Language Technologies for Lifelong Learning

LTfLL -2008-212578

Project Deliverable Report

Deliverable D2.1 – Services Approach & Overview General Tools and Resources

Work Package	WP Infrastructure
Task	2.1, 2.2, 2.4, 2.5
Date of delivery	**Contractual**: 31-10-2008 **Actual**: 15-12-2008
Code name	D2.1 **Version**: 1.0 Draft ☐ Final ☒
Type of deliverable	Report
Security (distribution level)	Public
Contributors	Reinhard Dietl (WUW), Fridolin Wild (WUW), Bernhard Hoisl (WUW), Christian Buchta (WUW), David Meyer (WUW), Kurt Hornik (WUW), Stefan Sobernig (WUW), Felix Mödritscher (WUW), Berit Richter (BIT MEDIA), Markus Essl (BIT MEDIA), Gerhard Doppler (BIT MEDIA)
Authors (Partner)	Reinhard Dietl (WUW), Bernhard Hoisl (WUW), Fridolin Wild (WUW), Berit Richter (BIT MEDIA), Markus Essl (BIT MEDIA), Gerhard Doppler (BIT MEDIA)
Contact Person	Fridolin Wild (WUW)
WP/Task responsible	WUW
EC Project Officer	Mr. M. Májek
Abstract (for dissemination)	The contents of this deliverable are split into three groups. Following an introduction, a concept and vision is sketched on how to establish the necessary natural language processing (NLP) services including the integration of existing resources. Therefore, an overview on the state-of-the-art is given, incorporating technologies developed by the consortium partners and beyond, followed by the service approach and a practical example. Second, a concept and vision on how to create interoperability for the envisioned learning tools to allow for a quick and painless integration into existing learning environment(s) is elaborated. Third, generic paradigms and guidelines for service integration are provided.
Keywords List	Service Concept, NLP, Educational Mash-Ups

LTfLL Project Coordination at: Open University of the Netherlands
Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands
Tel: +31 45 5762624 – Fax: +31 45 5762800

Table of Contents

List of Figures

Executive Summary

Together with WP3 on scenarios, this horizontal work package aims at guiding and supporting research and development within the project. Work within the work package is organised along two groups of tasks: the first being targeted towards the natural language processing (NLP) services, whereas the second serves the set-up, maintenance, and further development of the learning tool (LT) infrastructure (which deploys the natural language processing services).

In a way, one could say that the overall goal of this infrastructure work package is to create interoperability – and the break-down into two lines of tasks (plus the final roadmap) is merely the realisation of interoperability with the two facettes natural language processing services and learning tools.

Task T2.1 covers the development of a concept and the basic set-up of the infrastructure including a definition of guidelines for the services and technical validation criteria.

Task T2.4 relates to designing, developing, and/or enabling access to general utilities and services to support the language technologies.

Interoperability can be defined a property that emerges, when distinctive information systems (subsystems) cooperatively exchange data in such a way that they facilitate the successful accomplishment of an overarching task (Wild & Sobernig, 2006).

Naturally, as this deliverable has to outline the service approach and to give a general overview on existing tools and resources, it has to follow two research lines on how interoperability can be realised within the project.

First, a concept and vision is sketched on how to establish the necessary NLP services including the integration of existing resources. Therefore, an overview on the state-of-the-art is given, incorporating technologies developed by the consortium partners and beyond, followed by the service approach and a practical example.

Second, a concept and vision on how to create interoperability for the envisioned learning tools to allow for a quick and painless integration into existing learning environment(s) is elaborated.

The concepts proposed are generic enough to integrate the background brought in by partners into the work packages 4 to 6 – and beyond. Still, it will ensure their interoperability and will provide a solid infrastructure to build the next showcases as well as, subsequently, the services on. How this can be achieved is especially illustrated with the practical example in 2.3 which is a simplification (distorted but understandable!) closely related to a service envisioned for T5.2 of WP5 on assessing student writings.

To round up this deliverable, generic guidelines for the service integration are provided within the last section.

1. Interoperability

A change in perspective can be certified in the recent years to technology-enhanced learning research and development: More and more learning applications on the web are putting the learner centre stage, not the organisation. They empower learners with capabilities to customise and even construct their own personal learning environments (PLEs). These PLEs typically consist of distributed web-applications and services that support system-spanning collaborative and individual learning activities in formal as well as informal settings.

Technologically speaking, this shift manifests in a learning web – where information is distributed across sites and activities can easily encompass the use of a greater number of pages and services offered through web-based learning applications. Mash-ups, the 'frankensteining' of software artefacts and data (Hartmann et. al., 2008), have emerged to be the software development approach for these long-tail and perpetual-beta niche markets. Core technologies facilitating this paradigm shift are Ajax, JavaScript-based widget-collections, and micro formats that help to glue together public web APIs in individual applications.

Interoperability is the necessary precondition for this service concept. Interoperability is a property that emerges, when distinctive information systems (subsystems) cooperatively exchange data in such a way that they facilitate the successful accomplishment of an overarching task (Wild & Sobernig, 2006).

Interoperability, for the scope of this project, has to be achieved on three layers (see Figure 6). First, data collection and management has to be enabled by establishing a set of light-weight data formats and data access methods. Second, services have to be established (e.g., LSA-based natural language processing services) that allow for remote access of all crucial processing services. Third, the application front-ends have to be turned into widgets and portlets in order to allow 'google-maps-style', convenient re-use of logically coherent components. By achieving interoperability on these three layers, an architecture can be realised that maximises re-usability while at the same time allowing for heterogeneity in the implementation processes.

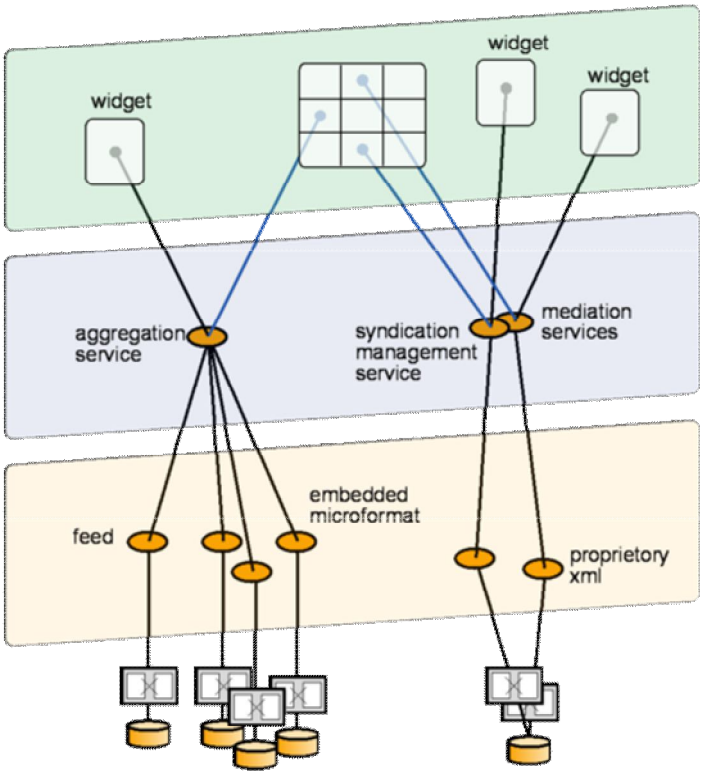

Figure 1. Architectural layers.

Learners produce **data** in various ways during their learning activities. Digital traces left, for example, can be a document, postings in a forum, and utterances in a chat: any tool involved in the learning process can hold valuable input data to the analysis services envisioned in LTfLL. This scattered data have to be transported to the point where they are needed in order to allow the analytical data processing serves to perform their calculatory duty. To allow for data gathering from multiple sources, feeds provide an established and simple solution (Wild et. al., 2008b). Enriched with microformats, they provide an expressiveness that may suffice for most of the applications faced within the project.

How data is processed in the aggregating system depends on the **services** of the different work packages 4, 5, and 6 enriched by generic services provided from within the infrastructure work package. For most services, REST style services will provide – through their simplicity – advantages over more complex alternatives like SOAP or XML-RPC.

Most of these tools have some output that has to be visualised to the user. Some tools work as standalone applications, others are embedded into a more complex web application, often a kind of learning management system (LMS) or content management system (CMS). To encapsulate the logical user interface units, i.e. dialogue-sized visual appearances with a particular, use-case sized behaviour, and to enable them for direct re-use across several systems, the use of **widgets** is recommended.

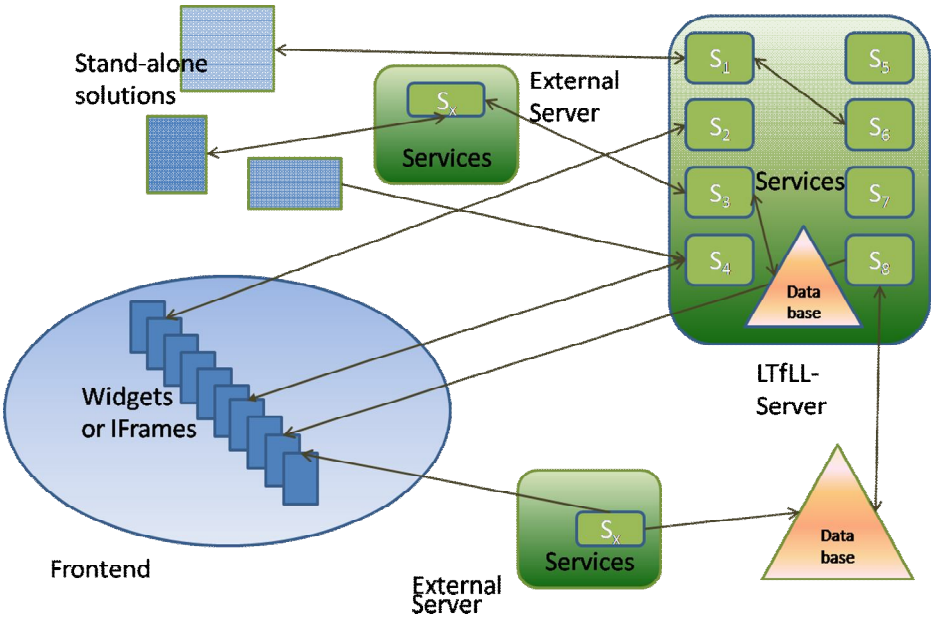

Figure 2. Architecture Overview.

Put into practice, a potentially complex system results. Widgets, services, and data storage facilities are distributed across a heterogeneous set of web applications and servers:

- Services can be hosted on one LTfLL server or on multiple servers.
- Services can also be hosted on any other application server.
- Output can be displayed in widgets embedded in a frontend.
- Output can be displayed in any stand-alone solution.
- Data can be stored in a central database on one or many LTfLL server.
- Simple data exchange between components can be realised with feeds.
- Complex data exchange is encapsulated in services.

In the following two chapters, the layered concepts for the natural language processing services and for the learning tools will be developed against the background of the state of the art. Both will be illustrated with an example.

This deliverable is rounded up by a chapter on guidelines.

2. Interoperability for Natural Language Processing

Within each development cycle, the necessary tools and resources will be elaborated and documented in subsequent deliverables. Here, the aim is on outlining a service oriented framework that allows for the integration of heterogeneous natural language processing services.

Added in the Annex 1, there is a listing of the language corpora currently available within the consortium which can perform as data inputs for the software components still to be developed. Additional information on existing services, tools, and resources in the context of the first show cases can be found in D3.1.

This deliverable draws from D3.1 and D7.1 regarding the elaboration of the scenario-driven design process. Reading them first is highly advised.

2.1 State of the Art

This section describes existing technologies – that means state-of-the-art frameworks and their tools plus resources – available for text mining and natural language processing. Thereby, the focus is set on available interfaces, used data formats, and support for orchestration facilities.

