomly, thus load balancing is an additional benefit.

5.3.2 Workflow
Workflows in CLARIN should be used for four general purposes:

- Service orchestration, i.e. building a flow of control around a process for its interactions with other partner processes.
- Human task management, i.e. handling of tasks that require human interaction.
- Creating reusable components, i.e. each workflow process definition may be stored and may be included in other workflow processes at a later stage.
- Process visualisation and authoring. This way the user may monitor process progress and may influence that manner in which the process is executed.

Service orchestration
At a glance, it might seem that there is a significant overlap between a service pipeline and service orchestration. A service pipeline indeed represents a sequential orchestration of actions. Like a workflow, it provides a unified control flow to the developer, since the main logic is defined in one coherent stream. But, the things that service pipelines do not support well are: decisions, conditional transitions and parallel execution, which are common orchestration requirements. Please see the test case description for TRIANA, KEPLER and TAVERNA in section 4.3.1 on the and section 4.3.3 for more details on these issues. They could
be implemented as a part of pipeline definition, but scalability issues must be taken into account here as a result of blocking processes. Also, for service orchestration, a very good process visualization is really a must, since it allows faster and easier application composition. It also makes the composed service logic more visible. Therefore, the idea is to use a service pipeline for bringing together basic business functionality and additional infrastructure support, including data transformation, execution monitoring, etc and use a Workflow Management System (WFMS) with an advanced Workflow Editor for orchestration of CLARIN services.

CSB/WFMS integration
There should be two types of CSB/WFMS integrations:

- Calling workflow processes from services (CSB to WFMS)
- Calling services from workflow processes (WFMS to CSB)

To be able to make a call from the CSB to a WFMS the CSB should provide a special action, which makes calls to a WFMS. This action will take a process definition name as a parameter and then: create and start a new process instance. The created process instance can be executed:

- Either asynchronously in a separate thread which means that the service will return the reply to the invoker while the workflow process is running.
- Or synchronously, which means that the result of the workflow process execution might be used as a service reply.

To allow communication between a WFMS and the CSB the WFMS should provide two special handlers for talking to CSB.

- A request-reply type WF Handler, which drops a message on a service bus and then waits for a response.

The envisaged scenario diagram for this type of integration is presented in Figure 20. The WF Handler will send a request message to the CLARIN Service on the CSB and put a process execution in the wait state. When an execution of a CLARIN Service completes, it will invoke a special CSB callback service, which signals the waiting process to continue.

---

66 An application can’t be locked into a single process on a single machine, if it aims for scalability. At any point during execution of a workflow process, the process execution can be interrupted and stored. Later the execution state can be retrieved from persistent storage and continued. This is essentially different from the service pipeline where the callstack is not persistable. Such capability of interrupting an execution, storing it, retrieving it and resuming the execution, without blocking thread resources can also be described as support for wait states.
An appealing aspect in this scenario is the asynchronous execution support for a process with multiple execution paths (tokens), e.g. in the case of fork/join or parallel paths and its notification of the WF Handler for one-way calls, which will only drop a message on a service bus and continue its process execution. The interaction with the CSB will be asynchronous in nature and will not block the process instance while the called CLARIN Service is executed.

Notification WF Handler for one-way call, which will only drop a message on a Service Bus and continue its process execution. The interaction with CSB will be asynchronous in nature and will not block the process instance while the called CLARIN Service is executed.

In general the term “workflow”, is used to describe traditional processing chains which in most cases are synchronous. However in most language technology applications, the algorithms are not smart enough to cope with all language phenomena for example, i.e. there will be steps requiring manual correction where humans need to be involved as part of the workflow. Human involvement has two main consequences. Firstly, humans will participate in workflow processes through the use tools. In general these tools will need to be able to:

- Accept a task to be performed by a human
- Get the relevant data (resource (fragment) and additional parameters) and
- Either return the result or signal cancellation of the task.

**Human task support**

This will require that tools participating in the workflow scenarios must support some type of service interface, so it partially behaves like a service. For an example of such a tool, please see Figure 1 which displays the Annotation Editor in Gate. These service interfaces will need to be registered in the CLARIN registry as well. Secondly, particularly the human/tool interaction may typically take a long time to complete. Our participant may just have gone on a 2 week holiday trip, which makes this type of interaction typically asynchronous. workflows system should cope with this very well since it should inherently support wait states where execution context is persisted in a database.
Provenance data in WFMS
Recording provenance data becomes simple for a WFMS since the Workflow engine knows the boundaries between workflow activities, it can easily determine when each of those activities starts and ends. Therefore providing automatic tracking of the workflow execution is straightforward. Many implementation ways are possible; one example is to log activities, with a database record written each time an activity begins and ends execution. It is also possible to track specific variables, i.e. the workflow state, which involves recording their values at various points during workflow’s execution.

Activity Reuse
Support for reusable custom activities becomes possible by considering a specific set of activities executed by the workflow engine as some sort of language. If the CLARIN community creates a group of custom activities that can be reused to solve specific problems across multiple research processes, this can be described through a domain-specific language (DSL). Once this has been done, it might be possible for less technical people to create workflow processes using these pre-packaged chunks of custom functionality.

Process visualization and authoring
The main requirement for a CLARIN workflow editor is that its target user base is the audience without a programmer background. On the other hand, the editor should support quick changes by the advanced users. Aiming to meet both extremes, a workflow editor must have very good graphic support with drag & drop mechanisms, include various kinds of wizards and library components and offer an intuitive workflow description format, which can be easily understood and modified.

Very good graphical support allows users to create workflow process definitions visually. Nodes and transitions between nodes can be added, modified or removed. The process definition can be saved as an XML document which can be stored on a file system and deployed to a Workflow Registry. The following components have to be integrated in CLARIN workflow editor:

- CLARIN Registry browser and search
- Contextual help
- The history of workflow execution which will be used for various functionalities such as:
  - Intelligent suggestion mechanism
  - Execution time estimation
  - Displaying number of successful executions

---

67 Actions and tasks in case of JBPM,
6. CLARIN activities

6.1 Spain

CLARIN got the support of the Spanish Ministry Ministerio de Educación y Ciencia for the development of a Spanish CLARIN Mockup late 2007. The Spanish CLARIN team at the Universitat Pompeu Fabra made a preliminary survey of the research projects in Humanities and Social Sciences in Spain. We contacted Spanish groups and researchers that were funded by the Spanish Ministry in 2007 and research groups at UPF. The objective of this survey was the analysis of the user requirements and the identification of potentially most used tasks to be performed by researchers in these fields.

We chose different real cases to define representative prototypical scenarios. These scenarios where the leading rationale that made us to chose particular NLP applications already developed as web applications to be transformed into web services. We distinguished two main groups of analysis tasks that had to be deployed as web services:

- Statistical and lexicometric analyses. These were found interesting by researchers from different fields who need to perform some kind of discourse analysis and need tools to deal with word distribution across corpus, word co-occurrence and different lexicometric measures.
- Advanced search tools using pattern matching techniques and classification tools for the corresponding results. These tools were interesting for those researchers who need to find samples of (complex) linguistic constructions and deal with large amount of data.

