The ultimate objective of CLARIN is to create a European federation of existing digital repositories that include language-based data, to provide uniform access to the data, wherever it is, and to provide existing language and speech technology tools as web services to retrieve, manipulate, enhance, explore and exploit the data. The primary target audience is researchers in the humanities and social sciences and the aim is to cover all languages relevant for the user community. The objective of the current CLARIN Preparatory Phase Project (2008-2010) is to lay the technical, linguistic and organisational foundations, to provide and validate specifications for all aspects of the infrastructure (including standards, usage, IPR) and to secure sustainable support from the funding bodies in the (now 23) participating countries for the subsequent construction and exploitation phases beyond 2010.
Help Desk and Registry Prototype Report

CLARIN-2008

EC FP7 project no. 212230

Deliverable: D6C-3.1
Responsible: Dan Cristea

Contributing Partners: UAIC, DFKI

© all rights reserved by UAIC on behalf of CLARIN
Scope of the Document

This document presents the help-desk prototype report. The help desk represents the most dynamic channel of communication with CLARINs community, which is always in need of support for solving various problems. It is a vital component for attracting users and achieving CLARINs goals. We outline the steps taken to come up with a final prototype. In the beginning of our work a more research-rooted approach has been proposed. Finally, responding to comments received from the Executive Board, another version of the prototype described a solution immediately configurable on the ground of a ready-made open source framework.
Common Language Resources and Technology Infrastructure

Contents

Contents .............................................................................................................................. 5

1. Configuration of needs .................................................................................................. 6

2. The ALPE structure ..................................................................................................... 7
   2.1. The graph augmented with processing power ......................................................... 7
   2.2. Navigation inside the augmented graph ...................................................................... 8
       2.2.1. Flows ................................................................................................................ 9
       2.2.2. Operations in the augmented graph ..................................................................... 9
   2.3. Features .................................................................................................................. 10
       2.3.1. Multilinguality ....................................................................................................... 10
       2.3.2. Distributivity ........................................................................................................ 11
       2.3.3. Versioning of language resources ......................................................................... 11
       2.3.4. Manual versus automatic annotation .................................................................... 12
       2.3.5. IPR and pricing issues .......................................................................................... 12
       2.3.6. How can the model accommodate the diversity of annotation variants? .............. 12
   2.4. Design cues for a first-stage help-desk on CLARIN resources .................................. 13
       2.4.1. The BLARK zone .................................................................................................. 13
       2.4.2. The interface put to work ..................................................................................... 18
   2.5. Conclusions ............................................................................................................ 19

3. The registry prototype .................................................................................................. 19

4. The help desk prototype .............................................................................................. 21
   4.1. The end-user interface panel ..................................................................................... 21
   4.2. The expert staff interface panel ................................................................................ 23
   4.3. The admin interface panel ......................................................................................... 24
   4.4. Conclusions ............................................................................................................. 25

5. Extending the Help Desk ............................................................................................. 26

6. Conclusions ................................................................................................................ 29

References ....................................................................................................................... 30
1. Configuration of needs

The model we propose in this paper describes an environment able to allow a user to get information on possible configuration of NLP processing architectures within a CLARIN setting. A user, possessing only knowledge of XML formats of texts, can provide the system with a sample input and a sample output which he wants to obtain. The system, then, can advice the user about the processes that are required to obtain such an output, given a file containing information as that presented in the input. Such a system could point the user to the exact processing chain which is required and which can be performed by means of CLARIN’s tools.

The service should be accessed through a section of the CLARIN web site. Apart from interactive access dedicated to problem solving, this area should also contain a knowledge base, organized in the form of a database of frequently asked questions on relevant topics. Accessing the interactive entry gate, the user should be able to:

- browse through a predefined list of cases in search of one which could be similar to her. If found, then see the solution offered to that case and interact with a specialist or an automatic service in order to find out if a possibility of adapting the solution to her particular needs exists;
- assemble a solution by her own, out of components offered by CLARIN, if the input-output specifications are clearly defined;
- find information regarding standards, the input-output connecting requirements for her/his data or modules, and the possibilities of conversions that CLARIN can offer in order to make her/his files/processing components compatible;
- ideally, offer the researcher the possibility to interact with the system using natural language statements/commands, and obtain either a complete solution or have the problems encountered displayed as well as guides on how to solve them; if an automatic solution cannot be given, the system will point the researcher towards a human expert, depending on what were the problems the system encountered;
- obtain information on places, people, resources and tools, their owners, IPRs, prices, possibility of integration of modules into larger architectures, ready-made solutions, language coverage, etc.
2. The ALPE structure

In the following, we present a construction defined around a dynamic hierarchy of XML annotation schemas. The hierarchy (a directed acyclic graph – DAG) is made of nodes (representing schemas) and oriented edges (representing subsumption relations). It is dynamic because, as a result of multiple interactions, it can grow from an initial trivial shape including just one empty annotation schema up to a huge graph accommodation a diversity of annotation and processing needs. By ‘annotation’ in this context we understand XML additions to an original text, which make unambiguous linguistic phenomena. Annotations can be used as intermediate formats in a processing chain, and the more the text supports processing, the heavier could be its XML supplements.