2.1.1 General Architecture for Text Engineering (GATE)

GATE was developed in 1995 by the University of Sheffield and its first release was in 1996. GATE is an infrastructure for developing and deploying software components that process human language. It specifies a software architecture for natural language processing, a framework which implements the architecture and which can be used to embed language processing capabilities in diverse applications, and a development environment built on top of the framework made up of convenient graphical tools for developing components.

GATE is open-source and distributed under the GNU Lesser General Public Licence (LGPL)[1]. GATE is written in Java, which is also its preferred programming language, although using JNI nearly every language can be used to deploy GATE applications. GATE supports both Oracle and PostgreSQL databases.

GATE uses specialised types of Java Beans, and comes in three flavours (Cunningham, Maynard, & Bontcheva, 2008):

- **Language Resources (LRs):** lexicons, corpora, ontologies
- **Processing Resources (PRs):** algorithmic (parsers, generators, modellers)
- **Visual Resources (VRs):** visualisation and editing components (components that 'participate' in graphical user interfaces)

GATE is able to read Unicode format (e.g. UTF-8), thus it has no problems with different character-sets of languages. There exists GUK, the GATE Unicode Kit for developing applications using Unicode support. Supported document formats are: XML, RTF,

[1] That means that GATE can be embedded in commercial products if required.

HTML, SGML, and plain text (there is also a basic support for PDFs and Microsoft Word documents[2]). In all cases the format is analysed and converted into a single unified model of annotation (Cunningham, Maynard, & Bontcheva, 2002, p. 3). The annotation format is a modified form of the TIPSTER format, which has been made largely compatible with the Atlas format (Bontcheva, Kiryakov, & Cunningham, 2003, p. 2). XML is used for annotations. JAPE (Java Annotation Patterns Engine) provides regular-expression based pattern/action rules over annotations.

GATE has a single model for information that describes documents, collections of documents (corpora), and annotations on documents based on attribute/value pairs. Attribute names are strings; values can be any Java object. The API for accessing this feature data is Java's Map interface (part of the Collections API).

A corpus in GATE is represented in Java as a Set interface whose members are documents. Both corpora and documents are types of Language Resources (LR); all LRs have a FeatureMap (an extended Java Map) associated with them that stores meta-data ('attribute + value' style) about the resource (Cunningham, Maynard, & Bontcheva, 2008).

For information extraction (IE) tasks there exists ANNIE (A Nearly-New Information Extraction system) which is a processing resource (PR) for language analysis. Some of the features coming with ANNIE are: a tokeniser, a gazetteer, a sentence splitter, a part-of-speech tagger, a semantic tagger, an orthographic co-reference tagger, and a pronominal co-reference tagger (Wikipedia, 2008).

It is possible to combine GATE and UIMA (see next section) through UIMA's SDK (support for Java and C++). The two main parts needed for this integration are:

- A wrapper to allow a UIMA Text Analysis Engine (TAE), whether primitive or aggregate, to be used within GATE as a processing resource (PR).
- A wrapper to allow a GATE processing pipeline (specifically a CorpusController) to be used within UIMA as a TAE.

The GATE structure is based on components: reusable chunks of software with well-defined interfaces. The set of resources integrated with GATE is known as CREOLE (Collection of REusable Objects for Language Engineering). All resources are packaged as Java archives (JAR) and deploy XML configuration files (Cunningham, Maynard, & Bontcheva, 2008). Therefore, a set of developed resources can be embedded in the target client application through the GATE framework.

As GATE is entirely written in Java it can draw from many functionally rich libraries available for this programming language. For example, providing GATE functionalities as web-services can conveniently be realised with the help of 'Metro' (java.net Community, 2008) which provides APIs and tools for generating web-services. Metro itself consists of JAX-WS, a reference implementation for deploying web-services.

[2] Only the text is extracted from the documents, no formatting information is preserved.

Components of GATE are written as Java Beans, so they obey to certain conventions regarding method naming, construction, and behaviour.

As for the idea of orchestration facilities, numerous orchestration servers exist for the Java programming language, where the BPEL and WS-BPEL standards are implemented. An extended BPEL definition called BPELJ also provides the possibility to include Java code (called Java Snippets) in BPEL process definitions.

2.1.2 Unstructured Information Management Architecture (UIMA)

UIMA was first developed at IBM and is now an Apache Software Foundation project. It is open source and distributed under the Apache License. The Apache UIMA framework includes an all-Java SDK implementation of the UIMA framework. It also includes a C++ version of the framework (not contained in the core distribution) and enablement's for annotators built in Perl, Python, and TCL (IBM, 2006).

In analysing unstructured content, UIM applications make use of a variety of analysis technologies including those from statistical and rule-based natural language processing, information retrieval, machine learning, ontologies, automated reasoning, and a diverse set of semantic resources (e.g., CYC, WordNet, FrameNet)[3] (IBM, 2006, p. 24). Unstructured content is not limited to purely plain text, but can be an audio or video stream, an HTML page or similar – all of them called artefact. Each representation of an artefact is called a 'Subject of Analysis' (Sofa), for which the corresponding data types can be Java Unicode strings, feature structure arrays of primitive types (special objects of byte or float arrays), or a URI.

UIMA defines a Common Analysis Structure (CAS) for annotators to represent and share their analysis results. It is an object-based data structure that allows representation of objects, properties, and values. UIMA provides an implementation of the CAS with multiple programming languages. Through these interfaces, the annotator developer interacts with the document, and reads and writes analysis results. For Java annotator developers, UIMA provides the JCas, a Java based interface to CAS objects. Each type declared in the system appears as a Java class.

For every component specified in UIMA, there are two parts:

- The declarative part (metadata describing the document, structure and behaviour; XML)
- The code part (algorithm implementation; mainly Java)

The UIMA framework can handle tightly-coupled (running in the same process) or loosely-coupled (running in separate processes or even on different machines) analysis engines (software objects which do the computation parts). The framework supports a number of remote protocols for loosely coupled deployments, including SOAP – the standard web service protocol (IBM, 2006, p. 33). The other important communication protocol is called Vinci, which is a lightweight version of SOAP, included as a part of Apache UIMA. An existing component is the Vinci directory, known as VNS (Vinci

[3] A semantic search engine from IBM's alphaWorks is also provided (IBM, 2008).

Naming Service), which provides information about the different available services (name of the host machine, name of the service).

The UIMA framework can make use of these services in two different ways:

- An **Analysis Engine** can create a proxy to a remote service; this proxy acts like a local component, but connects to the remote. The proxy has limited error handling and retry capabilities. Both Vinci and SOAP are supported.
- A **Collection Processing Engine (CPE)** can specify non-Integrated mode. The CPE provides more extensive error recovery capabilities. This mode only supports the Vinci communication protocol (IBM, 2006, p. 147).

To deploy a UIMA component as a SOAP service one has to install the following software components: Apache Tomcat (Java Servlet and JSP implementation) and Apache Axis (SOAP implementation).

With the UIMA framework it is possible to increase performance using parallelism. This can be done with additional threads within one Java virtual machine on one host or by deploying different analysis engines on a set of remote machines (IBM, 2006, p. 156). Monitoring performances can be done via the Java Management Extensions (JMX), e.g. CASs per second.

Another component of the UIMA framework is the so-called Flow Controller, which plugs into an Aggregate Analysis Engine and determines the order in which the components of that aggregate are invoked.

For distribution and re-use of developed components within the UIMA framework, it is possible to generate a PEAR (Processing Engine ARchive) file, which is a standard package for UIMA components.

As the UIMA framework has a Java programming language interface, it can benefit from the already existing software components, toolkits and frameworks described in the chapter before.

2.1.3 The Language and Environment R

R is an integrated suite of software facilities for data manipulation, calculation and graphical display (Venables & Smith, 2008, p. 2). It consists of a programming language plus a run-time environment with graphics, a debugger, access to system functions, and the ability to run programs stored in script files. R claims to be a fully coherent system, not a collection of different tools merged together. R can be regarded as an implementation of the S language which was developed originally at the Bell Laboratories by John Chambers and colleagues. The main focus of R can be seen as to serve as a statistic computing system (but it is not only limited to statistical computing). Many modern statistical techniques have been implemented. There are about 25 core packages supplied with R and – as R is an open-source GNU project – many more are available as extensions (more than 1.300).

R has an object oriented programming language implementation, providing a number of specialised data structures to access data stored in memory. For data storage, the most

important are vectors, lists, factors, matrices, and data frames. Vectors can be thought of as contiguous cells containing data. R has six basic vector types: logical, integer, real, complex, string (or character), and raw. Contrarily, lists have elements, each of which can contain any type of R object, i.e. the elements of a list do not have to be of the same type. Factors are used to describe items that can have a finite number of values (gender, titles, etc.). Data frames are generally speaking matrices of data. A data frame is a list of vectors, factors and/or matrices all having the same length (R Development Core Team, 2008c, p. 3ff).

As for data import and export there exists several interfaces. The easiest is importing data from a text file. Data can also be stored using XML (provided that the necessary package has been installed). In addition, there exists a package which has the ability to import data from other statistical software, like SAS, SPSS, S-PLUS etc. But there are limitations on the types of data that R handles well. R is not well suited to extremely large data sets, since all data being manipulated by R reside in memory. Though there are extension packages for R that help with the manipulation of high volume data. Furthermore, R does not easily support concurrent access to data. For the purpose of managing data, there exists much better solutions like (R)DBMSs. Available packages support on the one side MySQL, SQLite, and Oracle directly and on the other side there is a package for interacting with DBMSs through the ODBC standard interface. Tested DBMSs are Microsoft SQL Server, Access, MySQL, PostgreSQL, Oracle and SQLite (R Development Core Team, 2008a, p. 2ff).

R supports a variety of connection types, including file connections, text connections[4], pipes[5], URLs, and sockets. By looking at the ability of R for network interactions, there are interfaces for (D)COM, CORBA, SOAP, and others (R Development Core Team, 2008a, p. 20ff).

In R there exist interface functions for compiled C and FORTRAN code. Therefore, it is possible to pass R objects to compiled code. Additionally, the C function can be used with other languages which can generate C interfaces, for example C++. It is also possible to use R from C code (and in a limited way from FORTRAN code, as well). As R can be built as a shared library, it can be used to run R from alternative front-end programs. This means, a GUI or any other application that has the ability to submit commands to R and perhaps receive results back.

Concerning web interfaces and orchestration abilities of R, there exist packages, for example, for running R scripts through the CGI interface, allowing submissions of data using both GET and POST methods. Moreover, there is an R/Apache integration, which consists of an Apache module that embeds the R interpreter inside the web server. Another package has implemented a TCP/IP server which allows other programs to use facilities of R from various languages. Additional packages for the R environment enables R to be used with other programming/scripting languages than C/C++ and FORTRAN, for example Java, Tcl/Tk (in core distribution), Perl, Python, XLisp, or PHP, for which some of them are highly suited to handle shared network interactions.

[4] Allow R character vectors to be read as if the lines were being read from a text file.
[5] A special form of file that connects to another process.

Furthermore, support for web services are enabled through a package using Java/Axis/Apache as underlying technologies. Hence, there exist implementations for using XML based documents and the SOAP protocol for exchanging these documents/messages over a network (Hornik, 2008).

For the distribution of self-developed software, it is very easy to create an extension to R bundled in one single package, which is able to run on all operating system platforms where the R program can be installed on[6]. If a binary package is deployed and made available on the Internet, any user can download and install it either through the R GUI or with an install-package-command from the R console. In either case the installation is straight-forward and does not need any in-depth knowledge of the underlying software system.

An R package consists of a special subfolder structure and may contain files for information and configuration of the package as well as license and copyright descriptions. Additionally, there may be a news, change log or readme file (R Development Core Team, 2008d, p. 2). Furthermore, help files are describing the functionalities of the package in detail, while demonstration programs can present them. For internationalisation purposes one can set the encoding type of the description files so that non-ASCII characters of different languages can be displayed (R Development Core Team, 2008d, p. 42).