Most of this work has been later presented in WP2 and WP5 Clarin related activities. The list of web services that have been deployed and their characteristics have been described in section 4.2.6 UPF web services.

In addition to analysis tasks, researchers were also interested in tools that give them easy access to primary data. Very often their primary data are distributed and heterogenous which makes it difficult to locate and access them. We have been in contact with data providers in order to study and eventually implement ways to access distributed data. Again, two different scenarios have been analysed:

- Access to distributed indexed corpora. We are collaborating with three corpus providers for Catalan texts:
  - The Institut d’Estudis Catalans which owns the reference corpus for Catalan.
  - The Corpus Informatitzat del Català Antic from the Universitat Autonoma de Barcelona which compiles a corpus of old Catalan, and
  - The Diccióari del Català Antic from the University of Barcelona which also deals with old Catalan texts.
- Access to distributed digital newspaper libraries. Many researchers need access to digital libraries. In this case we are collaborating with the local archive of Girona.

In both cases the goal is to provide means to access desired (distributed) data and build a unified temporal corpus that fits the requirements of the researcher. At this stage we are developing two different types of web services:

- Corpus accessing tools that take into account criteria such as date, language and author.
- Corpus building tools that once all data are collected indexes a temporal corpus to be exploited by the researcher.

6.2 Germany

The project D-Spin (Deutsche Sprachressourcen-Infrastruktur) is the German partner of CLARIN. Within the D-Spin project a prototypical web-service infrastructure is being implementing including a range of web services which are either directly accessed or via a web application (tool chainer) for successively invoking web services.
At the moment, the web application offers LRT services that were developed independently at the Institut für Informatik, Abteilung Automatisierte Sprachverarbeitung at the University of Leipzig (tokenizer, lemmatizer, co-occurrence detector, and frequency analyzer), at the Institut für Maschinelle Sprachverarbeitung at the University of Stuttgart (tokenizer, tagger/lemmatizer, German morphological analyser SMOR, constituent and dependency parsers), at the Berlin Brandenburgische Akademie der Wissenschaften (tokenizer, tagger, NE recognizer) and at the Seminar für Sprachwissenschaft/Computerlinguistik at the University of Tübingen (format conversion for file upload, GermaNet, Open Thesaurus synonym service, and Treebank browser). They cover a wide range of linguistic applications, like tokenization, co-occurrence extraction, POS Tagging, lexical and semantic analysis and several languages (currently German, English, Italian, and French). For some of these tasks, more than one web service is available.

In order to facilitate data transfer between these individual web services, it became clear that one or more unique XML-based data formats need to be developed on short term. One such data format for encoding linguistic text corpora has been developed. The data format is flexible and can be extended to fulfill new requirements. It is already used by the currently available web services and the tool chainer used in the D-SPIN demo project. This data format will be made subject of discussion within D-Spin, CLARIN and standardisation bodies such as ISO.

A main goal when developing this data format was to enable semi automatic tool chaining. In order to allow the extraction of a very basic description of the linguistic data/content, all tool wrapper formats contain a header area in which the content is described more detailed. This interface description is given for both ends, input and output description, of the service. Based on this information an automatic chaining component is able to derive which services may be executed in which order. It is possible to validate process chains and also to generate proposals on which service may run after a couple of other services were already executed.

Additional ad hoc formats for transferring lexicons, dictionaries and corpus queries are under development by the D-Spin partners. The nature of these APIs makes it possible to access the web services from nearly any programming language, shell script or workflow engine (UIMA, Gate etc.). For all format specification activities it is evident that these will be brought in to a wider discussion about standardisation.

6.4 European prototype

It is the intention in CLARIN to work on a joint prototypical project including web services where groups from different European countries will participate. The proposed prototype should take current pillars for the CLARIN infrastructure into consideration such as centers, metadata, PIDs and federation technologies. Further it must make use of the work done by the CLARIN members and the Spanish and German CLARIN projects.

An analysis of the progress revealed that a combined metadata/content search covering resources from various partners covers many aspects and can be realized as a prototype demonstrating the power of the CLARIN infrastructure.

This prototype:

- Test out in a joint collaboration of the registry mechanisms and the web services specifications.
- Consideration of IPR issues since not all resources will be copied to one site that will then create a huge index, but the mechanisms provided by each repository taken into consideration.
- Test the federation principles, since ideally the researcher will be identified by his home organization and his credentials will be passed between participating organizations.
- Harmonize on the types of search engines interfaces which are out there in large quantity - in fact the current habitude is that almost each corpus comes with its own (user) interface.
- We will be able to extend this prototype that will first ignore structure by making use of the ISOCat concept registry to relate between tag sets. The latter will not be part of the prototype work.

In addition to the aspects that have been covered in this chapter and document already, such as service metadata, metadata and provenance data for service invocations, the following aspects deserve specific concern:

- **Asynchronous behavior**. Search results may take long to compose by individual services something which the distributed search will need to take into account.
• **Scope.** The number of resources to search may be limited by a metadata criteria such as the language of the resource to search through.

• **Paging.** Large numbers of results may be returned. In general a paging mechanism will be in place to deal with this.

• **Ranking.** There are no particular ways to rank the returned results.

• **Merging.** Is it feasible to merge the result sets that are being returned from individual sites in a coherent manner.

• **Statistical information.** Some services provide statistical information, such as the total number of hits. These should be returned as well.

• **Result set types.** Result sets may be returned in different manners, depending upon the type of resource that is being searched. For lexica it is customary to return the lexical entry information, for plain or annotated texts concordance information may be provided.

• **Temporary storage.** Result sets may need to be stored temporarily as they may represent starting resources for further work flow processes such as currently being worked out in the Spanish and German CLARIN projects. The prototype thus also serves as a test bed for dealing with storage, access and metadata of transient resources and resource collections in workspaces.

• **PIDs.** Resources which are identified by PIDs can be categorized into three major types[ISO24619]:
  - Independent resource
  - Any part of such a resource that requires further specification (fragment identifiers)
  - A collection of resources that is referred to as a whole

  Each of these types can be distinguished in the proposed prototype, resources, resource collections and parts of resources such as collocations. The prototype thus provides a test bed for dealing with these as well.
7 State and Evaluation
A number of activities have taken place since the first version of this report has been delivered:

- In Leipzig another workshop (November 09) has been organized (http://clarin.informatik.uni-leipzig.de/) that was focusing on web services and workflow mechanisms
- Another workshop in January 2010 in Amsterdam was focusing on demonstrator use cases where processing chains also played an important role
- At different centers much development work has been carried out where the major developments were driven by national funding streams
- A first metadata component for tools and services has been developed.

In this chapter we will first report on the activities and then give a summary and evaluation of the state.

7.1 Leipzig Workshop
First we want to report on the results of the workshop in Leipzig where we on purpose also invited external groups (also from industry) to understand how far we are and what others are doing. Cross-fertilization in this area of rapid development and continuous interaction about the many unclear is seen as very important.

