

# ALPE as LT4eL processing chain environment

Dan Cristea

Faculty of Computer Science, A.I.I. Cuza  
University of Iași, Romania  
Institute for Computer Science,  
Romanian Academy, Iași, Romania  
dcristea@info.uaic.ro

Ionut Cristian Pistol

Faculty of Computer Science, A.I.I. Cuza  
University of Iași, Romania  
ipistol@info.uaic.ro

Corina Forăscu

Faculty of Computer Science, A.I.I.  
Cuza University of Iași, Romania;  
Research Institute for Artificial  
Intelligence, Romanian Academy,  
Bucharest  
corinfor@info.uaic.ro

## Abstract

This paper briefly describes the concept, initial implementation and usage of the ALPE<sup>1</sup> system for natural language processing. A hierarchy connecting annotation schemas, processing tools and resources is used as working environment for the system, which can perform various complex NL processing tasks. ALPE will be used to build linguistic processing chains involving the annotation formats and tools developed in the LT4eL<sup>2</sup> project. The particularities and advantages of such an endeavor are the main topics of this paper.

## Keywords

XML annotation, processing architectures, e-learning, linguistic processing systems, multilinguality

## 1. Introduction

One of the latest developments in Computational Linguistics, and one which promises to have a significant impact for future linguistic processing systems, is the emerging of linguistic annotation meta-systems, that make use of existing processing tools and implement some sort of processing path, pipelined or otherwise.

The “wild” diversity of formats, tools and resources scares off a newcomer or less informed user who needs to configure an NLP (Natural Language Processing) architecture to solve a given task. Configuration and parametrisation of processing chains is also very time consuming due to the heavy documentation the component modules come with. The more sophisticated the task, the more likely it is that it requires complex pre-processing steps involving several other NLP systems, which have to be chosen, documented and interfaced.

The newly emerged linguistic processing metasystems make use of existing modules in building LP chains, use existing linguistic resources and allow the user to add/build new ones, and also allow the user to compare and choose

among the available modules. The two most prominent systems of this type are GATE<sup>3</sup> and IBM's UIMA<sup>4</sup>.

GATE [3,4] is a versatile environment for building and deploying NLP software and resources, allowing for the integration of a large amount of built-ins in new processing pipelines that receive as input single documents or corpora. The user can configure new architectures by selecting from a repository pool the desired modules, as parts of a chain. The configured chain of processes may be put to work on an input file and the result is an output file, XML annotated.

UIMA [7] is a new promising release of IBM Research (first freely available version – June 2007). It offers the same general functionalities as GATE, but once a processing module is integrated in UIMA it can be used in any further chains without any modifications (GATE requires wrappers to be written to allow two new modules to be connected in a chain). Also, UIMA allows the user to work with various annotation formats and perform various additional operations on annotated corpora. Since the release of UIMA, the GATE developers have made available a module that allows GATE and UIMA processing modules to be interchangeable, basically merging the “pool” of modules available.

ALPE is another approach to the task of developing an LP meta-system, offering more flexibility than existing systems. ALPE is based on the hierarchy of annotation schemas described in [1]. In this model, XML annotation schemas are nodes in a directed acyclic graph, and the hierarchical links are subsumption relations between schemas. In [2] it is described the way the graph may be augmented with processing power by marking edges linking parent nodes to daughter nodes with processors names, each realizing an elementary NL processing step. On the augmented graph, three operations are defined: simplification, pipeline and merge. A navigation algorithm is described in this hierarchy, which computes paths between a start node, corresponding to an input file, and a

<sup>1</sup> Automated Linguistic Processing Environment

<sup>2</sup> Language Technology for e-Learning, IST 027391, <http://www.lt4el.eu/>

<sup>3</sup> <http://gate.ac.uk/>

<sup>4</sup> <http://www.research.ibm.com/UIMA/>

destination node corresponding to an output file. To these computed paths relate sequences of operations, which are equivalent to architectures of serial and parallel combinations of processors. When an input file is given to a system that implements these principles, and the requirements of an output annotation are specified as the destination node, first the XML annotation schema of the input file is determined, then this schema is classified onto the hierarchy, becoming the start node, then the expression of operations corresponding to the minimum paths linking the start node to the destination node is computed (the architecture). Finally the input file can be given to this architecture, resulting in the expected output file.

Section two of this paper presents the theory behind the ALPE system, and briefly describes the current state of development. Section three describes how the ALPE system will be used in the framework of the LT4eL European project. The conclusions, as well as the further planned developments are described in section four.