There is an established development process for engineering new packages that involves versioning, packaging, consistence checking, testing, profiling, documenting, and the like. Through the comprehensive R archive network (CRAN), R distributions and packages are made available in a worldwide network of mirrors. One of the success factors of R and CRAN may lie in the tradition to accompany a package release with the publication of so-called 'vignettes', i.e. papers which document the aim of the software released and explain its usage with code samples. By using LaTeX, these vignettes can even contain executable R code which again is used in the testing routines executed for a package release to check for errors.

Among its vast amount of extension packages, there also are several ones dedicated to natural language processing and text mining. To develop this area and help developers align their agenda, a so-called R task view has been introduced for natural language processing (Feinerer & Wild, 2008). Here, a quick overview of the most important extension packages is given:

- Natural Language Processing
    - *openNLP*: An interface to openNLP, a collection of natural language processing tools including a sentence detector, tokeniser, pos-tagger, shallow and full syntactic parser, and named-entity detector, using the Maxent Java package for training and using maximum entropy models.

[6] Packages may be distributed in source form or compiled binary form. Installing source packages requires that compilers and tools be installed. Binary packages are platform-specific and generally need no special tools to install (R Development Core Team, 2008b, S. 15).

- *openNLPmodels*: English and Spanish trained models for the package *openNLP*.
- *RWeka*: is an interface to Weka which is a collection of machine learning algorithms for data mining tasks written in Java, containing tools for data pre-processing, classification, regression, clustering, association rules, and visualisation.
- *Snowball*: provides the Snowball stemmers which contain the Porter stemmer and several other stemmers for different languages.

- Text Mining
  - *tm*: a comprehensive text mining framework including count-based analysis methods, text clustering, text classification and string kernels.
  - *lsa*: provides routines for performing a latent semantic analysis.
  - *corpora*: offers utility functions for the statistical analysis of corpus frequency data.
  - *languageR*: provides data sets and functions exemplifying statistical methods.
  - *zipfR*: offers some statistical models for word frequency distributions. The utilities include functions for loading, manipulating and visualising word frequency data and vocabulary growth curves. The package also implements several statistical models for the distribution of word frequencies in a population.
- Keyword Extraction and General String Manipulation
  - *RKEA*: provides an R interface to KEA (Keyphrase Extraction Algorithm), which allows for extracting key phrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary.
  - *gsubfn*: can be used for certain parsing tasks such as extracting words from strings by content rather than by delimiters.
- String Kernels
  - *kernlab*: allows to create and compute with string kernels, like full string, spectrum, or bounded range string kernels.
- Lexical Database
  - *wordnet*: provides an R interface to WordNet, an on-line lexical reference system[7].

2.1.4 Architecture and Tools for Linguistic Analysis Systems (ATLAS)

ATLAS addresses an array of applications needs encompassing corpus construction, an evaluation infrastructure, and multi-modal visualisation (Cover, 2000). The ATLAS

[7] In WordNet English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets.

framework provides an architecture targeted at facilitating the development of linguistic applications. It is made of four main components: an annotation ontology, an API, an interchange format for linguistic data, and MAIA, a type definition infrastructure (Laprun, Fiscus, & Garofolo, 2002, p. 264).

The annotation ontology at ATLAS' core provides the abstractions on which the rest of the framework is built. These abstractions can be implemented using diverse programming languages. A Java instantiation of the data model is available (jATLAS) and provides an API to the core objects allowing their easy manipulation (jATLAS Development Team, 2003). jATLAS supports some basic low-level services, like data import/export, management utilities, definition of a Service Provider Interface (SPI) and automatic validation services via MAIA. jATLAS is open source and freely available, but only in a beta version, yet.

Moreover, linguistic data expressed using ATLAS abstractions can be serialised to XML using the ATLAS Interchange Format (AIF) to facilitate their exchange and reuse. For physical storage the AIF or a RDBMS system (accessible from ODBC-compliant calls) can be used.

The ATLAS approach separates logical and physical levels from application-specific levels (Bird, Day, & Garofolo, 2000, p. 1700f):

- The **logical layer** consists of a linguistic formalism and an API. The formalism is the annotation graph model and it's generalisation to higher-dimensional cases. The API defines a set of procedures for creating, modifying, searching and storing well-formed annotation sets.
- The **physical layer** where API specification will allow various physical storage implementations that applications are free to access in multiple ways. As mentioned earlier, the two dominant storage strategies are AIF and RDBMS.
- The **application layer**, which is left up to the developer. Any application that can read, manipulate, or annotate ATLAS data would fall under this category.

The Atlas project is no longer developed (since 2003).

1.1.5. Summary

Regarding their potential, the frameworks GATE, UIMA, and R express a comparably rich architecture that allows for the integration of heterogeneous natural language processing services. As, however, important core components are already available for R, most notably the text mining framework *tm* and the package for latent semantic analysis *lsa* both developed by consortium partners (and not to forget the big set of SNA packages), there is a strong favour for R. Through R's capabilities to interface with other environments, this seems to be a good starting point for the interoperability work within LTfLL. Future integration work will show whether this initial favour has to be revised.

2.2 Service Approach

Although a unified definition of a service oriented architecture has not yet been agreed on, this section will try to describe the service-oriented architecture (SOA) of a natural language processing system using a reference model based on software patterns as suggested in (Avgeriou & Zdun, 2005)Reference_Zdun2005 by looking at the architecture of the services and the logic behind them from different points of view.

2.2.1 Layer View

Figure 3 shows how the components of a typical NLP system can be split into different layers. The suggested 4-tier-architecture comprises the layers typical for a client-server-architecture, augmented with an indirection layer, represented by the services.

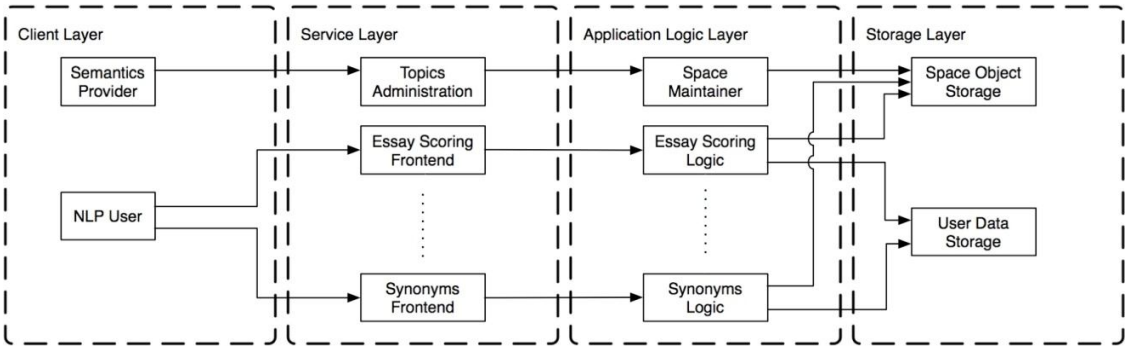

Figure 3. Layer decomposition of a possible NLP system.

Starting with the highest-level components on the left side, the client layer contains all software used to interface with NLP services. In this context, a 'semantics provider' is any client serving text data or other semantic information to the system that is used to build reusable objects like spaces; an 'NLP user' is a client used to provide semantic data to the system that is used to perform semantic calculations utilizing the reusable objects on the server, using the NLP service.

The service layer exposes the key functionality of the system to the clients. It serves as an indirection layer, as it can expose the functionality from the application logic layer in a condensed form if necessary. Note that the "topics administration" functionality is used to create, modify and delete reusable objects that are stored on the server for later use ("topic" refers to the context of LSA, where a semantic space represents a specific topic) It is also responsible for handling any connection-related issues in the client-server communication process. In Figure 3, the dotted line between the two frontends implies that multiple NLP tasks can be wrapped this way, the essay scoring and the synonym search being only examples.

The application logic layer holds any infrastructure responsible for the actual calculations. The space maintainer is a routine capable of creating, modifying ('fold-in') and dropping actual latent-semantic spaces, and therefore encapsulates the core NLP logic. Furthermore, this layer comprises any task-specific logic used to serve NLP user

requests. A more extensive description of latent semantic analysis and the corresponding application logic is contained in the deliverables D4.1 and D5.1.

The storage layer represents a supporting sub-system, serving the application logic layer. The space object storage is able to hold generated spaces in a highly accessible way. If transfer of spaces is chosen to be avoided in favour of a reference-driven communication, the storage must be able to serve a space identification token ('space ID') for every space provided, and vice versa. The user data storage holds parameter data provided by the user, possibly on a per-session or a per-account basis.

2.2.2 Data Flow View

Figure 4 shows how data is moved within an NLP system during the two key processes: space generation and NLP task execution. It also shows the key input and output data types at each stage, which is important for realisation of a pipes-and-filters-architecture.

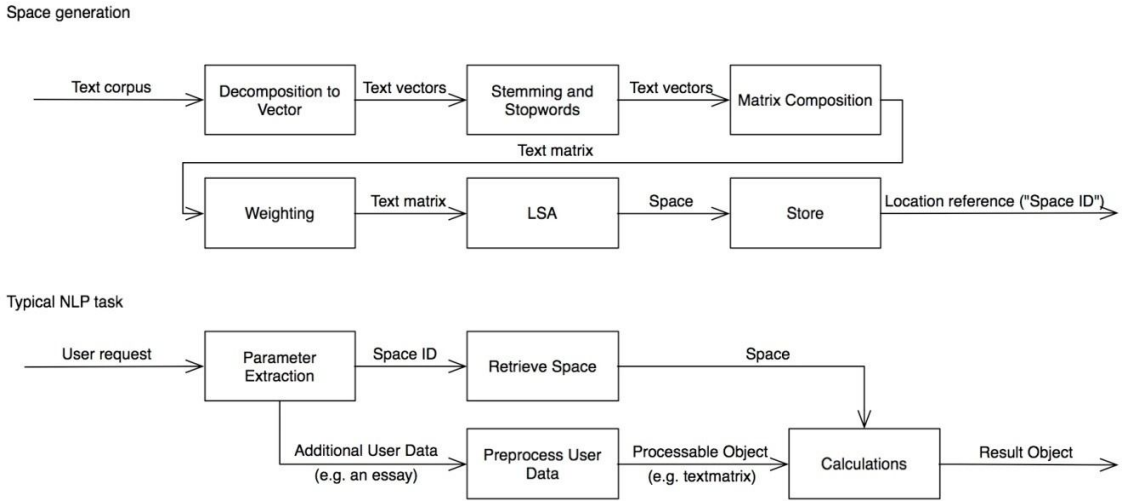

Figure 4. Data flow during a typical NLP process.

For the space generation, a text corpus is put into the process, where it is transformed into a computable object and then transformed into a space (Figure 4 shows this process for an LSA-based computation). Note that this model suggests, that the actual space is not returned to the requestor, but rather, a reference to the space's location. This is due to the fact that LSA spaces are large, complex, and non-sparse objects that actually have to be available on a fast medium, which suggests handling the actual space data internally and only exposing a 'space ID' to other functions.

During a typical NLP task, the service interface receives a user request holding the execution parameters, one of which must be the space (again, space handling via space locators is only a suggestion). After pre-processing the user parameters and data, an internal LSA logic is invoked, returning a result object to the communication controller.

Note that the typical NLP task allows a parallel execution of space retrieval and user data pre-processing, while the space generation is a pipelined operation.

2.2.3 Component Interaction and Distribution View

This view augments the layered view by looking closer at the interface structure as depicted in Figure 5.