WP2-WG2.6/2.7 Report
M. Kemps-Snijders (MPI Nijmegen)
It was briefly presented what has been discussed so far about a generic framework for an interoperable web services framework for CLARIN. Most important are: (1) the CMDI metadata framework that must be used to describe data resources and web services using the same elements and vocabularies to be registered in ISOcat; (2) a common framework for provenance data which is a specific kind of metadata which is of more dynamic type which is the main reason to not be mixed with the static descriptions of resources and tools; (3) acceptance of different communication protocols (REST, SOAP), interface specifications (WSDL, WADL) and transport protocols (HTTP, FTP). Generic wrapper functionality for implementing these interaction possibilities needs to be developed so that all CLARIN web services providers can make use of these components. An interface needs to be specified allowing web services providers to hook up to this wrapper functions. For maintenance reasons an implementation in form of a CLARIN service bus seems to be optimal.

Finding Services by using NLP services
I. Gurevych, R.E. de Castilho (TU Darmstadt)
In this work in which NLP processing chains are used to find useful services on the web by using formal description is part of the German Theseus TEXO subproject. The view in TEXO is that web services are tradable, composable from other services, offered by various parties and can automatically be executed. The goal is to establish a rich service marketplace with customized and personalized services that can easily be integrated into user environments. One of the key pillars of such an open market place is easy service discovery. Since manual tagging of services is seen as very time consuming the question is tackled whether useful discovery can be done by using formal WSDL descriptions and self-description which can be found in the programming code, documentation, blogs etc. applying NLP technology. Tests were made for large web services such as the Programmable Web with using WSDL descriptions. The results indicate the difficulties by using such techniques which are for example due to the semantic incompleteness (topics not mentioned) and semantic blurring (differences in terminology) requiring an ontology. In this work web service pipelines are used for web service retrieval and for web service indexing both including typical components. Both are being composed by using the UIMA framework.

Need for Data Cleansing
R.E. de Castilho, I. Gurevych (TU Darmstadt)
There is much user-generated discourse on the web (wikis, blogs, chats, etc) which is excellent material for language studies and various sorts of exploitations. However such data is full of typos, emoticons, shorthands etc. which need to be transformed into standard language and formatting to allow parsers to work on them to do for example sentiment analysis. A whole nested processing chain has been developed that includes a number of data cleansing services and if wanted a nested NLP processing chain to carry out additional work. The data cleansing operations typically carry out the removal of emoticons, the expansion of shorthands and the correction of spelling errors. The environment in which all can be executed is UIMA and a repository of
components is available. This includes knowledge about dependencies, i.e. chains can be built automatically. A task to be tackled is the remote execution of such processing chains or components of them. Another aspect is the availability of data, since some components need much data such as a Wikipedia dump.

Harnessing SOA Best Practices for Creating a Shared Service Backbone
C. Utschig (Oracle)
Web services are a major issue for Oracle to optimize various application scenarios driven by increasingly demanding customer needs and increasingly complex infrastructures. It is only the service-oriented architecture approach that will offer enough flexibility and interoperability to solve these challenges. A number of key issues for SOA were presented and discussed such as Web Service Addressing and Asynchrony, Web Service security, process orchestration and process monitoring. For WS addressing the WSDL (web services description language) and WSIF (web services invocation framework) standards and a few non-soap bindings (JCA, JDBC, RMI, HTTP) are relevant. For handling security issues SAML assertions and X500 based certificate propagation is relevant where signatures are checked, encryption can be applied and roles will be verified. For the orchestration of processes BPEL is the de-facto standard, since it is based on more than 10 years of experience and in particular handles asynchrony. For wiring web services together the Enterprise Service Bus concept has been adopted widely, since it facilitates performance improvements, transactionality, quality of service and interoperability and it helps in maintaining web services deployment and versioning. The ESB can be used as heterogeneous messaging backbone, content based routing, monitoring and reporting, security control, service workload checks etc. Finally a 4 layer architecture can be drawn where rich user interfaces (applications) interact with BPEL and workflow tools, these with the enterprise service bus from which finally backend services such as database connections are initiated. A new trend that is emerging is about federating ESBs, i.e. the collaboration of various ESBs in an environment of trust.

WebLicht - A Service Oriented Architecture
E. Hinrichs, Th. Zastrow (U Tübingen)
WebLicht is intended to produce incremental annotations based on web services processing chains which are interacting via Rest interfaces. HTTP Post method is used to interact with user interface tools. Processing and user interface components can thus run on distributed servers. Already a number of different NLP components from 4 different partners can interact and produce annotation results. A special text format has been created for WebLicht, but converters to ISO standardized formats are available.

WS registry and Chaining in DSpin
V. Boehlke (U Leipzig)
The Weblicht architecture allows to easily building NLP process chains by selecting web services that are described in a web service repository. The Registry Management tool contains descriptions that make use of elements that are registered in the ISOcat concept registry; thus it follows the CLARIN requirements in this respect. For automatically selecting suitable components it will try to find a perfect match between the description of a text format and those of all registered web services. The registry has been setup and tested and a first version of the match making algorithm has also been tested. Yet the CMDI metadata description standard is not used to describe text resources and web services and the connection to ISOcat has not been implemented. Both extensions are planned for the future.

The D-SPIN Text Corpus Format
U. Heid, K. Eckart (U Stuttgart)
Essential for the proper functioning of processing chains is the understanding of formats and encoding details of a data resource to the following processing step. Therefore the WebLicht partners agreed on a format (TCF) that is suitable to their needs, i.e. the processing components involved (tokenizers, lemmatizers, tagger, named entity recognizer, GermaNet, parser, frequency analyzer). This format has the advantage that tools only have to be modified minimally. It also covers some functional metadata. With respect to interoperability the following issues are of relevance: (1) format (syntax) of representations and their geometry (shape of trees) and (2) the meaning of the used tags. Interoperability thus includes reformatting and tagset mapping if they do not adhere to widely accepted standards as they are now under development in initiatives such as ISO TC37/SC4. Here LAF and GRAF are syntactic metamodels, FSD/FSR are methods to encode features, SemaAF/SynAF/MAF are models for encoding different linguistic layers and the tagsets used should be declared in the DCR. To achieve interoperability WebLicht therefore will provide converters that adapt the WebLicht internal processing steps with general standards that are in development. The consequence is that it will be difficult to integrate the WebLicht components easily with others at this moment.
Fragments of eAQUA
M. Büchler (U Leipzig)
The eAqua project is working on classical text fragments and wants to apply statistical NLP techniques to extract knowledge and construct relations such as for example re-usage of fragments. Ancient Greek philology, ancient history and computer science come together in this project. A larger number of processing components is made ready as web services in the FLEX framework including components from spell checking to the extraction of essential fragments. All processed texts are saved in TEI P5 in combination with epiDoc for epigraph extensions and object relations are stored using CIDOC-CRM. Yet there is not clear link to the CLARIN framework however a number of ingredients such as the usage of ISOcat and others are being analyzed.