Nodes of the graph
A node of the graph is an XML annotation schema.

Edges of the graph
We say that a node A subsumes a node B in the hierarchy (therefore B is a descendent of A) if and only if:
- any tag-name of A is also in B;
- any attribute in the list of attributes of a tag-name in A is also in the list of attributes of the same tag-name of B;
- any restriction which holds in A also holds in B (for a formal definition of restrictions and adequate examples we refer to [1]);

As such, a hierarchical relation between a node A and one descendent B describes B as an annotation standard which is more informative than A and/or defines more semantic constrains. In general, either B has at least one tag-name which is not in A, and/or there is at least one tag-name in B such that at least one attribute in its list of attributes is not in the list of attributes of the homonymous tag-name in A, and/or there is at least one constraint which holds in B but which is not prescribed by A. The subsumption relation is transitive, reflexive and asymmetrical.

If a node A subsumes a node B in the hierarchy, there is a directed edge in the graph from A to B. We will agree to use the term path in this DAG with its meaning from the support graph (the graph derived when directions of edges are removed), i.e. a path from the node A to the node B in the graph is the sequence of adjacent edges, irrespective of their orientation, which links nodes A and B. As will be seen later, the way this graph is being built triggers its property of being fully connected. This means that, if edges are seen undirected, there is always at least one path linking any node to any of the other nodes.

2.1. The graph augmented with processing power

Modern software engineering design uses interchangeable modules, which are interconnected in complex processing architectures. In NLP, this approach has proven advantages with respect to reusability, and language and application independence. In such a view, each module has inputs and outputs, but also resources which it access for configuring purposes. For the beginning, we will prefer to neglect the modules’ resources, meaning that the interface of a module is given by a pair
input-output. In order for the modules to be truly interconnectable, each of the module’s inputs and outputs must observe the constraints of certain annotation schemas. GATE [3, 4], a versatile environment for building and deploying NLP software and resources, allows for the integration of a large amount of built-ins in new processing pipelines that can process single documents or corpora. In order to assemble a pipeline, the user is instructed to select the modules needed as parts of the processing chain, in the correct processing order, and to instantiate their parameters. When all these are done, the configured chain of processes may be put to work on an input file, with the result of obtaining an output file, XML annotated.

When we place processes on the edges of the graph, the hierarchy of annotation schemas becomes a graph of interconnecting modules. More precisely, if a node $A$ subsumes a node $B$ (see Figure 1), there should be a process which takes as input a file observing the restrictions imposed by the node $A$ and produces as output a file observing the restrictions imposed by the node $B$.

![Figure 1: Equivalence between the subsumption relation and a processing step](image)

Let’s note that the directionality of a process, as attached to an edge of the graph, is that of the subsumption relation. So, if node $A$ subsumes node $B$, then the hierarchical link is from $A$ to $B$ (from the parent to the descendant). In our figures this will be marked by oriented edges (arrows). We will call a graph (or hierarchy) of annotation schemas on which processing modules have been marked on edges as being augmented with processing power (or simply, augmented). The null processor, marked $\emptyset$, is a module that leaves an input file unmodified.

### 2.2. Navigation inside the augmented graph

We claim that the graph augmented with processing power allows for the automatic identification of processing steps for configuring NLP applications. This behaviour is a result of a navigation process within the augmented graph. Any resulted process is a combination of serial processing with merges. The difference from GATE, which allows only pipeline processing in which the whole output of the preceding processor is given as input to the next processor, is that in our model the required processing may result in a combination of branching pipelines.

Once the computation of steps is done using the augmented hierarchy, then the computed process can be applied on an input file, eventually producing an output file. These files comply with the restrictions encoded by a start node and, respectively, a destination node of the hierarchy.

A processing task is defined by a pair of annotation schemas, start and destination. Transposed on the processing graph, provided the two schemas are represented as nodes in the graph, and since the graph is fully connected, there should always be at least one path connecting these two nodes. The paths found are made up of oriented edges, and, as we will see, it is important if the orientation of the edges is the same as that of the path or not.
2.2.1. Flows

As will be described below, three operations can be associated with the computed paths. Due to these operations, from the otherwise static set of alternative paths linking a start node to a destination node, a set of alternative processing paths or a flow is determined, which corresponds to a dynamically configured processing architecture. There are two ways to look at flows as processes: applied to nodes of the graph and applied to files. A flow transforms an input (node or file) by adding and/or deleting some mark-up elements, attributes or values, seen as annotation restrictions in a node (schema) and as actual mark-up in a file. The term “flow” comes easily if we imagine that the information actually “flows” through the edges of the graph, while also producing changes in the input files. Different examples of flows, linking start nodes with destination nodes, are sketched in Figure 2 in thin, interrupted arrows.