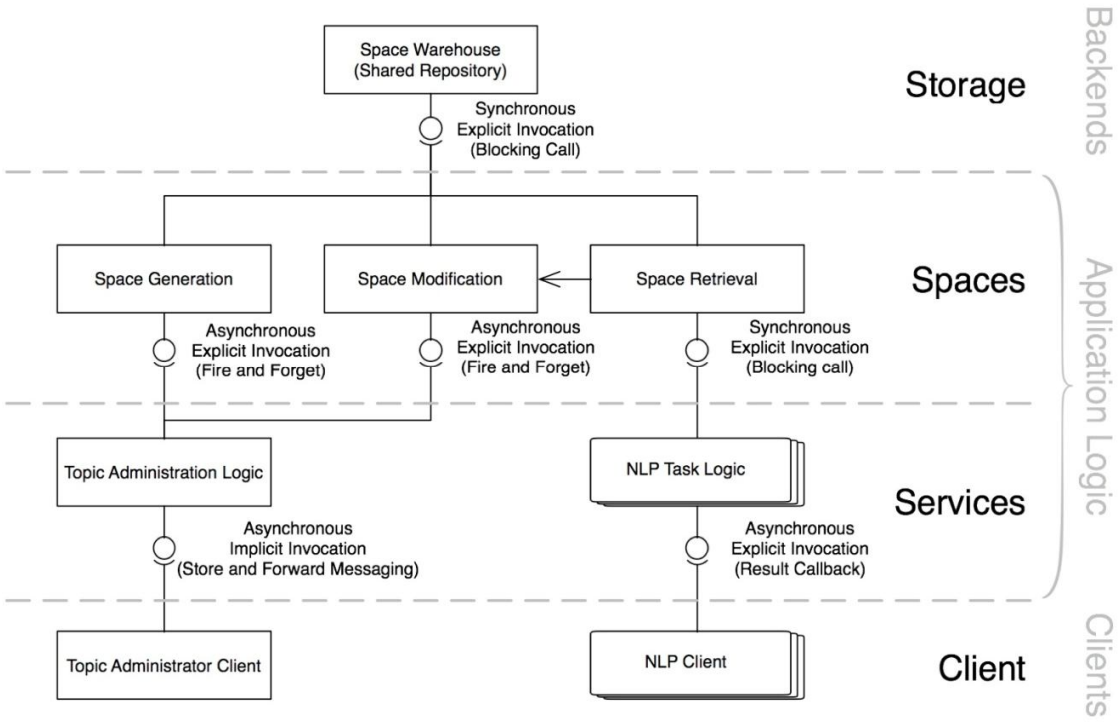

Figure 5. Component Interaction and Distribution View.

Again, a 4-tier-layering has been chosen. The grey annotations on the far right show that this model adheres to a typical 3-tier-client-server-architecture, with the application logic layer comprising the higher-granular services layer that holds the logic and interfaces of the task-specific calculations, and the spaces layer that holds the logic and interfaces for space maintenance and retrieval. This view describes the space object storage as a 'backend', which again emphasises the support function of this layer.

Component interaction is depicted with the invocation types next to the interfaces. Starting at the least-granular layer, the space warehouse holds objects, which are essential for the execution of the accessing components' logic and therefore, retrieval of the space blocks the accessing component until completion.

Space generation and modification (the latter relying on the component of space retrieval) are a time consuming task, and most likely no client will want to wait for its completion. Therefore, together with the topic administration logic (addressed, e.g., using a web service) a store-and-forward-messaging architecture is suggested. The topic administrator client sends a request object (including parameters and data) to a queue managed by the topic administrator logic, and receives nothing but a confirmation of receipt at the queue. The topic administrator logic then retrieves the topmost element in the queue as soon as processing capacity is available and forwards it to the space generation/modification

logic, using a 'fire and forget' invocation. At any time of this process, the topic administrator client can access information about the progress by accessing the space retrieval logic via the modification logic.

Finally, the clients of the NLP tasks (depicted by a stack in Figure 5, as there can be many different tasks, addressed by different specific clients) access their underlying logic via their respective service interfaces, using arbitrary remote invocation methods, most likely, (a)synchronous explicit invocations. The respective logic components then access the space retrieval component using a blocking call, as again, the spaces are vital for the calculations.

From the distribution view, Figure 5 shows the different remoting approaches used for the topic administration on the one hand and the NLP task execution on the other. The topic administration uses a message queuing remoting pattern, for the reasons described in the component interaction view above. The invocation of task logic is realised using the remote procedure calls remoting pattern.

From the user interaction view, the relevant layers 'clients' and 'services' interact utilising a model-view-controller (MVC) pattern, the model being the NLP task logic or the space maintainer, respectively, the view being the client software which sends request to the service interface, and the controller being the infrastructure used to provide this interface (e.g. an XML or SOAP service).

2.3 Example: Classroom Widget for Essay Scoring

The following scenario has been created as a prototype, with the aim of discovering a first set of technologies that may be used to realise the service architecture concept of Section 2.2.

Essay scoring is a process in which a topic is defined by a tutor using text corpora specific to this topic, and essays written by students can then be rated using a scoring mechanism. This prototype uses LSA to generate a space for the topic and to fold in a student's essay, finding the score using Pearson correlation as a proximity measure.

On the server machine, an Apache server is listening for REST-style requests for *.rws scripts, which are R scripts that can be executed by an Apache module called Rapache. These scripts execute the request and return custom XML data as a result.

2.3.1 Tutor's view: Corpus and Topic Administration

Corpus and topic administration is realised using a PHP-based frontend to generate the requests. Using a PHP command as shown in Listing 1, PHP generates a request like in Listing 2 at runtime. The server will return a list of existing corpora as in Listing 3:

```
$requestURL = 'http://host.com/webservice/corpus_list.rws';
$xml_response = file_get_contents($requestURL);
```

Listing 1: PHP instructions used to generate a request

```
GET /webservice/corpus_list.rws HTTP/1.1
User-Agent: PHP/5.2.4-pl2-gentoo
Host: host.com
Accept: */*
```

Listing 2: HTTP request for a list of existing corpora

```
HTTP/1.1 200 OK
Date: Wed, 29 Oct 2008 12:55:27 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/xml

358
<WSR:webServiceResponse xmlns:WSR="http://www.w3c.org/WSR"
xmlns:ltfll="http://www.ltfll.org/">
<ltfll:corpus id="1">
<ltfll:title>Medical Texts</ltfll:title>
<ltfll:original_filename>med.all</ltfll:original_filename>
<ltfll:textsize>1114373</ltfll:textsize>
</ltfll:corpus>

<ltfll:corpus id="2">
<ltfll:title>CISI Test Texts</ltfll:title>
<ltfll:original_filename>cisi.all</ltfll:original_filename>
<ltfll:textsize>2561998</ltfll:textsize>
</ltfll:corpus>
</WSR:webServiceResponse>
```

Listing 3: XML response from the server

This XML data is then processed using PHP to generate a GUI for topic administration. Upload of a corpus is done using HTTP POST utilising the RFC 1867, which is commonly used by browser-based forms. The form itself has been generated by the PHP script and is then utilised by the client browser.

```
POST /webservice/corpus_upload.rws HTTP/1.1
Host: host.com
Content-Type: multipart/form-data; boundary=---------------------cc1b3257ba
Content-Length: 309

-----------------------cc1b3257ba
Content-Disposition: form-data; name="corpus[1]"; filename="test.txt"
Content-Type: text/plain

This is a simple text corpus.

-----------------------cc1b3257ba
Content-Disposition: form-data; name="title[1]"

Title of the test
-----------------------cc1b3257ba--
```

Listing 4: HTTP request for upload of a new corpus

```
HTTP/1.1 200 OK
Date: Wed, 29 Oct 2008 13:10:22 GMT
```

```
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/xml

9a
<webServiceResponse xmlns="WSR" xmlns:ltfll="LTfLL"><ltfll:success>The file
test.txt has successfully been saved.</ltfll:success></webServiceResponse>
```

Listing 5: XML response upon upload

Using these technologies (REST-style requests for a small set of parameters, RFC 1867 style POST-upload for corpora), all functionality from the tutor's view is implemented, providing a GUI for the topic management.

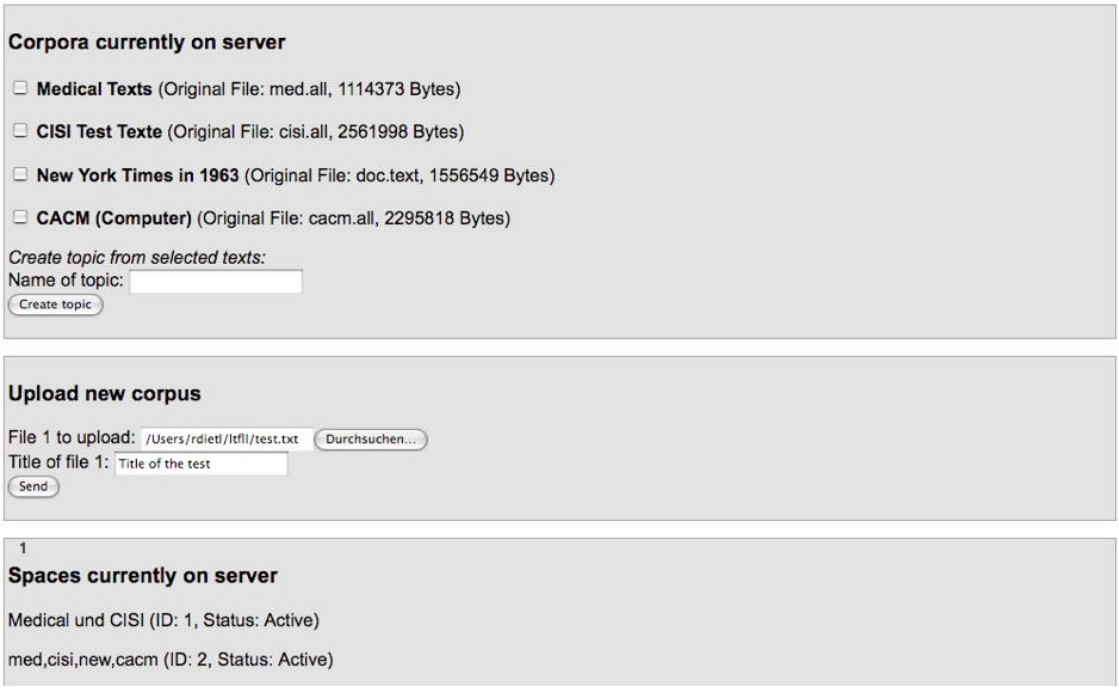

Figure 6. Screenshot of the PHP-based tutor GUI.

Space generation jobs are passed to the server using the message queuing mechanism outlined earlier. A GET request states the IDs of the corpora to be put into the space, and an R script on the server generates the space as soon as computation capacity is available, utilising the library 'lsa' (Wild, 2008). The spaces are then stored in a persistent R instance (using Rserve) that acts as the space object storage outlined in section 2.2. Therefore, the spaces are held in RAM and are highly available. The status of generation can be monitored using the GUI (see the bottom of Figure 6).

2.3.2 Student's View: Essay Scoring

For the student's side, an AJAX based GUI has been developed. It utilises the same technologies as the tutor's side. Creation of GET and POST requests is handled using the "Yahoo! User Interface" library (Yahoo, 2008b) module "connection", which allows for asynchronous invocation of the R services.