A Case of Web Service Standards
S. Bel, V. Rodriguez (UPF Barcelona)
The Spanish CLARIN project worked on demonstrators to show the potential of web services. Two scenarios were presented: (1) a distributed metadata search and (2) a corpus analysis by making use of a workflow option. From an easy to use web application users can run statistics on selections of texts. In the CQP framework a metadata search is carried out to create a virtual collection selected from different sources, the results are being integrated and finally some analysis is carried out which includes graphical representations. It is concluded that we do not have interoperability in the web services domain yet, that it is absolutely necessary to use existing protocols, interfaces and standards wherever possible, that we need common interfaces for web services and that we need to find a way to store instance parameters and common interface definitions. In this demonstrator, in both scenarios, a large number of graphs and charts were presented to show final and intermediate results. The visual display of the data and a set of statistics describing these data were a key factor for the success of the Spanish demonstrator. Therefore, this could be taken into account for future CLARIN demonstrators. This project also indicates that in CLARIN we are not yet at a level where all activities in the web services area can easily come together.

7.2 Amsterdam Workshop
The Amsterdam workshop was organized to discuss Task 8 in WP2 (Guide the creation of some basic Services and Applications) and define demo cases that fulfill a number of requirements such as

- cross-country, cross-institutional and cross-lingual operation
- collaboration between several centers and teams
- offer new functionality and possible to be integrated by others
- integration of distributed resources
- combination of metadata and content search by making use of the agreed CMDI

The general architecture of the search case has been presented and discussed at earlier meetings already and will briefly be discussed here. It assumes that there are different repositories, preferably in different countries, that store language resources. These repositories offer the metadata descriptions of their resources preferably via OAI PMH to be harvested by a joint portal and they also make their resources Internet accessible via persistent links (in the future preferably persistent identifiers). It is also assumed that each repository provides a local content search engine that has an API that can be invoked from outside.

The indicated search engine actually can better be seen as a web application that includes two different applications. One is a complete metadata search engine operating on the harvested metadata descriptions. This machinery is ready in its first version called Virtual Language Observatory which in addition to the OAI PMH harvester includes a catalogue/search feature, a faceted browser and a Google Earth overlay all of which are integrated. For the demo cases a special more focused solution such as an API to the metadata database may be needed. Thus metadata searching will happen centralized in contrast to content search. There are two major reasons for this difference: (a) metadata is open while access to content data often is restricted; (b) metadata in general are a few bytes which can be easily transferred while copying content data will spoil the networks. The content search application therefore will provide a suitable user interface and will invoke the local content search engines by sending query information and receiving hit information. The final application should of course offer an integrated metadata and content search interface. For different types of use cases, however, more focused user interfaces will be required. Of course it should be possible for the user to immediately start suitable applications such as visualization of data when the hits are returned.
The plan is that it should be possible to input the resulting hits (if it related to retrievable text) to NLP web services to carry out some NLP and statistics operations. The usability will largely depend on the type of texts.

Three demo cases seem currently being prepared by multinational groups of CLARIN centers:

- the multimedia/multimodal demo case will bring together the resources of a variety of centers being active in this area with the intention to allow combined metadata and content searches and direct visualization for analysis
- the European Identity Index demo case will include a variety of newspapers from different countries to look for topics in common as to measure similarities. The result will express the similarity of the citizen matters independently of the European country. It will include distributed search by metadata and distributed processing using trans-national NLP tools and statistics based on web services.
- the C4 demo case where a large corpus will be used to feed an NLP processing chain which is being executed at different institutions.

### 7.3 Web Services Development

At the Amsterdam workshop a number of partners briefly sketched how far they were progressing with web services environments:

- the German WebLicht group (U Tübingen, U Leipzig, U Stuttgart, BBAW) is well on the way to create an easy to use NLP processing chain framework allowing its various chaining components to be executed at different locations;
- the DFKI will make its web services available to be invoked by others
- the UPF team presented its web services framework that was also very much focusing in the automatic generation of statistical data for better understanding of the experiments and improvement of the user experience when using NLP tools.
- the Sheffield team which is well-known for its GATE software has made huge steps to offer its processing components as web services that can be invoked by others
- the RACAI team presented its web services suite which is ready to be invoked by others
- the Charles University team described its will and capacity to offer their services as web services

---

68 This report is not meant to present the details; a more elaborated analysis of the linguistic functions will be carried out by WP5.
Summarizing we can say that the workshop showed that there are three potential multi-country groups that will collaborate on a joint solution that already now CLARIN can refer to a large number of web services - mostly with NLP functionality - that can be used.

7.4 Metadata Component for Web Services and Tools
In the meantime a web services metadata component has been worked out for web services and tools that can be used by everyone for test purposes. This component makes also use of those metadata elements which are describing functional aspects, which have been identified so far as being important for profile matching and which have been entered into the ISOcat metadata profile. Some teams tested its usefulness, but yet it has not been put in practical use since the complete CMDI framework will not come into functioning before summer 2010 as planned. However, all teams working on web services are asked to produce metadata descriptions soon following the CMDI principles.

7.5 SLCS Procedure for Local Applications
A first startup federation including 5 institutes from 3 different countries has been established based on formal agreements and made also contracts with the corresponding national identity federations. In doing so an initial trust domain was created which will be quickly expanded to other countries and centers. The participating centers are prepared to use Shibboleth or SimpleSAMLPHP components to give access to their resources by making use of distributed authentication, thus implementing a single identity principle. However this only works for browser initiated activities.

In collaboration with the Dutch grid experts this was extended to locally installed applications that are able to access resources with a X.509 certificate. This solution makes use of so-called Short Lived Certificates based on X509 certificates. This could now be used by CLARIN teams, provided we find a solution for either a EU wide SLCS service or a CLARIN specific one. However for web services we need to see if certificate based security is the right solution for us. WS security, and especially the delegation problem, can be based on either security tokens, SAML assertions and X.509 certificates. A follow-up project together with the Dutch grid experts is aimed at finding the best solution and test this in our environment.

Due to the lack of a full functioning solution for web services everyone is told in CLARIN to not touch the authentication issue and the transfer of user attributes between web services in the preparatory phase.

7.6 Evaluation
In this chapter we want to draw a first conclusion about what has been achieved so far in the area of web services.

7.6.1 Dynamics
The area of web services is still characterized by high dynamics and uncertainties. This has to do with almost all aspects ranging from user friendly interfaces and service bus implementations to create workflows up to security issues. Obviously CLARIN cannot re-invent the wheel or take a leading role in all these areas and develop solutions, since we would have problems to maintain the software. CLARIN is doing wise to continue its user oriented developments and to collaborate with other initiatives such as for example from the grid area.

7.6.2 Industry Developments and Standards
Industry developments are very important to be followed since often de facto standards are defined which one has to adhere to as well. Yet in the area of web services there are not so many standards. Obviously WSDL and SOAP are standards we need to consider for specifying APIs and for the information exchange. Since it requires more discipline and code when setting up web services people in general like to turn to using the REST guidelines which requires less rigidity. Yet we do not have a clear argument for choosing one of them, since we do not have the situation where we have hundreds of web services that need to collaborate with each other in an open domain.