More precisely, a flow must be seen as summing-up sequences of processing steps. We will denote by $f(x)$ the flow applied to the input node or file $x$. So, $y=f(x)$ means either that the destination node $y$ is obtained by applying the flow $f$ to the start node $x$, or that the output file $y$ results by the application of the flow $f$ to the input file $x$. All the three navigation operations which will be defined below produce flows. Trivially, an empty flow, denoted by $\emptyset$, leaves the input unmodified. So, $f(\emptyset)(x)=x$, for any node or file $x$. It can be proved that the way in which we define the computation of flows will make that exactly one flow $f$ exists between a start node $A$ and a destination node $B$ in a graph, such that $B=f(A)$. Flows can be combined, more exactly, it is possible to have $B=f_1(f_2(A))$. This notation inspires the generalisation of a flow as applying to flows instead of nodes or files. Indeed, if a node $B$ is placed along a path from $A$ to $C$, we may say that the flow that transforms $A$ onto $B$ combines with the one that transforms $B$ onto $C$ to produce the flow that transforms $A$ onto $C$. As such, we may see the input of the second flow as being the first flow, instead of the intermediary node (or file).

2.2.2. Operations in the augmented graph

Two hierarchy building operations (initialize-graph and classify) and three navigation operations within the hierarchy (simplify, pipeline and merge) are defined below.

The initialize-hierarchy operation receives no input and outputs a trivial hierarchy formed by a ROOT node (representing the empty annotation schema).

Once the graph is initialised, its nodes and edges are contributed by classifying documents in the graph. The classify operation takes an existing hierarchy and a document and classifies the document with respect to the hierarchy (the classification algorithm is described in [1]). It will output a pair made by the updated hierarchy and a location of a schema (node) inside the hierarchy – the node in which the schema supporting the input document has been classified. If the input document fully complies with an existing schema within the hierarchy given in input, the hierarchy remains unchanged and the output indicates this found schema; otherwise a new annotation schema is inserted within the hierarchy, and the output is the pair consisting of the upgraded hierarchy and the location of the attached node within this hierarchy. As such, building of a hierarchy can be done two ways: ad-hoc, by manually declaring it, when there is sufficient a-priori knowledge over a full range of corpus annotations, already existent or to-be-created; or corpus-driven, by an initialize-hierarchy command followed by any number of classify commands, when a range of annotated documents are used to populate a hierarchy. One should note that in this case it is not compulsory that all annotated documents from which the hierarchy is triggered represent annotation-variants of the same initial (hub) raw text. When annotation conventions (names of elements, attributes, and values) are consistent within the collection of documents, different hub documents can be used to incrementally build a hierarchy of annotation schemas.
We say that a node $A$ is simplified to $B$, and we write $B=S^B(A)$, if $A$ and $B$ are both placed on a path from the start to the destination node, in this order, and $B$ subsumes $A$. When a file corresponding to the schema $A$ is simplified to $B$, it will lose all annotations except those imposed by the schema $B$. In Figure 2a, we have $B=S^B(A)$, and on Figure 2b it holds that $C=S^C(A)$ and $E=S^E(A)$. In accordance with our discussion on flows above, we may look at $S^B$ as a flow, which allows us to apply it to a node, to a file or to another flow.

If an edge linking a node $A$ to a node $B$ is marked with a process $p$, we say that $A$ pipelines to $B$ by $p$, which may be written as $B = A \xrightarrow{p}$. Equally, when a file corresponding to the schema $A$ is pipelined to $B$ by $p$, it will be transformed by the process $p$ onto a file that corresponds to the restrictions imposed by the schema $B$. In Figure 2b it holds that $B = C \xrightarrow{b}$ and $B = E \xrightarrow{a \xrightarrow{c}}$.

The merge operation can be defined in nodes pointed by more than one edge on the hierarchical graph. It is not unusual that the edges emerging onto the same node are labelled by empty processors. The merge operation applied to files corresponding to parent nodes combine the different annotations contributed by these nodes onto one single file corresponding to the schema of the emerging node. If $f_1, \ldots, f_n$ are flows entering a node $A$, we say that flows $f_1, \ldots, f_n$ merge into $A$ and we write $A = f_1 | \ldots | f_n$. For instance, in Figure 2b we have $B = C \xrightarrow{b} | E \xrightarrow{a \xrightarrow{c}}$. Merging a flow $f$ with the empty flow $f^\varnothing$ leaves $f$ unchanged.

With these definitions, for the graph of Figure 2b it holds that $B = S^C(A) \xrightarrow{b} S^E(A) \xrightarrow{a \xrightarrow{c}}$, and for the graph of Figure 2c it holds that $B = (A \xrightarrow{c} | S^E(A) \xrightarrow{a \xrightarrow{d}} e) | S^E(A) \xrightarrow{b \xrightarrow{f}} g$.

### 2.3. Features

#### 2.3.1. Multilinguality

Usually the adaptation of a module to process a certain natural language is given by the specific set of resources it accesses. For instance, a POS-tagger runs the same algorithms on different sets of language models in order to tag documents for POS in different languages. To take another example, a shallow parser applies a set of regular expressions, which are language dependent, in order to identify chunks. In both cases the processing modules are language independent and only the specific language model or the specific set of rules make them applicable to the language $L1$ or $L2$. 