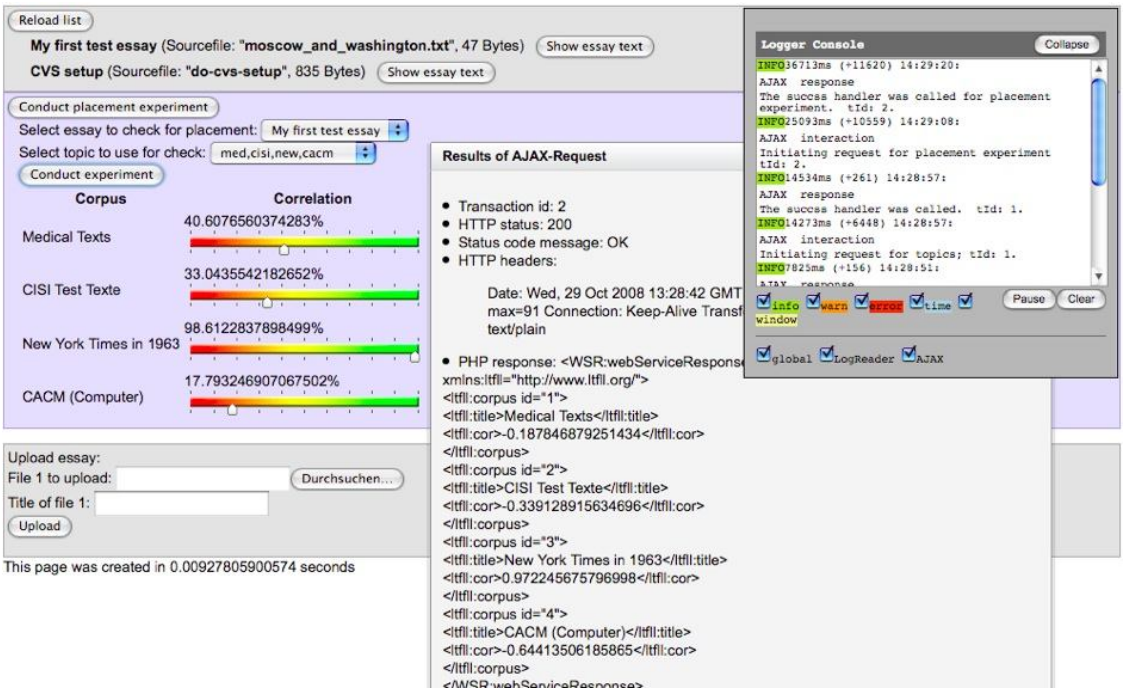

Figure 7. Screenshot of student's GUI based on the Yahoo! User Interface library.

2.3.3 Technologies used in Prototype

The technologies used for the prototype in 2.3. have been chosen for the following reasons:

HTTP 1.1

This protocol is very common for text transmissions over carrier media like the Internet. It has been preferred over different or even custom-tailored IP-based protocols for its wide distribution, the broad base of communication middleware and the openness of the standard, which results in better accessibility of the service.

GET/POST based RESTful requests

This technique has been chosen over alternatives like XML-RPC or SOAP for its simplicity. The communication features required for this prototype could easily be handled by HTTP utilising the custom syntax used here. XML-RPC would have allowed for a more programmatic approach to service interaction, but would have imposed the overhead of strict parameter typing in an environment that is not too sensitive for data types. SOAP would have imposed even more overhead, actually wrapping the current XML structure into just another structure that does actually not yield any benefit for this prototype. For the corpus management, both techniques would have lead to the problem of encapsulating text corpora in an XML structure. There are techniques for achieving this like soap-attachment, but for the reason of simplicity, the widely used RFC 1867 has been chosen here.

Custom XML responses

This method of encoding the results using a custom XML scheme has been chosen over serialisation of the result object using WDDX or JSON since the result objects R returns might not be easily converted to the data type used by the client software. The method chosen allows for easy communication using a simple, understandable response syntax that can be parsed in almost every software framework.

Persistent R instance as space object storage

This method has been chosen over storage of spaces on the hard disk for the almost instant availability of the spaces (as they are kept in RAM). It has been chosen over transfer of space objects to the client for reasons of data size (spaces are large, complex objects), potential serialisation problems, and again, availability (transfer of data over remote carrier vs. RAM access).

3. Interoperability for Learning Tools

To meet the needs of a Scenario Driven Design Progress in a heterogeneous system it can help to find an overall conceptual model to create a more general approach. A conceptual model is a model that describes a system in a common way. It serves to specify the design by the needs of the user.

In the PALETTE project a very interesting advance has been made base on a model called 3A. The three As stand for Actors, Activities and Assets. An Actor is producing an Asset being within an Activity. An Actor is a person, a software agent or any other intelligent object. An asset is a document or a collection of documents or items: discussion thread, wiki page, image album, and the like. An activity describes a formalisation of a common objective to be archived by a group of actors: representation of a tangible or abstract space: classroom or project management environment. The structure is similar to a graph: nodes (AAA, so called entities) are connected with several directed or undirected links with a specific type and weight (Bogdanov, 2008).

A different approach has been developed in the context of the iCamp project. At the core of this approach stands learning environment design which manifests in a learner interactions scripting language (LISL) and a prototypical implementation, called Mash-UP Personal Learning Environment (MUPPLE). LISL gives end-users the possibility to direct manipulate the composition of their personal learning environment. A simple learner interaction model has been deduced to describe the physical and social environment of learners. The activities, an actor is engaged in, are composed of actions that include tools, artefacts (objects), and other actors. A learning situation is represented by an activity that consists of actions that refers to objects and requires tools. With the help of LISL, learners manipulate actions, artefacts, and tools. Each action is bound to an artefact and at least one tool and produces one tangible or intangible outcome. Triples of actions, artefacts, and tools across activities and actors form learning networks within the MUPPLE platform (Wild, Mödritscher, and Sigurdarson, 2008).

For the LTfLL environment, these two models have been adapted to a more suitable one that serves as a real interface between development and scenarios. For sure actors need to be defined as well, but as one actor can be acting in different roles it seems to be obvious to use roles instead. The role defines which kinds of users are needed. Learners and educators do not use the tool the same way. Input and Output vary. There are two main activities, independent of the functionality of the tool. An actor produces input and expects output. Actors produce data in various ways, as a document or paper, in a forum or chat: anything provides input. Independent of how the input has been processed, the output has to be visualised browser-friendly. So after having defined who is doing something, it has to be defined, what the person is putting into the system, and what he gets back. It can be seen as a mash-up of asset and activity. A third important point is the environment. Some tools have special needs concerning the environment and the embedding. There is the possibility to communicate and interact with tools that are completely independent other services will be embedded in a frontend. Tools that are all stand-alone at least have to share their interface specification.

Exemplification:

Activity	Role	Input	Output	Widget
Feedback	Learner	Essay	Score; Feedback	APEX
Conceptual Development	Facilitator	Known mis-understandings; Learner Evidence Material	Learner Progress Report	Progress Monitor

Figure 7. Conceptual Model (Example).

3.1 Data Gathering

Users generate, edit, or delete data that has to be processed. According to Tim O'Reilly, RSS is one of the most significant advances in web architecture (O'Reilly, 2005).

So it seems obvious to use RSS for data transport to profit from its advantages. An RSS feed is an XML file that contains text and additionally metadata like the author or a description.

As it is necessary to delete data, RSS is not sufficient. It is strongly recommended to use Atom Syndicate Format (Nottingham & Sayre, 2005) as well which goes along with the Atom Publishing Protocol and supports all the core functionalities of the HTTP protocol like GET, PUT and DELETE. All input data will be feeds.

RSS 2.0	Atom
No XML schema	Contains XML schema
Description tag can contain summary or content	Separation between summary and content
Date can be any format	Date has to comply with RFC 3339 format
No mark if it's plain text or HTML	marked if it's HTML or plain text
No support for relative URIs	Supports relative URIs
No ID	Every entry has a unique ID
widely spread	Not so popular yet

Figure 8. Conceptual Model (Example).

The use of Yahoo pipes (Yahoo, 2008a) could be reasonable to filter the RSS or Atom Feeds before processing them. Yahoo Pipes is a web application from Yahoo that allows combining, sorting, filtering and translating feeds.

To get all the available information, you have to be aware of new data. With the use of Yahoo pipes you can update this information periodically. There is also the possibility to use FeedBack (Wild & Sigurdarson, 2008). FeedBack is a specification on how to support management processes for feeds, i.e. provides facilitates to manage the data flow from sources to sinks.

3.2 Widgetising

A widget is a small program embedded in a graphical user interface. There is a need for tools that are adapted so that language services developed by the different partners are fed with data from the tools (e.g. forum or blog entries) and the services return feedback e.g. in the form of a widget.

Widgets are small applications that are embedded in kind of a framework or widget engine. There are two possibilities provided: Widgets that are able to communicate directly with the server and are interactive or widgets that get their information periodically with the help of feeds which are not interactive. To stay as flexible and as open-ended as possible the output can be a pure RSS Feed (or Atom) as well. In this way, information remains independent of any tool or widget. Furthermore this offers the possibility to use feed output as input for other applications.

Most of the time, widgets are tools or some kind of help or service application. Widgets first arose in operating systems such as Apple's dashboard widgets. Parallel to this development was the appearance of web widgets, mainly to serve as a container for information from any external source. In the world of Web 2.0 widgets are often used to embed photos or videos, as with Flickr or YouTube.

Widgets are written in HTML and JavaScript. A widget engine is needed to host a widget in an environment. There are a lot of such environments already, the natural choice being another web-page, an approach that, for example, iGoogle follows. But even modern operating systems support widgets natively, for example Microsoft Window Vista ('SideBar') and Apple's Mac OS X ('Dashboard'). On the World Wide Web there are certain platforms that provide this functionality such as iGoogle, Facebook, netvibes, pageflakes, and others.

For LTfLL, an LMS or CMS frontend is assumed to be the runtime container in which widgets are executed. If the LMS or CMS doesn't provide this functionality by itself, this can be achieved with the help of DHTML, AJAX, Adobe Flash or Java-Applets.

It is also possible to use something similar on the server side. In that approach, the HTML is dynamically put together by the web server instead of the client's browser. That approach is older, from the times when the client browsers were not that capable as they are today. This approach is taken by Java Portlets and WSRP. WSRP can be used in SharePoint Portal Server, amongst others. More suitable, however, is a widget-based approach which is easily transportable so that it can be easily integrated in a broad range of containers.

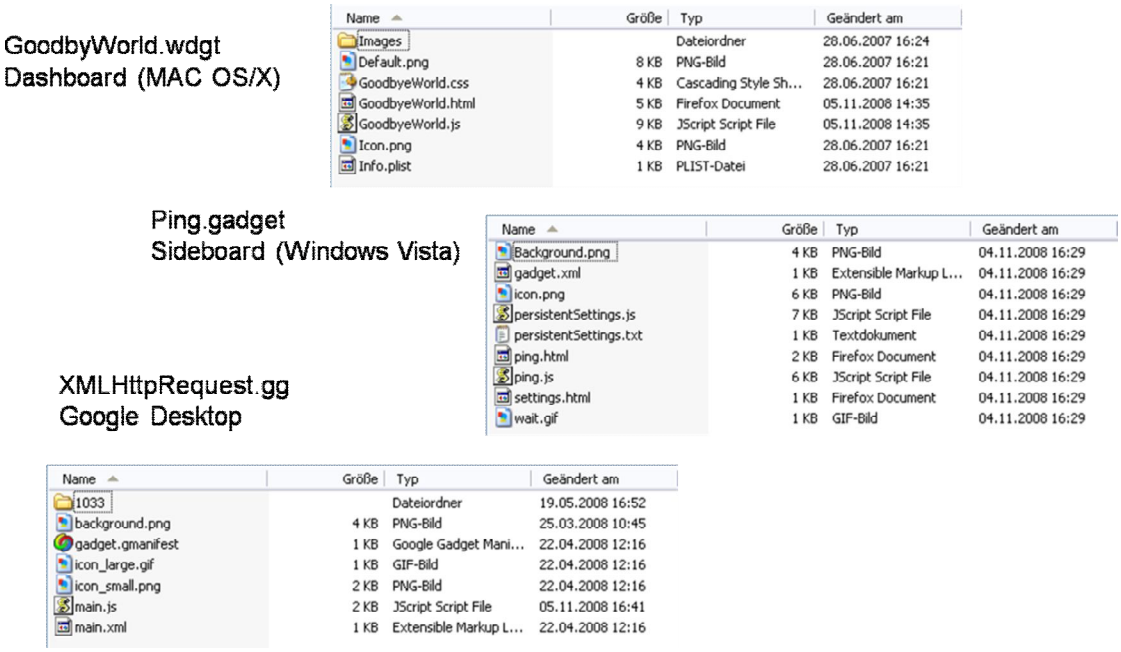

Figure 9. Example of Common Widget Structures.