69 The U Wroclaw team could not participate but it is known that they also have turned some of their interesting tools into web services so that they can be invoked by others.
It seems to be clear that BPEL is the de facto standard language in defining workflows since it has all required features such as for example supporting asynchronous service orchestration. We need to take more effort to analyze BPEL, but we certainly can say that we miss a simple graphical user interface tool that will hide the complexity of the BPEL language from the naive user. Again this is an area where CLARIN needs to be careful with proposing its own software developments since considering all aspects would lead to a considerable software package that can hardly be supported by CLARIN partners. Industry is working with high pressure on service bus implementations mainly to consolidate and concentrate common web service functionality. CLARIN will intensify its efforts to analyze these developments, to come up with own harmonized specifications and to see whether we can take over existing solutions and concepts.

7.6.3 Production Environment
Several teams in NLP are using the UIMA framework to configure processing chains and in doing so to create production frameworks for repeating tasks. Within UIMA the interoperability is solved by adhering to strong typing for the interchange format, i.e. any intermediate format needs to be specified exactly to allow subsequent processors to read the data. In this solution the interoperability therefore is not solved by metadata descriptions as specified in the CMDI metadata proposal but by creating a CAS specification. Both approaches have their advantages, again we cannot judge finally about pros and cons.

7.6.4 Interoperability
Currently we cannot speak about interoperability between all approaches since this was not the primary of WP2. Current reports[CLARIN-2009-DSR-3a] indicate that there is no convergence on standard interchange formats which are widely accepted. It was obvious that the pressure to show fast and convincing results influenced decisions on how to implement use cases for assessing feasibility and costs. In the two main and excellent demonstrator projects WebLicht (DE) and Language Indicators (SP), which were both mainly funded by national money, short-term solutions were found to be able to investigate on the solution of the most imminent problems.

In WebLicht a new format was defined that includes minimal metadata and that does not adhere to current standards. In Language Indicators also a minimal set of specialized metadata descriptions was proposed as well as common interfaces for tool types. As indicated both decisions were important to make fast progress and to be able to demonstrate the potential of the new technologies in a convincing manner. When starting the construction phase interoperability will get higher relevance, since we need mechanisms that allow users to combine any kind of web service that is offered from some partner. CMDI and standardized formats for data will be the only way to achieve this. Otherwise the investments to maintain format specifications and converters will be too high.

7.6.5 Joint Registry
As already indicated we miss a joint registry of all the created web services. The CMDI component is a basis and we need to convince every service producer to create CMDI compliant descriptions. It seems to be possible that in summer 2010 a first version of the CMDI infrastructure will be available which would allow us to build the required registry. It needs to be looked at how we can integrate the created descriptions in the Virtual Language World. The basic principles behind CMDI have been described in the report of the Data Management Task Force which has been accepted by the ESFRI and e-IRG boards, i.e. they are widely accepted principles.

7.6.6 User Interfaces for Workflows
Both major developments WebLicht and Language Indicators came up with proposals for nice and handy user interfaces for allowing naive users to build processing chains. Whatever will happen in the future we need to build on these experiences. Thus we probably will not have only one graphical user interface allowing users to specify workflows but we will have several dependent on the degree of complexity. Providing excellent guidance of users of various levels of technical expertise must remain a strong focus here.

7.6.7 Scalability and Workspaces
The scalability aspect was addressed several times at meetings. Currently the test cases are executed by one person during demo sessions and certainly this will scale up to a few users. But this is not sufficient. If CLARIN wants to establish a scenario where any arbitrary researcher can build a workflow by making use of some service being deployed on some departmental computer we need to tackle the scalability problem. This should not be solved by the CLARIN centers.

WG2.6 Requirements
Currently there are discussions at DG INFSO and DG Research level to build a layer for data services in addition to the existing infrastructures. CLARIN is invited to participate in these discussions from the beginning, since we have clear requirements that result from our advanced state of discussions about services and centers. CLARIN is aiming at a 2-tier structure of CLARIN centers on the one hand and data services centers on the other both with lean services restricted to where they are good in.

- The CLARIN centers will take care of all community related aspects such as for example creating and testing web services, giving access to data and allowing to enrich them, etc.
- The Data Services centers will take care of data storage and preservation and computational services which is easily possible since copies of the data will already be on these centers for preservation reasons. They will also provide workspaces for high and low end computations.

According to this scheme we will not need to copy data again and again to where the tools are installed, but we will need to work out a deployment method where services can easily be executed at the places where the data is residing. This solution needs to be worked out together with grid IT experts.
8 CLARIN Requirements

In this chapter we are summarizing the requirements for the emerging CLARIN infrastructure as they emerge in the area of web services and workflow.

1. The CLARIN infrastructure is an open infrastructure and must provide ease of use of integrating existing web services into the CLARIN landscape.
2. The CLARIN infrastructure must be able support multiple communication protocols. SOAP and REST are considered the main communication protocols for web services, although the infrastructure should be open to extensions to other protocols as well if the need arises.
3. The CLARIN infrastructure must be able to support multiple transport protocols. Although the current web services primarily communicate via HTTP, the need may arise to support other protocols as well. An example of this is FTP for file transfer.
4. Each web service must be registered at a CLARIN compliant web service registry. All service metadata is to be registered according to the CLARIN MetaData Infrastructure (CMDI) guidelines, that is metadata is to be provided in stand-off XML documents using the CLARIN MetaData model. This is obligatory in CLARIN for all activities.
5. Functional metadata of resources and services is to be described through the data categories specified in the metadata profile of ISOcat. Among others this will cater for profile matching.
6. The result of each web service invocation must be associated with CLARIN metadata. This metadata is to be provided in an automated manner through the metadata component. CLARIN must make a standard basic metadata component available which is capable of delivering at least the basic level of required CLARIN metadata to reduce the amount of work needed to integrate existing web services into the CLARIN infrastructure.
7. CLARIN must ensure that users have access to temporal workspaces where they can carry out their operations and store created temporary data in a temporary fashion.
8. Each web service invocation needs to be associated with provenance data. Provenance data must be stored through a provenance component. CLARIN must make a basic standard component available which is capable of storing provenance data in a uniform manner to reduce the amount of work needed to integrate existing web services into the CLARIN infrastructure.
9. An extension of the single sign-on principle to web services needs to be worked out.
10. The authorization information to web services is exclusively maintained by the originating institution - this right is not touched by the AAI.
11. Service providers must be allowed to operate under their own business model. The process of operating web services means that service providers will incur operational and maintenance costs and must be free to choose in the manner in which they want to handle these.
12. At first instance a wrapper functionality will be provided within CLARIN for test purposes and prototyping. But CLARIN will also need to study the CSB solution which will consist of action code, configuration files and library dependencies. This may be packaged as a single deployable unit. The packaging and deployment strategy needs to be worked out.
13. Handling of workflows must be addressed in subsequent workshops.
   a. How will (partial) workflow templates be handled and stored?
   b. Is there specific metadata for workflows?
   c. Which additional data categories are needed?
   d. Which workflow frameworks should be supported?
14. The search solution will be worked out as a CLARIN prototype.
15. The discussion about format and linguistic encoding harmonization and standardization needs to be continued, however, this is an activity of WP5.