---

**Figure 2**: Examples of processes in the graph (flows): thin, black, interrupted arrows represent flows, superposed over thick, grey, continuous arrows, representing subsumption relations.
To realise multilinguality within the proposed model means to map the edges of the augmented graph on a collection of repositories of configuring resources (language models, sets of grammar rules, etc.) which are specific to different languages. This can be achieved if the edges of the graph are indexed with indices corresponding to languages. This way, to each particular language an instance of the graph can be generated, in which each edge keeps one and the same index – the one corresponding to that particular language. This means that all processors of that particular language should access the configuring resources, specific to that language, in order for the hierarchy to work properly. For instance, in the graph instance of language \( L_x \), the edge corresponding to a POS-tagger has as index \( L_x \), meaning that it accesses a configuring resource file that is specific to language \( L_x \) – the \( L_x \) language model (Figure 3).

![Figure 3: Processes along edges are language indexed](image)

It is a fact that different languages have different sets of processing tools developed, English being perhaps the richer, presently. Ideally, the lack of a tool in a specific language should be put on to the lack of the corresponding configuring resource, once a language independent processing module has been created for that task. It is also the case that differences exist in processing chains among languages. For instance one language could have a combined POS-tagger and lemmatiser while another one realises these operations independently, pipelining a POS-tagger with a lemmatisation module. These differences are reflected in particular instances of sections of the graph, which although reproduce the same set of nodes, do not allow but for certain edges linking them. The missing edges do not allow pipelining operations, but are suited for simplification operations.

### 2.3.2. Distributivity

Edges, as recorders of processors, can be seen as Web services, therefore can be physically supported by servers anywhere on the virtual space. Similarly, documents (the files attached to nodes) could be physically located remotely one of the others. In a sense, we can say that the whole augmented graph of annotation schemas is distributed over the Web. However, as the unique accessing gate, the CLARIN portal holds a representation of the entire graph, on which classification and navigation operations can be performed. By repeated classification accesses, issued by users, the graph grows. Also navigation accesses are initiated by users and run on the portal. They leave the graph constant while returning flows. The computed flows are to be performed mainly remotely from the portal, by activating chains of processors (Web servers) which are not all located on the same machine, but which are pointed to by edges of the graph.

### 2.3.3. Versioning of language resources

To one document corresponds a multitude of possible annotations, one for each node of the hierarchy. It is possible that some of these resources are created by human annotators (gold standards), while others are created automatically. Even more, the same hub document could have versions corresponding to the same node in the hierarchy, which are created both ways. The versioning problem could be accommodated in the model through an indexing mechanism similar to
language indexing of edges, by allowing the attachment of different versions of the same document to nodes of the hierarchy. All nodes of an $L_x$ indexed version of the hierarchy, which correspond to the same hub document and the same version, are indexed with the same index.

2.3.4. Manual versus automatic annotation

While automatic annotation is supported by the graph, how can manual annotation be accommodated by the approach?

Usually, in order to train processing modules in NLP, developers use manually annotated corpora. To create such corpora, they make use of annotation tools configured to help placing XML elements over a text, and to decorate them with attributes and values. As such, if annotation tools do, although in a different way, the same jobs which can be performed by processing modules, it is most convenient to associate them with edges in the graph in the same way in which processing modules are associated with these edges.

Meanwhile, it is clear that manual annotation cannot be chained in complex processing architectures in the same way in which automatic annotation can. In order to differentiate between automatic and manual processes, as encumbered by pairs of schemas observing the subsumption relation, it results that edges should have facets, for instance AUT and MAN. Under the AUT facet of a POS-tagging edge, for instance, the automatic POS-tagger should be placed, while under the MAN facet – the POS-tagging annotation tool should be placed.

The configuration files of these tools can usually be separated from the tools themselves. We can say that the corresponding configuration files particularise the annotation tools, which label edges of the graph, in the same way in which language specific resources particularise processing modules.

2.3.5. IPR and pricing issues

IPR can be attached to documents as well as modules as access rights. Only a user whose profile corresponds to the IPR profile of a file/processor can have access to that file/service. As a result, while computation of processing chains within the hierarchy is open to anybody, the actual access to the dynamically computed architectures could be banned to users which do not correspond to certain IPR profiles of some component modules.

More than that, some price policies can be easily implemented within the model. For instance, one can imagine that the computation of a flow results also in a computation of a price, depending on particular fees the chained Web servers charge for their services.

Out of this, it is also imaginable the graph as including more than one edge between the same two nodes in the hierarchy. This can happen when different modules performing the same task are reported by different contributors. When these modules charge prices for their services, it is foreseeable also an optimisation calculus, over the set of flows that can be computed for a transformation, with respect to the overall price.

2.3.6. How can the model accommodate the diversity of annotation variants?

It is a fact that nowadays a huge diversity of annotation variants circulate and are being used in diverse research communities. It is far from us to belief that a Procrustes' Bed policy could ever be imposed in the CL or NLP community, that would aim for a strict adoption of standards in the annotated resources. On the other hand, it is also true that efforts towards standardisation are continually being made (see the TEI, XCES, ISLE, etc initiatives). Moreover, Semantic Web, with its tremendous need for interconnection and integration of resources and applications on communicating environments, boosts vividly the appeal for standardisation. It is therefore
foreseeable that more and more designers will adopt recognised standards, in order to allow easy interoperability of their applications. A realistic view on the matter would bring into the focus the standards while also providing means for users to interact with the system even if they do not rigorously comply to the standards.