Most widgets have certain things in common:

- Manifest file: This is the configuration file. Here the name of the widget, the author, the size, the ID, or anything else can be configured here.
- Media type
- Packaging format
- APIs
- Resources
    - XML or HTML file: in any case an index.html (or xml) file is needed that contains valid XML/HTML(see Listing 6).
    - JavaScript (see Listing 7)
    - Images
    - CSS

A detailed specification will be provided for LTfLL widgets based on the widget 1.0 working draft (Caceres, 2008).

Example (Google Desktop gadget):

```
<view height="150" width="250">
  <img x="4" src="stock_images\background.png"/>
  <div enabled="true" height="22" hitTest="htclient" width="111" x="126"
    y="26" onclick="div_onclick()" background="#000000">
    <combobox    height="100"    name="combobox"    width="109"    x="5"    y="2"
background="#FFFFF1"
      itemHeight="20" itemOverColor="#CCFFCC" onchange="onChange();"
      maxDroplistItems="4" type="droplist">
      <item height="20">
        <label valign="middle">Berit
        </label>
      </item>
      <item height="20">
        <label valign="middle">Gerhard
        </label>
      </item>
      <item height="20">
        <label valign="middle">Martin
        </label>
      </item>
    </combobox>
  </div>
  <img height="100" name="imageControl" width="100" visible="false"
    x="18" y="13"/>
  <script src="main.js" />
</view>
```

Listing 6: Main.xml (kind of index.xml or index.html, varies from widget to widget)

```
var URL = null;
var imgRequest = null;

//On chaning the choice in comobox, the Index of the selected item gives the
needed URL
function onChange() {
             switch (combobox.selectedIndex) {
      case 0:
       var URL = "http://partners.ltfll-project.org/user/pix.php/60/f1.jpg";
        break;
      case 1:
        var URL = "http://partners.ltfll-project.org/user/pix.php/67/f1.jpg";
        break;
      case 2:
        var URL = "http://partners.ltfll-project.org/user/pix.php/33/f1.jpg";
        break;
       }

  try{
             imgRequest = new XMLHttpRequest();
             imgRequest.open("GET", URL, true);
             // Set the callback for when the downloading is completed (or
failed)
             imgRequest.onreadystatechange = loadImg;

             // Start the download
```

```
            imgRequest.send();

 // Obtain the stream of image data and set it as the image.
            imageControl.src = imgRequest.responseStream;
            imageControl.visible = true;

  // Destroy the XMLHttpRequest object since it isn't being used anymore
            imgRequest = null;
      }
      catch(e){
            // Catch errors sending the request
            imgRequest = null;
            return;
      }

}

function loadImg(){
  // Verify that the download completed
try{
      if(imgRequest.readyState !=4)
            return;
  // Verify that the download was successful
      if(imgRequest.status != 200)
            imgRequest = null;
            return;
}
catch(e){
}
}
```

Listing 7: Main.js

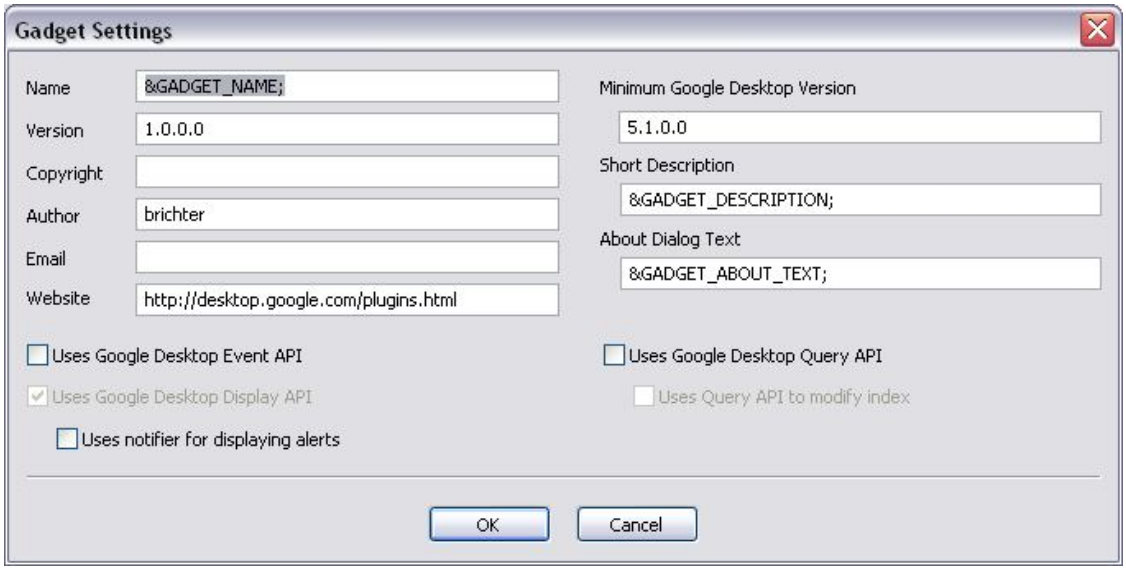

Fig. 10. Gadget settings.

```
<gadget minimumGoogleDesktopVersion="5.1.0.0">
	<about>
		<name>&GADGET_NAME;</name>
		<description>&GADGET_DESCRIPTION;</description>
		<aboutText>&GADGET_ABOUT_TEXT;</aboutText>
		<smallIcon>icon_small.png</smallIcon>
		<icon>icon_large.png</icon>
		<version>1.0.0.0</version>
		<author>brichter</author>

	<authorWebsite>http://desktop.google.com/plugins.html</authorWebsite>
		<id>AD9F5FE2-77F7-46E5-BEFD-BF7E66520C1F</id>
		<copyright></copyright><authorEmail></authorEmail></about>
</gadget>
```

Listing 8. Gadget settings

3.3 Runtime Environment: Glueing and Executing

To specify the behaviour of the web services there has been the idea of using BPELhttp://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (Business Process Execution Language, see Jordan & Evdemon, 2007). BPEL is an XML based language that is often used to orchestrate web services, so partners could configure the educational tool landscape for a specific scenario e.g. by using Eclipse and the BPEL plug-in.
One major point is the integration of the existing learning tools (e.g. Wikis, Forums) into the LTfLL learning environment. Generally there are various options available for doing this:

a) Hard-wired by a configuration file or a scenario manager: e.g. a user selects a scenario, manager service launches tools. A state-of-the art approach would be to use web application frameworks, like Spring (SpringSource, 2008) or OpenACS (OpenACS Community, 2008), probably also Apache/PHP-based frameworks.
b) Separate applications without any interactions.
c) Runtime orchestration of tools, connecting tools to background services on-the-fly like Spring, Openwings, OpenACS, or even BPEL-based workflow engines like ActiveBPEL or the YAWL-engine. There is also the option of using IMS-Learning Design (IMS, 2008) to orchestrate learning services which can be evaluated.

Regarding the requirements that have arisen until now, it seems to be sufficient to use a hard-wired configuration with a configuration file on the server. This approach complies very well with the idea of a scenario driven design process (Armitt et al., 2008). In a configuration file there would also be the possibility to configure everything else that is needed like the task scheduler frequency or the URL where the service are found.

```
<xml>
<application>
< task scheduler run =”hourly” command=”$SVN/latestpost.php”/>
<administration group=”wp2”>
<title>Latest Posts</title>
</administration>
<widget url=”latestpostwidget”>
<title>Show Latest Post</title>
</widget>
<widget url=”topactivitywidget”>
<title>Top Activity</title>
</widget>
</application>
</xml>
```

Listing 9: Example configuration file

The different tools that are the results of the activities of work packages 4, 5 and 6 are hosted on the server and executed there. The server must offer the option of getting the data from the feeds and of generating feeds to give some output back. The configuration file is committed over SVN, so there is no need for the project members to have direct access to the server and other servers (VM images) can easily be configured too. In the configuration file the producer of a tool has to provide the start URL of the application. On the server a database is provided which makes all necessary information available to the user - in this case the user will be the widgets – so only one database user is needed. It is possible for the application to register to a task scheduler periodically: daily, hourly, or user-defined.
For the different services, that are used by different users at the same time instances of the services could be needed. There will be the possibility for administrators to create new ones. Users will have this possibility as well. Every widget carries information about the need for instances as a parameter.
The runtime environment represents a container that glues together the widgets. Certainly there will be further demands that the platform will have to meet, but these requirements will arise during the development of the tools and techniques and cannot be specified yet.
As frontend for the users a platform is needed that is easy to use, flexible and open-ended. It must also have a clean API, support feeds and provide a forum, chat and blogs. There are several open source platforms that comply with these requirements, LMS/CMS systems like Moodle, Joomla, Ilias or weblog systems like WordPress.

Example:

Requirement	Moodle	Joomla	Ilias	WordPress
Feed support	x	x	x	x
Chat, Forums	x	Plug in	x	Plug in
Community management	x	x	x	-

Clean and well documented API	x	x	x	x
Open-ended	x	x	x	x

Fig. 11. Possible Environments.

At this point it makes no sense to choose one system as the most suitable one because not all of the requirements are known. There is the need for a continuous evaluation of the requirements to be able to choose the right system at the right time. The requirements are expressed by the individual work packages and we have to evaluate whether a platform suits these needs or not.

3.4 Authorisation and Authentication

There is the need for an API to identify the user. The user is granted access to a rather personal environment where his activities are noted. To be able to log in at all services at the same time the use of a single sign on system is recommended. There are several different Single-Sign-On systems that have one thing in common: They provide a possibility to have only one ID, or one password to sing-on to several systems. In addition, some Single-Sign-On systems also provide a Single-Log-Off function that automatically logs you off all services you have been logged in. After being authenticated the next step is to be authorised. This can be depending to a role: learner, teacher, tutor, admin, or anything else (Sams, 2008).

For Single-Sign-On there are several providers that offer systems. We have to consider whether the Google API Open Social that uses OAuth (OAuth Community, 2008) could help in this case. OAuth is an open protocol to allow secure API authorisation in a simple and standard method (Google, 2008).

Another possibility would be the use of OpenID, a well known single sign-on system that is open source. Open ID has been developed for the huge community of online users to eliminate the need of multiple user-names and passwords. OpenID is easy to install and widely spread. Because of the popularity of OpenID the support of this system is highly recommended (OpenID Foundation, 2008).

Depending on the requirements of the project there could arise the need for a system like Shibboleth. Shibboleth provides both, authentication and authorisation for web services and web applications. It is based on an extension of the SAML (Security Assertion Markup Language) standard. Shibboleth is a rather institution centric approach where one institution acts as broker (OASIS, 2008).

3. Development Guidelines

An infrastructure serves as a framework to support both development and deployment encompassing both hardware and software aspects. In this chapter, these hard- and software specifications are outlined, the software development and release process is sketched, and guidelines regarding testing and compliance are elaborated.

One of the tasks of work package two is to serve as kind of a service provider for the other work packages. While it is our task to guide, it is the task of the other work packages to communicate their requirements. While some of the needs will still arise, some rather basic decisions can already be made: Server Specification, general Software Development and Release Process, Documentation and Software Testing. For the development of the other work packages, this information is the general set-up, the development guidelines.

3.1 Server Specifications

There are two physically independent infrastructures needed within the project. One to host classic web-applications, the other to equip the language processing services with the computational power, big memory resources, and particular software or operating system requirements needed.

Learning Tool Infrastructure

To set up the server on the basis of the requirements provided by the project partners, a server setup documentation is created. Following the documentation it is possible for all project members to get and run a virtual machine image of the staging server themselves (provided by bit media).