---

76 Requirements with respect to metadata infrastructure are discussed in other documents.
9 Bibliography


[CITER] Citation of Electronic Resources, ISO Draft (2008)


---

WG2.6 Requirements


10 Appendices

Appendix A: Example user-defined UIMA Type System

An example of a user-(developer-)defined UIMA Type System. This Type System was defined to map elements from D-Spin TextCorpus format and covers several basic linguistic annotation types.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<typeSystemDescription xmlns="http://uima.apache.org/resourceSpecifier">
  <name>clarinTextCorpusDescriptor</name>
  <description/>
  <version>1.0</version>
  <vendor/>
  <types>
    <typeDescription>
      <name>textcorpus.Token</name>
      <description/>
      <supertypeName>uima.tcas.Annotation</supertypeName>
    </typeDescription>
    <typeDescription>
      <name>textcorpus.Sentence</name>
      <description/>
      <supertypeName>uima.tcas.Annotation</supertypeName>
    </typeDescription>
    <typeDescription>
      <name>textcorpus.Postag</name>
      <description/>
      <supertypeName>uima.tcas.Annotation</supertypeName>
      <features>
        <featureDescription>
          <name>token</name>
          <description/>
          <rangeTypeName>textcorpus.Token</rangeTypeName>
        </featureDescription>
        <featureDescription>
          <name>tag</name>
          <description/>
          <rangeTypeName>uima.cas.String</rangeTypeName>
        </featureDescription>
      </features>
    </typeDescription>
    <typeDescription>
      <name>textcorpus.Lemma</name>
      <description/>
      <supertypeName>uima.tcas.Annotation</supertypeName>
      <features>
        <featureDescription>
          <name>token</name>
          <description/>
          <rangeTypeName>textcorpus.Token</rangeTypeName>
        </featureDescription>
      </features>
    </typeDescription>
  </types>
</typeSystemDescription>
```
<typeDescription>
  <name>textcorpus.Parsenode</name>
  <description/>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>leaf</name>
      <description/>
      <rangeTypeName>uima.cas.Boolean</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>token</name>
      <description>should be set only if feature leaf == true</description>
      <rangeTypeName>textcorpus.Token</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>childNodes</name>
      <description>should be set if feature leaf == false</description>
      <rangeTypeName>uima.cas.FSArray</rangeTypeName>
      <elementType>textcorpus.Parsenode</elementType>
    </featureDescription>
    <featureDescription>
      <name>tag</name>
      <description/>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>root</name>
      <description>should be set to true for all sentences root nodes</description>
      <rangeTypeName>uima.cas.Boolean</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>

<typeDescription>
  <name>textcorpus.Dependency</name>
  <description/>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>dependent</name>
      <description/>
      <rangeTypeName>textcorpus.Token</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>governer</name>
      <description/>
      <rangeTypeName>textcorpus.Token</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>tag</name>
      <description/>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>

<typeDescription>
  <name>textcorpus.Coreference</name>
  <description/>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
  </features>
</typeDescription>
<description/>
<supertypeName>uima.tcas.Annotation</supertypeName>
<features>
  <featureDescription>
    <name>referents</name>
    <description/>
    <rangeTypeName>uima.cas.FSArray</rangeTypeName>
    <elementType>uima.tcas.Annotation</elementType>
  </featureDescription>
</features>
</typeDescription>
</typeSystemDescription>

<typeDescription>
  <name>textcorpus.TCMetadata</name>
  <description>Textcorpus metadata that describes some general properties of annotations the textcorpus contains, e.g. the tagsets used.</description>
  <supertypeName>uima.tcas.Annotation</supertypeName>
  <features>
    <featureDescription>
      <name>artificialText</name>
      <description>set to true if documentText was artificially composed from tokens instead of given.</description>
      <rangeTypeName>uima.cas.Boolean</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>posTagset</name>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>parsingTagset</name>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
    <featureDescription>
      <name>dependencyTagset</name>
      <rangeTypeName>uima.cas.String</rangeTypeName>
    </featureDescription>
  </features>
</typeDescription>
</types>
</typeSystemDescription>
Appendix B: Example UIMA analytical descriptor

Example of an analytic descriptor in UIMA. This analytic represents UIMA wrapper for Stuttgart Part-of-speech tagger. In descriptor it is specified that the analytics uses Type System shown in Appendix 3.3.2.1 and some other Type Systems. It is specified what types the analytics expects to be in input CAS and what types CAS will have at output. It declares the languages supported. It also declares behavioral metadata of the analytic (operationalProperties).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<analysisEngineDescription xmlns="http://uima.apache.org/resourceSpecifier">
    <frameworkImplementation>org.apache.uima.java</frameworkImplementation>
    <primitive>true</primitive>
    <annotatorImplementationName>de.tuebingen.uni.sfs.clarin.uima.wrappers.StuttgartTaggerWrapper</annotatorImplementationName>
    <analysisEngineMetaData>
        <name>stuttgartTaggerWrapperAEDescriptor</name>
        <version>1.0</version>
        <vendor/>
        <configurationParameters/>
        <configurationParameterSettings/>
        <typeSystemDescription>
            <imports>
                <import name="org.apache.uima.examples.SourceDocumentInformation"/>
                <import location="../types/clarinTypeSystemDescriptor.xml"/>
                <import location="../types/posPennTreebankTypeSystemDescriptor.xml"/>
                <import location="../types/posSTTStypeSystemDescriptor.xml"/>
            </imports>
            <typePriorities/>
            <fsIndexCollection/>
            <capabilities>
                <capability>
                    <inputs>
                        <type allAnnotatorFeatures="true" org.apache.uima.examples.SourceDocumentInformation/>
                        <type allAnnotatorFeatures="true" de.tuebingen.uni.sfs.clarin.uima.types.Sentence/>
                        <type de.tuebingen.uni.sfs.clarin.uima.types.Token>de.tuebingen.uni.sfs.clarin.uima.types.Token:lemma</type>
                    </inputs>
                    <outputs>
                        <type allAnnotatorFeatures="true" org.apache.uima.examples.SourceDocumentInformation/>
                        <type de.tuebingen.uni.sfs.clarin.uima.types.Token>
                            <type de.tuebingen.uni.sfs.clarin.uima.types.Pos/></type>
                            <type de.tuebingen.uni.sfs.clarin.uima.types.Sentence/>
                            <type de.tuebingen.uni.sfs.clarin.uima.types.Token:lemma/></type>
                        </outputs>
                        <languagesSupported>
                            <language>de</language>
                            <language>en</language>
                        </languagesSupported>
                    </capability>
                </capability>
            </capabilities>
        </typeSystemDescription>
    </analysisEngineMetaData>
</analysisEngineDescription>
```
<capabilities>
<operationalProperties>
  <modifiesCas>true</modifiesCas>
  <multipleDeploymentAllowed>true</multipleDeploymentAllowed>
  <outputsNewCASes>false</outputsNewCASes>
</operationalProperties>
</analysisEngineMetaData>
<resourceManagerConfiguration/>
</analysisEngineDescription>
Appendix C: Example UIMA analysis result