We have seen already that, by classification, any schema could be placed in the hierarchy. Of course, classification could increase in an uncontrolled way the number of nodes of the hierarchy. The proliferation could be caused not so much by the semantic diversity of the annotations, as by the differences in name spaces (names of tags, attributes and values). Suppose one wants to connect a new file to the hierarchy in order to exploit its processing power. What s/he has to do is to first classify the file. If the system reports the result as being a new node in the hierarchy, then its position gives also indications of its similarity/dissimilarity with the neighbouring schemas. A visual inspection of the names used can reveal, for instance, that a simple translation operation can make the new node identical to an existing one. This means that the new schema is not new for the hierarchy, although the set of conventions used, which make it different from those of the hierarchy, are imposed by the restrictions of the user’s application.

Technically, this can be achieved by temporarily creating links between the new schema classified by the hierarchy, as a new node, and its corresponding schema in the hierarchy. Processing along such a link is different than the usual behaviour associated to the edges of the graph. It describes a translation process, in which the annotation is not enriched, but rather names of tags, attributes and values are changed. Ideally, the processing abilities of the hierarchy should include also the capability to automatically discover the translation procedure. This task is not trivial since it would require that the hierarchy “understands” the intentions hidden behind the annotation, displaying an intelligent behaviour which is not easy to implement, but could make an interesting topic for further research.

### 2.4. Design cues for a first-stage help-desk on CLARIN resources

#### 2.4.1. The BLARK zone

A prototype design for the CLARIN help-desk, restricted to the BLARK (Basic Language Resource Kit), has been created. The idea is to show the user a visualisation on what is the BLARK and what type of knowledge about the text does each component of the resource kit add through its annotation (see Figure 4).
Figure 4: The BLARK hierarchy

The purpose of the online demo at [http://www.clarin.eu/vlw/blark-hierarchy.php](http://www.clarin.eu/vlw/blark-hierarchy.php) is to accommodate the user with the basic ways of processing text using natural language processing (NLP) techniques. The structure of the hierarchy follows the natural way various processing tools can be applied to a text. The user can see how several specialized tools can be chained together in order to perform more sophisticated text processing. Interact with the nodes and line handles to find out more information about the different types of annotation and processing tools.

This hierarchy presents the tools that are usually contained in the Basic Language Resource Kit (BLARK) of a language. Similar hierarchies can be built for a larger set of tools, like the Extended
Language Resource Kit (ELARK), or for other subdomains of natural language processing, like speech processing.

The graphical elements are nodes and edges, like in a graph. Nodes indicate annotations applied to text (metadata), while edges - subsumption relations. A relation links the nodes $A$ and $B$, in this order, if the notation attached to $B$ adds some more information to the one attached to $A$.

The names of the nodes suggest the type of corresponding metadata; for instance the node $Token+POS$ accommodates all the metadata variations which are used to mark token boundaries as well as POS information, also included in the TOK tag. Figure 5 shows the hidden information of a node of the hierarchy.
Figure 5: Information associated to a node of the hierarchy: the Token+POS node

Edges are drawn using three types of lines:

- normal arrows (marked with an orange circle) identify processes: a normal arrow connecting node A to node B signifies that there should be a process able to transform a file observing the metadata in A to one observing the metadata of B. For instance a text file can be transformed to one in which tokens are marked by an automatic process, usually called a tokenizer (see Figure 6);
Figure 6: Information associated to an edge of the hierarchy: The Tokeniser

- dash-dot arrows (marked with a white circle) merge of notations attached to the origin nodes into the destination node; no additional annotations are added by this type of node. For instance, to get a file in which, apart from tokens, both noun phrases (NPs) and verb phrases (VPs) are marked, no other processing is needed apart from merging two versions of the same hub document, if available: one displaying tokens and NPs, and the other - tokens and VPs;
- dashed arrows (marked with an yellow circle) identify manual annotation. The additional annotation that makes the difference between the origin and the destination node is contributed by experts through manual annotation of the input text. For instance, a text document can be annotated to NP's manually.

### 2.4.2. The interface put to work

The intended use of this type of hierarchy is to allow the visualization, at a theoretical level, of the possible annotation data and the corresponding processes that can be applied to an input text file.

This interface would also show the user which are the resources in the CLARIN repository that have a certain type of annotation and, conversely, which are the tools that add a given type of annotation to a text.