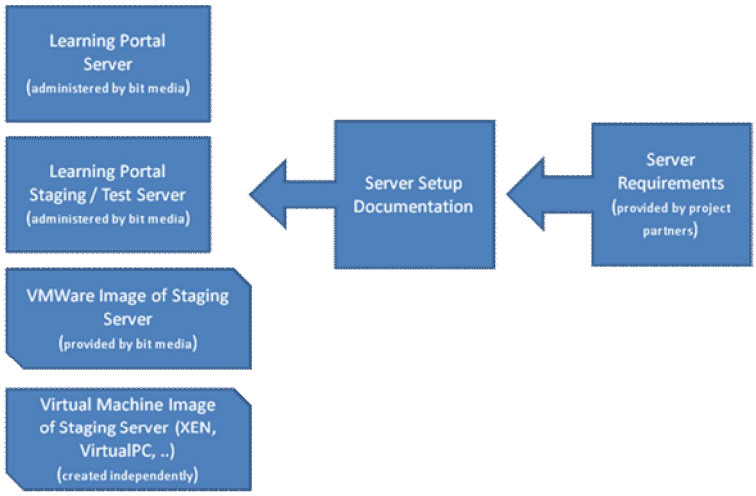

Fig. 12. Workflow Server Setup.

There are two servers, one acting as the live (‘Mittelerde’), the other as the development system (‘Elch’). The development system doesn’t only serve as test server for new code

or frameworks; it also helps to grant transferability, as it is not equal to the live system. The specification of the servers is as follows, there will be some adaptations to be made for LTfLL needs:

- Live System ('Mittelerde')
  - Hardware
    - Fujitsu Siemens Primecenter, 19" architecture
    - XEON Dual Processor, 2400MHz, 4GB Ram
    - 2 application servers, 2 database servers
    - Data back-up: Grandfather-Father-Son Backup[8]
    - Database back up
    - Capacity: 2x32 GB in Raid 1 each server + SAN (Storage Area Network): 2x2 64 GB in Raid1
  - Software
    - Application server
      - Suse Linux
      - Apache 1.3.29
      - PHP 4.4.6
    - Database server
      - Suse Linux
      - MySQL 4.1.13
- Development System ('Elch')
  - Hardware
    - Windows Server 2003
    - Intel Core2 CPU (2,4 GHz)
    - 4 GB RAM
    - 2x 320 GB HDD
  - Software
    - IIS 6
    - PHP 4.4.9
    - MySQL 4.1.20
    - MSSQL 2005
    - Oracle 10g

Natural Language Processing Infrastructure

As there are several partners who are deploying NLP related software, a similar developing infrastructure method is chosen like for the learning tool infrastructure part. To ensure that testing takes place and interoperability standards are met throughout the development process two separate systems exist, one acting as a development/test, the

[8] One of the most popular methods: three sets of backups are defined, such as daily, weekly and monthly

other as a live system. For the implementation it was chosen to install two virtual servers (using XEN virtualisation) running on one physical machine mainly for cost reasons. The most important specifications are:

- Hardware
  - 2x Quad-Core Xeon 2.8GHz: 1 Quad-Core for each VM
  - 32 GB RAM (8x4GB dual rank DIMMs): 16 GB for each VM
  - 1.8 TB HD (6 x 300 GB)
- Software
  - Operating System: Debian Lenny (64bit)
  - Apache httpd 2.2.10
  - MySQL Community Server 5.0.51a-17 (Debian)
  - PHP 5.2.6 featuring Zend Engine v2.2.0
  - R 2.8.0 with GotoBLAS 1.26

3.2 Software Development and Release Process

To host the code during the development and to share it with other project members we use SourceForge. A SourceForge project for LTfLL has been created and can be found here:

http://sourceforge.net/projects/ltfll/

To become a member of the project you just have mail with project admin (listed on the start page) stating your SourceForge account name. It is also possible to authenticate using OpenID.

SourceForge is the well-known open source web portal for distributed development. It provides not only the possibility to host and share code, but also to manage the development process offering functionalities such as message boards and bug tracking.

SourceForge supports version management handled by Concurrent Versions System (CVS) or Subversion (SVN) For LTfLL the use of SVN is proposed for several reasons as for example: lower traffic as only differences between the versions are transferred in both directions, better client tools for cross-platform use, wide distribution and the use of revision numbers; other solutions like Darcs, Git or Mercurial might be better suited, but are not supported by SourceForge (CollabNet, 2006).

Every SVN project is divided into three folders: trunk, branches, and tags. In the ‘trunk’ you find the current code, ‘branches’ are stable copies of the trunk where further, alternative, development can take place, and ‘tags’ are used in order to mark certain versions. In ‘trunk’ every work package will have its own folder so that the developers have the possibility to host their code and documents organised in the way they need it. Every developer would have the option to use working branches that can be merged to a stable branch, but with the separation in folders for every work package this was not considered necessary. All code for the server and the configuration is located in the WP2 folder. In addition, there is a folder for work that is common to all WPs.

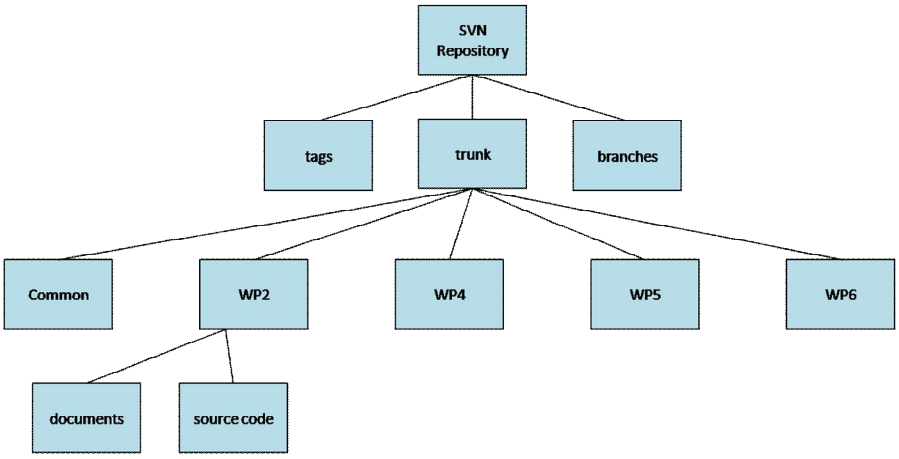

Fig. 13. SVN repository tree.

SVN is used to support the internal software development process. For releasing software components SourceForge offers functionalities for package releases. This means at a certain point in time a stable version of the software in the SVN repository is packaged and released to the public. Software releases within this project are defined to be done in line with the submission of the deliverables of the different WPs. The release process will be deployed internally to test the installations on the live servers.

3.3 Documentation

If any third party library is needed, this has to be announced via mail to the server administrators to be added to the requirements and to the documentation. The same procedure also applies to additional software or configuration needs. Software code which arises from the LTfLL project must be well documented. Doxygen (van Heesch, 2008) could help here to auto-generate the documentation from the code. Doxygen is an open source tool that uses in-code comments to create a well-structured documentation. Doxygen runs on Linux, but there is also a windows version available. C/C++, Java, Python, IDL, C#, Objective-C, and to some extent D and PHP sources are supported. Online formats (HTML and UNIX man page) and off-line formats (LaTeX and RTF) are available as output.

Technical documentation is required especially with respect to:

a) Installation Documentation (internal): what is installed on the server, how to access it (inclusive of all passwords)
b) Installation Documentation (external): what is installed on the server, how to access it (exclusive of all passwords)
c) Code Documentation: in code documentation, reference, vignette-like technical documentation

3.4 Software Testing

A very important issue in software development is the use of unit tests. A unit test is a small piece of code written by a developer to test the functionality of the code; in fact, it tests the smallest testable part of the source code. Depending on the code structure this could be a function, a method, or any other part of the program. With the use of unit tests it is easy to prove the functionality of software very quickly and at any time.

Unit tests are usually written before the real code is implemented to specify what the real code is intended to do. In addition, unit tests make debugging easier and behave as executable documentation. Generally, every time a new function has been written or a bug has been fixed unit tests have to be run. For LTfLL development, tools are tested by their developers. The APIs, for example, are tested with unit tests, such as probing if an API is available and returns any results.

A virtual machine of the server will be provided to all partners so that they are able to work and test locally. Every time something new has been installed, it is a good idea to test it on a virtual machine before doing so on the real one.

3.5 Compliance

To ensure **transferability**, i.e. to ensure that the developed software will run on all major systems, minimum requirements on software and hardware components have to be defined. As the developed learning tools are rendered using a web-browser, some generic guidelines can be defined. By optimising user interfaces for Microsoft Internet Explorer version 6 + 7 and for Mozilla Firefox, typically a range of clearly over 90% of all Internet users is covered. Furthermore, over 95% of all users have JavaScript enabled (Refsnes Data, 2008a). If web-based software is designed for a screen resolution of 1024x768 pixels and higher over 85% of the users are covered (Refsnes Data, 2008b). Regarding operating systems, over 90% of the users are working on a Microsoft Windows system (Vista, XP, 2000, 2003 or 98) (Refsnes Data, 2008c).

The XHTML and CSS **standards** can be validated using W3C's validators (XHTML: W3C, 2008b; CSS: W3C, 2008a). In the knowledge rich approaches, the standards covered with the GRDDL specification (Gleaning Resource Descriptions from Dialects of Languages) may serve useful. The GRDDL specification introduces mark-up based on existing standards for declaring that an XML document includes data compatible with the Resource Description Framework (RDF) and for linking to algorithms (typically represented in XSLT), for extracting this data from the document (Connolly, 2008).

The tools need to be able to cooperatively exchange data in order to support the successful accomplishment of the envisioned use cases. In several cases, ensuring this **interoperability** may relate to providing RSS of Atom feeds or implementing the feed management API FeedBack. The feed management API FeedBack can be validated using (iCamp, 2008). Additional data gathering standards or service APIs may become necessary during the project's lifetime.

Each service deliverable will cover compliance issues in more detail.

References

Advanced Distributed Learning (2007). *Advanced Distributed Learning - SCORM*. Retrieved November 24, 2008, from http://www.adlnet.gov/scorm/

Armitt, G., Braidman, I., Tim, D., Jan, H., Howard, S., Gaston, B., et al. (2008). *LTfLL_D71_Validation_design_final_version.doc*.

Avgeriou, P., & Zdun, U. (2005). *Architectural Patterns Revisited - A Pattern Language*. 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), (pp. 1-39). Irsee.

Bird, S., Day, D., & Garofolo, J. (2000). *ATLAS: A Flexible and Extensible Architecture for Linguistic Annotation*. Proceedings of the Second International Conference on Language Resources and Evaluation (pp. 1699 - 1706). Paris: European Language Resources Association.

Bontcheva, K., Kiryakov, A., & Cunningham, H. (2003). *Semantic Web Enabled, Open Source Language Technology*. Proceedings Language Technology and the Semantic Web, Workshop on NLP and XML (NLPXML-2003). Budapest: Advanced Knowledge Technologies.

Caceres, M. (2008, April 14). *Widgets 1.0: Packaging and Configuration - W3C Working Draft*. Retrieved November 24, 2008, from World Wide Web Consortium: http://www.w3.org/TR/widgets/

CollabNet (2006). Subversion. Retrieved November 24, 2008, from http://subversion.tigris.org/

Connolly, D. (2008, September 8). *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*. Retrieved November 25, 2008, from Wolrd Wide Web Consortium: http://www.w3.org/2004/01/rdxh/spec

Cover, R. (2000, November 16). *Architecture and Tools for Linguistic Analysis Systems (ATLAS)*. Retrieved November 24, 2008, from The CoverPages: http://xml.coverpages.org/atlasAnnotation.html

Cunningham, H., Maynard, D., & Bontcheva, K. (2008, November 11). *Developing Language Processing Components with GATE Version 5 (a User Guide)*. Retrieved November 24, 2008, from GATE - General Architecture for Text Engineering: http://www.gate.ac.uk/sale/tao/index.html

Cunningham, H., Maynard, D., & Bontcheva, K. (2002). *GATE: an Architecture for Development of Robust HLT Applications*. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (pp. 168 - 175). Philadelphia: Association for Computing Machinery.