An example of analysis results from a UIMA analytic serialized into XMI. For the purpose of fitting one page some elements are replaced with «...». The corresponding Ecore model that defines types is in Appendix 3.3.2.4. In this example the analytic that produced this result was a UIMA wrapper of Stuttgart Part-of-speech tagger service.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:chars="http:///de/tuebingen/uni/sfs/clarin/uima/types/chars.ecore"
    xmlns:examples="http:///org/apache/uima/examples.ecore"
    xmlns:resource="http:///resource.ecore" xmlns:tcas="http:///uima/tcas.ecore"
    xmlns:cas="http:///uima/cas.ecore" xmlns:example="http:///example.ecore"
    xmlns:lexicon="http:///lexicon.ecore"
    xmlns:textcorpus="http:///textcorpus.ecore" xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0">
    <cas:NULL xmi:id="0"/>
    <cas:Sofa xmi:id="1" sofaNum="2" sofaID="TextCorpus0" mimeType="text"
        sofaString="Die groen Leute haben mir geraten, mit den Zeichnungen von offenen oder geschlossenen Riesenschlangen aufzuhren und mich mehr fr Geographie, Geschichte, Rechnen und Grammatik zu interessieren."/>
    <tcas:DocumentAnnotation xmi:id="8" sofa="1" begin="0" end="195" language="de"/>
    <textcorpus:Token xmi:id="21" sofa="1" begin="0" end="3"/>
    <textcorpus:Token xmi:id="25" sofa="1" begin="4" end="10"/>
    <textcorpus:Token xmi:id="29" sofa="1" begin="11" end="16"/>
    <textcorpus:Token xmi:id="33" sofa="1" begin="17" end="22"/>
    ...
    <textcorpus:Postag xmi:id="153" sofa="1" begin="11" end="16" token="29" tag="NN"/>
    <textcorpus:Postag xmi:id="159" sofa="1" begin="17" end="22" token="33" tag="VAFIN"/>
    ...
    <textcorpus:Lemma xmi:id="333" sofa="1" begin="11" end="16" token="29" lemmatext="Leute"/>
    ...
    <textcorpus:TCMetadata xmi:id="13" sofa="1" begin="0" end="0" artificialText="false"
        posTagset="STTS" parsingTagset="" dependencyTagset=""/>
    <cas:View members="8 21 25 29 33 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101 105 109 113 117 121
              125 129 133 137 141 147 153 159 165 171 177 183 189 195 201 207 213 219 225 231 237 243 249 255
              261 267 273 279 285 291 297 303 309 315 321 327 333 339 345 351 357 363 369 375 381 387 393 399
              405 411 417 423 429 435 441 447 453 459 465 471 477 483 489 495 13"/>
</xmi:XMI>
```
**Appendix D: Graphical view of UIMA Ecore model**

Graphical view of Ecore model that corresponds to UIMA Type System presented in Appendix 3.3.2.1. All the nodes under «textcorpus» node are the types defined in Type System file from Appendix 3.3.2.1. Under «cas» and «tcas» nodes the build-in UIMA types can be seen.

```
platform://resource/EMTest/model/ecoreFromUIMATS2.ecore

textcorpus
  - uima.apache.org
    - Token -> Annotation
    - Sentence -> Annotation
    - Postag -> Annotation
      - token : Token
        - tag : EString
    - Lemma -> Annotation
      - token : Token
        - lemmatext : EString
    - ParseNode -> Annotation
      - leaf : EBoolean
      - token : Token
      - childnodes : ParseNode
      - tag : EString
      - root : EBoolean
    - Dependency -> Annotation
    - Coreference -> Annotation
    - TCMetadata -> Annotation
      - uima.apache.org
      - artificialText : EBoolean
      - postagset : EString
      - parsingTagset : EString
      - dependencyTagset : EString