The interaction of the user with the system would consist of the following steps:

- upload or select from an online repository a text to be processed by the system; this text should abide by the encoding/annotation standards that CLARIN supports;
- infer the language and the annotations that the text might have; position the text within the BLARK hierarchy and show which are the possible processing paths;
- show a list of compatible tools contained in CLARIN repositories that could be used to process the file (these will most likely be web services that CLARIN knows about); let the user choose a path to follow;
- show information on the availability, cost and intellectual property rights (IPR) of the tools on the selected path; if the chosen tools/resources pose no problems from either a cost, availability or IPR point of view, carry out the annotation of the text; otherwise, show the user which are the problems with the tools he chose and eventually help him configure a better solution that would yield the same end result, but without any costs/availability/IPR problems. This might be done by converting the input/output of some of the tools in the processing chain;
- process the input and obtain the requested output.

The prototype of the automatic help-desk can be implemented using the ALPE (Automated Linguistic Processing Environment) framework\(^1\). It is designed to intermediate the high level design of language processing architectures. The framework originates in a previous work developed at UAIC, describing a hierarchy of XML annotation schemas as a theoretical platform on which processing workflows can be configured and launched. The main features of ALPE allow the classification of an XML file into a hierarchy of schemas, merging of two different annotations, simplification of annotations, computations of processing flows. It is intended to guide a novice through the process of assembling a processing chain for the linguistic data she possesses. Future developments are specially designed to cope with the CLARIN needs: ease of interaction, multilingualism, distributed repositories of resources and tools, diversity of notations over the text, IPR and cost issues.

The problem of converting from one tagset to another has been tackled, in the form of a project called Tagset Semantic Exchange that deals only with part of speech tags. The idea behind it is that if we have two tagsets for the same language, they represent the same grammar, even if at different granularity levels, so we should be able to find a many-to-one mapping from the fine grained tagset

\(^1\) For a recent document, see (Pistol, 2011) – note added in June 2011.
to the coarser tagset. The mapping in the other direction, while possible to make, is a one-to-many mapping, and we would need additional information to be able to decide which one of the possible candidates to choose for a given tag in the source tagset. The semantic exchange can only work when the source tag contains at least the same amount of information as the target tag.

The application allows the user to map the description of a tagset given in natural language (but read from a fixed form XML file) to a language model. The system tries to infer automatically the correct mapping, but a human validation of the mapping can also be performed after this. The system needs at least two mapped tagsets to be able to make an exchange. After the mapping is done, the mapping provided by system can be used to change the textual appearance of tags in the finer tagset to the corresponding tags in the coarser tagset.

2.5. Conclusions

In this chapter we have presented a proposal for a theoretical model that fits the CLARIN needs for a highly intelligent help-desk type of interaction. The model is based on a hierarchy of XML annotation schemas, which should configure the universe of formats populating the CLARIN resources and tools. The type of interaction intended, in principle, should accommodate a variety of skills, from advanced researchers in the NLP field to the SSH researchers, considered to be NLP-illiterate. An instantiation of the model, dealing with BLARK, is then demoed.

3. The registry prototype

In this section we present a first proposal of a registry prototype based on DFKI’s LT-World universe.

In the context of CLARIN, DFKI-LT is providing for updates and further developments of the Language Technology Information Portal, also called LT-World (see http://www.lt-world.org/). This work is also aiming at providing a platform for the “Expertise Repository System”, which is a task in WP6 (M6C-3.1).

A major achievement is the imminent release of the new version of LT-World, which is available now for the CLARIN partners as a beta-version: http://beta.lt-world.org/.

The Beta LT World environment provides public access features, so that CLARIN partners (and other experts in the field) can login and enter new data, or manipulate existing entries according to assigned access roles. A next step here will be to set up specific access definitions for the CLARIN partners and for their specific expertise.

Within the Beta LT World portal - all entries have now unique IDs, that is - every person record has a unique object ID (e.g. Prof. Hans Uszkoreit has the following LT World URI/ID: http://beta.lt-world.org/kb/players_and_teams/people/obj_60023).

Improvement of the underlying knowledge base (an ontology of the field) has been provided; so for example for the cleaning of duplicate records in the LT World knowledge, a merging tool has been built - allowing for the fusion of duplicate entries - including the redirection of incoming references.
from various LT World record types (i.e. projects, organisations). This merging tool can be extended with minor efforts, to allow for a merging of external person records (of large numbers) with existing LT World person records (keeping their unique IDs).

Additionally to those topics immediately relevant to WP6, other developments have been undertaken, that are indirectly relevant to WP6:

- the data from the ACL Software Registry (hosted at DFKI, see http://registry.dfki.de) were prepared in XML form to be used by other CLARIN members and imported into the LT World web site.
- a mapping was defined from the OLAC based schema of the Registry to the OWL based ontology of LT World.
- the import of the Registry data into LT World naturally leads to the duplication of data in similar entries; a tool was developed to support the semi-automatic merging of this data, removing the duplication. The tool was designed to be generic in nature so that it could handle the merging of further data from CLARIN in the future.
- a distributed version of the LT World data base was set up, so that it could scale without difficulty with large volumes of data about the CLARIN corpora.
- the LT World data base was exported in an RDF form defined by the OWL ontology. This will allow parts of the database to be used elsewhere in CLARIN.
- as part of the export process we began to develop an infrastructure for permanent IDs. (this point being particularly relevant for WP2 as well).
4. The help desk prototype

Beside an automated tool for helping users with technical problems, which is not mandatory, the community of CLARIN needs to be able to communicate with experts of various areas. This is a vital aspect of the knowledge infrastructure. Given the ultimate objective of CLARIN, the help desk is a very important communication channel between the end users and the Natural Language Processing (NLP) experts.