Bogdanov, E., Salzmann, C., El Helou, S., Gillet, D. (2008). *Social Software Modeling and Mashup based on Actors, Activities and Assets.* In: Wild, Kalz, Palmer (Eds.):

Proceedings of the First International Workshop on Mash-Up Personal Learning Environments, Maastricht, The Netherlands, Retrieved November 26, 2008, from http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-388/bogdanov.pdf

Feinerer, I., & Wild, F. (2008, September 16). *CRAN Task View: Natural Language Processing*. Retrieved November 24, 2008, from The Comprehensive R Archive Network: http://cran.at.r-project.org/web/views/NaturalLanguageProcessing.html

Wikipedia (2008). *General Architecture for Text Engineering*. Retrieved November 24, 2008, from http://en.wikipedia.org/wiki/General_Architecture_for_Text_Engineering

Google. (2008). *OpenSocial*. Retrieved November 24, 2008, from Google Code: http://code.google.com/apis/opensocial/

Hartmann, B., Doorley, S., Klemmer, S.R. (2008): *Hacking, Mashing, Gluing: Understanding Opportunistic Design*. In: Pervasive Computing 7(3), pp.46-54, IEEE

Hornik, K. (2008). *R FAQ - Frequently Asked Questions on R*. Retrieved November 09, 2008, from The R Project for Statistical Computing: http://cran.r-project.org/doc/FAQ/R-FAQ.html

IBM (2006). *Unstructured Information Management Architecture (UIMA)*. SDK User's Guide and Reference. New York.

IBM (2008). *Unstructured Information Management Architecture*. Retrieved November 24, 2008, from IBM alphaWorks: http://www.alphaworks.ibm.com/tech/uima

iCamp (2008). *FeedBack Validator*. Retrieved November 25, 2008, from http://isdev.odg.cc/feedback/

IMS (2008). *IMS Global Learning Consortium: Learning Design Specification*. Retrieved November 24, 2008, from IMS Global Learning Consortium, Inc.: http://www.imsglobal.org/learningdesign/

jATLAS Development Team. (2003, October 15). *jATLAS: a Java implementation of ATLAS*. Retrieved November 24, 2008, from ATLAS overview: http://jatlas.sourceforge.net/index.html

java.net Community. (2008, November 04). *Metro Users Guide*. Retrieved November 24, 2008, from GlassFish - Open Source Application Server: https://metro.dev.java.net/guide/

Jordan, D., & Evdemon, J. (2007, April 11). *Web Services Business Process Execution Language Version 2.0 - OASIS Standard*. Retrieved November 24, 2008, from OASIS: Advancing open standards for the global information society: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Laprun, C., Fiscus, J., & Garofolo, J. (2002). *Recent Improvements to the ATLAS Architecture*. Proceedings of the second international conference on Human Language Technology Research (pp. 263 - 268). San Diego: Morgan Kaufmann Publishers Inc.

Lim, J. (2004). *ADOdb Database Abstraction Library for PHP (and Python)*. Retrieved November 24, 2008, from http://adodb.sourceforge.net/

Nottingham, N., & Sayre, R. (2005, December). *RFC 4287 - The Atom Syndication Format*. Retrieved November 24, 2008, from The Internet Engineering Task Force: http://tools.ietf.org/html/rfc4287

OASIS (2008). *OASIS Security Services (SAML) TC*. Retrieved November 24, 2008, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

OAuth Community (2008). *OAuth*. Retrieved November 24, 2008, from http://oauth.net/

OpenACS Community (2008). *OpenACS Home*. Retrieved November 24, 2008, from http://openacs.org/

OpenID Foundation (2008). *OpenID*. Retrieved November 24, 2008, from http://openid.net/

O'Reilly, T. (2005). *What Is Web 2.0*. Retrieved November 26, 2008, from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

R Development Core Team (2008a). *R Data Import/Export*. Retrieved December 1, 2008, from http://cran.r-project.org/doc/manuals/R-data.pdf

R Development Core Team (2008b). *R Installation and Administration*. Retrieved December 1, 2008, from http://cran.r-project.org/doc/manuals/R-admin.pdf

R Development Core Team (2008c). *R Language Definition*. Retrieved December 1, 2008, from http://cran.r-project.org/doc/manuals/R-lang.pdf

R Development Core Team (2008d). *Writing R Extensions*. Retrieved December 1, 2008, from http://cran.r-project.org/doc/manuals/R-exts.pdf

Refsnes Data. (2008a). *Browser Statistics*. Retrieved December 17, 2008, from W3Schools Online Web Tutorials: http://www.w3schools.com/browsers/browsers_stats.asp

Refsnes Data. (2008b). *Browser Display Statistics*. Retrieved December 17, 2008, from W3Schools Online Web Tutorials: http://www.w3schools.com/browsers/browsers_display.asp

Refsnes Data. (2008c). *OS Platform Statistics*. Retrieved December 17, 2008, from W3Schools Online Web Tutorials: http://www.w3schools.com/browsers/browsers_os.asp

Sams, B. (2008). *Single-Sign-On-Systeme - Eine Übersicht*. Retrieved November 24, 2008, from IT-Republik: http://it-republik.de/jaxenter/artikel/Single-Sign-On-Systeme-1499.html

SpringSource (2008). *Spring Framework*. Retrieved November 24, 2008, from http://www.springframework.org/

van Heesch, D. (2008). *Doxygen*. Retrieved November 24, 2008, from http://www.stack.nl/~dimitri/doxygen/

Venables, W. N., & Smith, D. M. (2008). *An Introduction to R. Network Theory*. Retrieved December 1, 2008, from http://cran.r-project.org/doc/manuals/R-intro.pdf

W3C (2008a). *The W3C CSS Validation Service*. Retrieved November 25, 2008, from http://jigsaw.w3.org/css-validator/

W3C (2008b). *The W3C Markup Validation Service*. Retrieved November 25, 2008, from http://validator.w3.org/

Wild, Fridolin; Mödritscher, Felix, Sigurdarson, Steinn (2008). *Designing for Change*. In: eLearning Papers 2008(9), http://www.elearningeuropa.info/files/media/media15972.pdf

Wild, Fridolin, Sigurdarson, Steinn (2008). *Distributed Feed Network for Learning*. In: Upgrade IX(3), CEPIS/ATI.

Wild, Fridolin, Sporer, Thomas, Chrzaszcz, Agnieszka, Sigurdarson, Steinn E., Metscher, Johannes (2008): *Distributed e-Portfolios to Recognise Informal Learning*. In: ED-MEDIA 2008, Proceedings of the 20th World Conference on Educational Multimedia, Hypermedia & Telecommunications, July, 2008

Wild, Fridolin (2008): *lsa: Latent Semantic Analysis*. R package version 0.61. Retrieved November 24, 2008 from http://cran.r-project.org/web/packages/lsa

Wild, Fridolin, Sobernig, Stefan (2006): *Interoperability Framework Draft for the Distributed Open Virtual Learning Environment*. Deliverable D3.1 of the iCamp Project, iCamp Consortium, Retrieved on November 30, 2008 from http://www.icamp.eu/wp-content/uploads/2007/05/d31___icamp___interoperability-framework-draft.pdf

Yahoo (2008a). *Pipes: Rewire the web*. Retrieved November 24, 2008, from http://pipes.yahoo.com/pipes/

Yahoo (2008b). *The Yahoo! User Interface Library (YUI)*. Retrieved November 24, 2008, from Yahoo! Developer Network: http://developer.yahoo.com/yui/

Appendix 1: Consortium Resources

- English
  - Chat protocols
  - Texts from student portfolios
  - Texts from WebCT discussion groups
  - Corpora in the field of psychology
  - WordNet
  - Stop words
- Romanian
  - Chat protocols
  - Blogs
  - Forum messages
- Dutch
  - Corpus from computing domain (200.000 items)
  - Cornetto lexical semantic database (40.000 items)
  - Spoken Dutch corpus (10.000.000 words)
  - D-Coi written Dutch corpus (500.000.000 words)
  - Stop words
- Bulgarian
  - Text archive (70.000.000 words)
  - TreeBank syntactic description (15.000 sentences)
  - Morphological lexicon (> 100.000 lemmas)
  - Gazetteer (> 15.000 nouns: person, location, organisation, other)
  - Bulgarian CLEF corpus
  - LT4eL Bulgarian corpus and lexicon
- German
  - Forum messages (> 100.000)
  - WordNet alike
  - Parsed Wikipedia version
  - German learning objects in the field of computer science (linguistically analysed)
  - LT4eL lexicon
  - Linguistically analysed reference corpus
  - Stop words
  - Collection of Essays and Human Scores
- French

    - Adult French Corpus (13.000.000 words), composed of
        - Children tales (3.300.000 words)
        - Newspaper articles (5.000.000 words)
        - Novels (5.000.000 words)
- Multi-language
    - Ontology (computing sector) from LT4eL
    - Term-concept lexicons in eight languages
    - English and German learning objects of course material (approx. 15.000)
    - Stemmer for many languages

Glossary

AIF	ATLAS Interchange Format
AJAX	Asynchronous JavaScript and XML
ANNIE	A Nearly-New Information Extraction system
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATLAS	Architecture and Tools for Linguistic Analysis Systems
CAS	Common Analysis Structure
CMS	Content Management System
CORBA	Common Object Request Broker Architecture
CPE	Collection Processing Engine
CPU	Central Processing Unit
CRAN	Comprehensive R Archive Network
CREOLE	Collection of Reusable Objects for Language Engineering
CSS	Cascading Style Sheets
CVS	Concurrent Versioning System
DCOM	Distributed Component Object Model
DHTML	Dynamic HyperText Markup Language
DIMM	Dual Inline Memory Module
GATE	General Architecture for Text Engineering
GB	Gigabyte
GHz	Gigahertz
GNU	GNU is Not Unix
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
GUI	Graphical User Interface
GUK	Gate Unicode Kit
HDD	Hard Disk Drive
HTTP	HyperText Transfer Protocol
IIS	Internet Information Services
JAPE	Java Annotation Patterns Engine
JAR	Java Archive
JAX-WS	Java API for XML - Web Services
JMX	Java Management Extension
JSON	JavaScript Object Notation
JSP	Java Server Pages
KEA	Keyphrase Extraction Algorithm
LISL	Learner Interactions Scripting Language
LMS	Learning Management System
LPGL	Lesser General Public License
LSA	Latent Semantic Analysis

MSSQL	Microsoft Structured Query Language
MUPPLE	Mash-UP Personal Learning Environment
NLP	Natural Language Processing
ODBC	Open Database Connectivity
OpenACS	Open Architecture Community System
PDF	Portable Document Format
PEAR	Processing Engine Archive
PHP	PHP: Hypertext Preprocessor
PLE	Personal Learning Environment
RAM	Random Access Memory
RDBMS	Relational Database Management System
REST	Representational State Transfer
RFC	Request For Comments
RPC	Remote Procedure Call
RSS	Really Simple Syndication
RTF	Rich Text Format
SAML	Security Assertion Markup Language
SAN	Storage Area Network
SDK	Software Development Kit
SGML	Standard Generalised Markup Language
SNA	Social Network Analysis
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVN	Subversion
TAE	Text Analysis Engine
TB	Terabyte
TCL	Tool Command Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TM	TextMining
UIMA	Unstructured Information Management Architecture
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
VNS	Vince Naming Service
WDDX	Web Distributed Data eXchange
WS-BPEL	Web Services - Business Process Execution Language
WSRP	Web Services for Remote Portlets
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation
YAWL	Yet Another Workflow Language