uima
  - cas
  - tcas
    - Annotation -> AnnotationBase
    - DocumentAnnotation -> Annotation
```
Appendix E: RACAI Web services in testing phase

The new web services under development at RACAI which have now entered the testing phase are:

1. Search Acquis
2. RC Acquis Question Classifier (JRCACQCWebService) for Romanian
3. JRC Query Generator for Romanian
4. Lexical Chainer
5. Paragraph Aligner

1. Search Acquis is a search engine web service. It returns an XML containing a list of paragraphs that responds to a given query entered as parameter. Other parameters are the number of paragraphs and the type of the question. It can also be used by entering a natural language question as a single parameter. In the question analysis phase, the searcher calls for the 2 and 3 web services in order to generate a query understandable by the search engine. Although the service was mainly built for searching through the Romanian version of JRC AC, it can be easily adjusted to other types of search engines and question classifiers.

2. The question classifier has a function (called predict) which needs as input a natural language question which is preprocessed by calling TextProcessing Web Service. The question analysis, which reveals the lemma and part-of-speech info of the words, is used to extract features representing the natural language question. The service would then return a Score object containing a string representing the class label (assigned by the classifier) according to these features and a sub-unitary number, representing the confidence score (for the assigned label), for the question. The classifier is based on a Maximum Entropy classification approach and it was trained on objects described by features representing questions formulated in Romanian. Therefore, the classifier correctly classifies only Romanian natural language questions.

At address http://nlp.racai.ro/WebServices/search.asmx?WSDL one could find the WSDL for the Search RespubliQA web.
Moreover, the set of class labels is adapted to fit the AC corpus and so, the service is supposed to classify questions asking about AC. In fact, it was implemented to work with search engine previously described.

For the question:

În câte categorii se diferențiază metodele statistice utilizate pentru elaborarea conturilor trimestriale?
(In how many categories can the statistical models, used for preparation of quarterly, be differentiated?)

The returned result is:

NUM 0.96331374922725521

Service

Click here for a complete list of operations.

predict

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter: Value
questionMFText: În câte categorii se diferențiază metodele statistice utilizate pentru elaborarea conturilor trimestriale?

Invoke

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

For now, one can find the WSDL at http://shadow.racai.ro/JRCACQCWebService/Service.asmx?WSDL.

3. The query generator has a function (called getQuery) that needs as input a preprocessed question (using the TTL Web Service) or a natural language question which is then preprocessed by calling TTL Web Service. It uses lemma, part-of-speech and chunk info of the content words to formulate a query which is to be fed to a search engine (the one described at 1). Beside mono-word terms, the query may contain multi-word expressions boosted by a factor equal to their word length and/or a span factor if the expression was formed by jumping over functional words. Like the classifier, the query generation service is designed to produce queries only for Romanian questions. In the next figure, the question introduced as parameter is the same as in the above example and the result returned is:

diferenția categorie utilize metode_statistice "elaborare cont trimestrial"^3 (elaborare AND cont AND trimestrial) "elaborare cont"^2 (elaborare AND cont) "cont trimestrial"^2 (cont AND trimestrial) elaborare cont
The lexical chainer web service is designed to return (see function getLexChains) a list of strings representing lexical/semantic paths in the WordNet hierarchy between two dictionary word forms (lemmas). It requires as input the two words, along with their general part-of-speech description (noun, adjective, verb or adverb). The result practically shows how much two words are semantically related. One can measure this closeness, for example, by assigning numbers to the semantic relations of the WordNet and then using them to measure the distances of the paths. The algorithm for such an application is language independent and can be implemented for any language which has a wordnet. However, our web service is built, for the moment, only for Romanian. In the next picture, one can see that we have introduced two nouns: roată (wheel) and volan (steering wheel). The service returns the WN path that links the two words:

roată (identical) ➔ roată|6n|ENG20-04398245-n (hypernym)
cârmă|1n|ENG20-03384266-n (hypernym) ➔ direcţie|8n|ENG20-04147387-n (holo_part) ➔ volan|2n|ENG20-04147670-n (identical) volan

The path runs through the words cârmă (helm) and direcţie (steering system) and 3 semantic relations. The analysis of these relations shows that the two words are very closely related.
5. Paragraph Aligner is a web service designed for aligning English-Romanian parallel texts at the paragraph level, but it can also be used for sentence alignment. The parallel texts can be provided both as plain preprocessed texts (using either TextProcessing or TTL Web Services) (with paragraphs delimited by ‘
’ chars) or XCES formatted (a parameter is used to signal the way the text is formatted). Other parameters are required: a Boolean parameter flagging if the plain text has the translation unit infos in it (tu info can be inserted in front of every paragraph delimited by ‘|’ char) (it’s not taken into account if the text is XCES formatted), an integer parameter representing the max radius of the search window around the main diagonal of the matrix formed by the two texts (if one doesn’t expect the texts to have the same discourse flow, he/she should set a larger window), and in case of XCES formatted texts, a Boolean parameter signaling whether the sentence Id or the translation unit Id should be used when returning the alignment result. The WSDL can be found at: http://shadow/PAlignerWebService/Service.asmx?WSDL.

For the following plain preprocessed texts:

**English:**

1. COUNCIL|council^Nc DECISION|decision^Nc of|of^Sp 21|21^Mc April|April^Nc 1970|1970^Mc Concerning|concern^Vm financial|financial^Af forecasts|forecast^Nc covering|cover^Vm several|several^Af years|year^Nc (|( 70|70^Mc /|/ 244|244^X /|/ ECSC|ECSC^Yn ,|, EEC|EEC^Yn ,|, Euratom|Euratom^Np )|)
2. THE|the^Dd COUNCIL|council^Nc OF|of^Sp THE|the^Dd EUROPEAN_COMMUNITIES|European_Community^Nc DECIDES|decide^Vm :|:
3. Article|article^Nc 1|1^Mc In|In^Sp order|order^Nc to|to^Qn place|place^Vm the|the^Dd budget|budget^Nc of|of^Sp the|the^Dd Communities|community^Nc within|within^Sp a|a^Ti framework|framework^Nc of|of^Sp forward|forward^Af planning|planning^Nc for|for^Sp several|several^Di years|year^Nc ,|, the|the^Dd Commission|commission^Nc shall|shall^Va ,|, each|each^Di year|year^Nc ,|, after|after^Sp receiving|receive^Vm the|the^Dd Opinion|opinion^Nc of|of^Sp the|the^Dd Budgetary|budgetary^Af Policy|policy^Nc Committee|committee^Nc ,|, draw|draw^Vm up|up^Sp a|a^Ti financial|financial^Af
Decizia Consiliului privind previziunile financiare multianuale

Decizia Consiliului privind previziunile financiare multianuale (CECO 1970/244) a fost adoptată în 1970, conform Deciziei Presedintelui Consiliului, care a stabilit date termenice pentru adoptarea, publicarea și execuția deciziilor și a avizurilor oficiale. Decizia include reviewing, întrucât în fiecare an, Consiliul va avea în vedere a) europenă, b) veche și c) a) avizul Adunării, b) avizul Comisiei, c) avizul Consiliului, d) avizul Comisiei, e) avizul Consiliului, f) avizul Comisiei, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 3/3 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 2/2 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 1/1 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

And Romanian:

1 Decizia Decizie a Consiliului a privit previziunile financiare multianuale. Acesta a fost adoptat în 1970, conform Deciziei Presedintelui Consiliului, care a stabilit date termenice pentru adoptarea, publicarea și execuția deciziilor și a avizurilor oficiale. Decizia include reviewing, întrucât în fiecare an, Consiliul va avea în vedere a) europenă, b) veche și c) a) avizul Adunării, b) avizul Comisiei, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 3/3 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 2/2 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

Articolul 1/1 Mc: Consiliul va examină în fiecare an, înainte de a stabilite previziunile financiare multianuale, a) avizul Consiliului, b) avizul Consiliului, c) avizul Consiliului, d) avizul Consiliului, e) avizul Consiliului, f) avizul Consiliului, g) avizul Consiliului, h) avizul Consiliului, i) avizul Consiliului, j) avizul Consiliului.

And xmlFormat parameter set to false, MaxLookRange set to 6 (representing the default 3 + text difference (3)) and showSId set to false, the resulted alignment is:
This means that paragraph 3 in en text is aligned with ro paragraph 6 with a score of 1.00077980405045. The sole importance of the score is to check if an alignment link is stronger/better that another, and nothing more.

As one can see, the ro paragraphs 1, 2 and 3 are left unaligned, but a simple heuristics can decide that they should go in the same alignment unit as paragraph 4:

Tu1: 

Tu2: 

Tu3: 

Tu4: 


The following picture is taken from the dev machine debugging session.

**Service**

Click [here](#) for a complete list of operations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputEN</td>
<td></td>
</tr>
<tr>
<td>inputCSO</td>
<td></td>
</tr>
<tr>
<td>xmlFormat</td>
<td></td>
</tr>
<tr>
<td>MsToNames</td>
<td></td>
</tr>
<tr>
<td>MaxLookAroundRange</td>
<td></td>
</tr>
<tr>
<td>showE60</td>
<td></td>
</tr>
</tbody>
</table>

**Invoke**

**SOAP 1.1**

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```xml
POST /AlignmentService/Service.svc HTTP/1.1
Host: racai.ro
Content-Type: text/xml; charset=UTF-8

SOAPAction: "http://tempuri.org/align"

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    xmlns:schema="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <align xmlns="http://tempuri.org/">
            <inputEN value="http://tempuri.org/">
            <inputCSO value="inputCSO"/>
            <inputXML value="inputXML"/>
            <xmlFormat value="xmlFormat"/>
            <MsToNames value="MsToNames"/>
            <MaxLookAroundRange value="MaxLookAroundRange"/>
            <showE60 value="showE60"/>
        </align>
    </soap:Body>
</soap:Envelope>
```

After the test phase all of the above services will be deployed at [www.racai.ro/WebServices](http://www.racai.ro/WebServices)