The main objectives of such an implementation are:
- to manage a large number of questions from end-users;
- to manage a large number of experts’ interactions answering questions submitted by users;
- to offer an easy to use interface for everybody (end-users and experts);
- to be easily customizable / extendable by one or more administrators;
- to be cost and time efficient.

We describe here a design for a CLARIN help desk interface based on the ticket support system. A ticket is created by the end-user when a question is submitted. All tickets are placed in a queue, and experts assign themselves to solve the tickets.

A way to classify and categorise both the tickets and the staff members is required because there is expected to be a huge number of submitted tickets per day, and also because there will be a great number of experts from various areas of expertise. The system offers the possibility to categorize the staff experts into departments, and have managers for each of these. Experts from a department can view and answer questions only from certain help topics.

The system allows the end-user to select the topic which best suits his question, but this is optional. In case the user doesn’t select a topic, the system automatically guesses the topic of the ticket, or it assigns it to an expert which can transfer it to the correct department. The help topics number could grow very fast, given the wide range of research areas in the NLP domain. This would make the help topics list a very long one and thus making it difficult for the end-user to choose a help topic. That is why the help topics can be organized in a hierarchy.

All the customizations are managed by administrators. The helpdesk is expected to evolve, to adapt and to change its structure in time, so that it best fits the end-users needs. That is why it has to be flexible and customizable with ease.

The ticketing system platform we used has three graphic interface panels: the end-user interface, the expert staff interface, and the admin interface.

4.1. The end-user interface panel

This part of the interface must be very easy to use, and very intuitive, because the end-user might not have a proper education regarding computer skills. This aspect will help covering a larger end-user target. The interaction with the interface is minimal. Only the submission of the question is done through this interface. The rest of additional messages with/from the staff is done through e-mail.
The concept of ticket is completely transparent to the user. The submit ticket interface can be seen in Figure 7.

![Submit Ticket Interface](image)

**Figure 7:** The submit ticket interface

The e-mail address, the name and the question fields are mandatory. For choosing the help topic, the user can select in the help topics tree. If he doesn’t select a leaf node, the ticket will go to a manager which will have the possibility to transfer it to the correct department. Users are advised to choose the topic carefully because otherwise, the time taken to get an answer can be greater (Figure 8). When the user selects a topic, a description is shown to the right of the tree (Figure 8). This is very helpful for the end-user to categorise the question correctly.
4.2. The expert staff interface panel

The staff interface is more complex (Figure 9). It is considered that the expert has good computer skills. This panel offers the possibility for the expert to view tickets and ask additional questions, to close a ticket, to transfer a ticket to another staff member or another department. The expert also has the possibility to create tickets or view closed tickets to find solutions in the archive to currently open tickets. The expert can post internal notes to his or others staff’s tickets and thus helping them. All the staff activity is logged and can be viewed by the department manager.
A staff member has the option to receive an e-mail, every time a new ticket is opened, or every time a user writes additional information in a ticket which he has assigned himself to solve. Also, when an expert goes on vacation, he can specify that in the interface so that emails will not be sent anymore, and his colleagues will know of his temporary absence.

4.3. The admin interface panel

The admin interface is also very important. This is the most complex one. The administrators can create departments and can invite experts to become staff members, by e-mail. They can also assign the managers of departments. Admins can create help topics, assign them to certain departments and modify the hierarchy tree (Figure 10). Help topics can have descriptions. These are configured through this interface. Admins can ban end-users by e-mail and IP preventing spam, or various types of attacks.

There are many other details that can be configured by the admin, like the automatic e-mails templates, all sorts of alerts, date and time formats, etc.

In case of any internal error concerning the functioning of the web site, the system immediately sends an e-mail to the admins.
4.4. Conclusions

The system described can be easily configured to adapt to the requirements of CLARIN users. The help desk prototype created is compact and very well structured, yet minimal.

The three interfaces it consist of offer a reliable and efficient manoeuvring, which fits very well with the CLARIN help desk needs. They ensure a safe and smooth flow of the user’s questions, from submission to solving, thus improving the average answering time.
5. Extending the Help Desk

This chapter describes several ideas and strategies aiming to further increase the efficiency of the Help Desk prototype presented. There are areas in the help desk system that can be enhanced by automatic mechanisms. The average speed of answering a user’s question should be attentively monitored because can it can make the difference between success and failure in attracting the user. Some future decisions and modifications might be required with the increase of the number of users. This paper also aims at providing strategies for detecting when such modifications will be required.

A first point of interest is to create a system in which staff members will not be required to answer the same questions for too many times. Given the fact that closed tickets are saved in a knowledge base, staff members could manage a public list of frequently asked questions, and keep it updated for users to search in, and inform themselves.

This is a simple solution which would allow storing a large quantity of frequently asked questions for indefinitely long time. Nevertheless, to browse among them implies some work on the part of the user. In order to optimise the user’s interactions, a lazy user should be considered, which doesn’t care too much that the system could contain a huge amount of available information but is only interested to find quickly what she/he is looking for. That is why, as an even better alternative, we propose a solution in which the system detects when two questions are similar.

In this model, when a user submits a question, the system detects first if similar questions have been answered before, and if yes, it returns an already known answer to the user. Only if the user is not satisfied, a new ticket will be created, which will go through the normal ticketing system flow, presented in M6C-3.2. A methodology of this type could provide solutions for some users’ issues in just a few seconds. It also prevents that the helpdesk staff members will have to deal with repetitive issues/questions, which could become cumbersome. They will be required by the system to solve only new, not asked yet, issues.
As can be seen in Figure 11, the knowledge base is managed by staff members, and it grows constantly. Even though it is represented as pairs of questions and answers, it could be easily extended to model more questions having the same answer. This would improve in time the precision of the similarity detection algorithm.

A help desk knowledge base can become a valuable resource for the staff members belonging to various departments. The inclusion of an algorithm which is able to detect similar questions can also be used to create statistics about user’s needs/questions, to discover glitches in various resources and tools provided by CLARIN and to detect ambiguous information in the documentation. Even though it is not evident in the first place, a help desk is an extraordinary source of feedback from the user. Many facts can be inferred out of the user’s questions. These facts, we believe, could be grouped into two big categories: characterising the user as belonging to a certain type/group and revealing weak points in the documentation. It is clear that certain questions say something about the users’ general experience with the NLP domain, or her/his experience with accessing the data and tools provided by CLARIN. But repeated occurrences of questions which raise the same issue are also a sign that the documentation relative to that issue is lacking and should be reconsidered. It would be a big mistake to neglect this gold mine accumulated in the knowledge base of a help desk.

Most of the time, users submit questions to a help desk only if they didn’t find an easy alternative solution to their problem. If they don’t receive an answer fast enough, they search for alternative solutions and CLARIN risks losing them as clients. Neglecting these facts about social and humanities researchers in need for resources and tools, in time, potential CLARIN users can become
disappointed and we can lose them. It is important to underline that there are competing projects and that the end-users’ experiences while accessing data and tools are very important for CLARIN.

Of course, the ideal situation is when users can find in the infrastructure what they are looking for without any help from the helpdesk. But each time they access the helpdesk, the system’s built-in strategy should result in an update of the knowledge base. This way, in time, the helpdesk will record a valuable resource which can be used to discover areas in the system (protocols of accessing data, tools, documentation, etc.), which are in need for enhancements.

Another development we suggest deals with the automatic categorization of the help topic. The helpdesk described in M6C-3.2 requires the user to select the help topic of the issue, which would allow the system to deliver the ticket to the appropriate staff department. Even though the menu for selecting a help topic is structured as a hierarchy, for helping the user to browse easier through it, it might be the case that, in time, this menu will become very complex. This might be annoying for the users which are in a hurry, browsing through a hierarchy tree in search for a help topic. That is why we propose the development of an algorithm which would automatically categorize a user’s question in a help topic.

Finally, in a heavily exploitation regime, it could become necessary that a dedicated Helpdesk Department should be organised and supported. The helpdesk system described in M6C-3.2 offers this feature. This department should not be loaded with the obligation to answer questions but, instead, be responsible with the maintenance of the helpdesk system itself. The staff of this department must be experienced users which should keep track of internal helpdesk issues, update its parameters, develop new strategies, curate the helpdesk database, and also answer questions addressed by other staff members. In the system described in M6C-3.2, internal tickets related to the functioning of the helpdesk can be created by any staff member.

There are still many unknown variables that make the helpdesk described to be only partially designed. We believe that parameters like: the number of questions per day, the number of help topics, or the number of staff members are key factors which will determine if any of the strategies presented will be absolutely necessary. We strongly believe that, if implemented, all these strategies will lead to a very smooth interaction between the inexperienced end-user and the CLARIN sophisticated infrastructure.
6. Conclusions

This document covered aspects regarding a diversity of approaches in building help desks in the CLARIN setting.

The first model (described in Section 2) deals with a sophisticated type of interaction. The main functionality is that a user is guided towards building by himself a complex NLP architecture, but other types of interactions are also possible: visualisation of intermediate XML data along a processing flow, localisation in a hierarchy of XML schema of the one supported by a user’s file, redesign and optimisation of a processing flow, etc. Moreover, issues related to availability of different modules, multilinguality, IPR and costs can be supported by the model.

Then, section 3 brings the issue of the Registry Prototype, built by DFKI.

Discussions within the CLARIN Executive Board drove us towards other type of user needs, of a more simple nature, and a complete reconsideration of viable Help Desk solutions. A whole range of help desk prototypes are described in Section 4, satisfying basic information requirements of users.

Finally, section 5, describes issues of enhancing the ticket-based models of the previous section, by adding to the architecture a knowledge base able to retain and classify questions, which could thus act as a buffer interposed between the user and the pool of experts.
References