## 2. ALPE

### 2.1 Linguistic metadata organised in a hierarchy

The basis of our model is a directed acyclic graph (DAG) which configures the metadata of linguistic annotation in a hierarchy of XML schemas. Nodes of the graph are distinct XML annotation schemas, while edges are hierarchical relations between schemas. Users' interactions with the graph can modify it from an initial trivial shape, which includes just one empty annotation schema, up to a huge graph accommodating a diversity of annotation needs. If there is an oriented edge linking a node  $A$  with a node  $B$  in the hierarchy (we will say also that  $B$  is a descendent of  $A$ ) then the following conditions hold simultaneously:

- 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  $B$ .

As such, a hierarchical relation between a node  $A$  and one descendent  $B$  describes  $B$  as an annotation schema which is more informative than  $A$ . 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  $A$ . We will agree to use the term *path* in this DAG with its meaning from the support graph, i.e. a path between the nodes  $A$  and  $B$  in the graph is the sequence of adjacent edges, irrespective of their orientation, which links nodes  $A$  and  $B$ . As we will see 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 two nodes.

### 2.2 The hierarchy augmented with processing power

In NLP, the needs for reusability of modules, and language and application independence impose the reuse of specific modules in configurable architectures. In order for the modules to be interconnectable, the module's inputs and outputs must observe the constraints expressed as annotation schemas.

When we place processes on the edges of the graph of linguistic metadata, the hierarchy of annotation schemas becomes a graph of interconnecting modules. More precisely, if a node  $A$  is placed above a node  $B$  in the hierarchy, there should be a process which takes as input a file observing the restrictions imposed by the schema  $A$  and produces as output a file observing the restrictions imposed by the schema  $B$ .

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**) [2]. The null process, marked  $\emptyset$ , is a module that leaves an input file unmodified.

### 2.3 Building the hierarchy

Three hierarchy building operations are used in our model: initialize-graph, classify-file and integrate-process. They are described 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 hierarchy.

The **classify-file** operation takes an existing hierarchy and a document marked with metadata observing a certain schema and classifies the schema of the document within the hierarchy. The operation results in an updated hierarchy and the location of the input schema as a node of the hierarchy. If the input document fully complies with a schema described by a node of the hierarchy, the latter remains unchanged and the output indicates this existing node; otherwise a new node, corresponding to the annotation schema of the input document, is inserted in the proper place within the hierarchy.

**Integrate-process** is an operation aiming to properly attach processes to the edges of a hierarchy of annotation schemas, mainly by labeling edges with processors, but sometimes also by adding nodes and edges and labeling the connecting edges.

An ALPE type hierarchy can either be defined completely manually, or partially manually and partially completed with operations described above, or only using the above operations. The ALPE hierarchy variant developed for the LT4eL project is fully constructed manually, as its purpose is testing the functionalities offered, and less the building procedure.

## 2.4 Operations on the augmented graph

Three main operations can be supported by the model, as follows.

If an edge linking a node *A* to a node *B* (therefore *B* being a descendant of *A*) is marked with a process *p*, we say that *A* **pipelines to B by 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*. This arises in augmenting the annotation of the input file (observing the restrictions of the schema *A*) with new information, as described by schema *B*.

For any two nodes *A* and *B* of the graph, such that *B* is a descendant of *A*, we will say that *B* **can be simplified to A**. When a file corresponding to the schema *B* is simplified to *A*, it will lose all annotations except those imposed by the schema *A*. Practically, a simplification is the opposite of a (series of) pipeline(s) operation(s).

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 pointing to the same node are labelled by empty processors. The merge operation applied to files corresponding to parent nodes combines the different annotations contributed by these nodes onto one single file corresponding to the schema of the emerging node.

With these operations, the graph augmented with processing power is useful in two ways: for goal-driven, dynamic, configuration of processing architectures and for transforming metadata attached to documents. Automatic configuration of a processing architecture is a result of a navigation process within the augmented graph between a start node and a destination node, the resulted processes being combinations of branching pipelines (serial simplifications, processing and merges). The difference with respect to GATE and UIMA, both allowing 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. This is due to the introduction of the merge operation which is able to combine two different annotations on the same file. Once the process is computed, then it can be applied on **an input file** displaying a certain metadata in order to produce **an output file** with the metadata changed as intended. These two files comply with the restrictions encoded by **the start node** and, respectively, **the destination node** of the hierarchy.

Since the graph is fully connected, there should always be at least one path connecting any two nodes. The paths found are made up of oriented edges and, depending on whether the orientation of the edges is the same as that of the path or not, we will have pipeline operations or simplification operations. The **flow of paths** between the start and the destination node configures the processing combination that transforms any file observing the

specifications of the start node (schema) onto a file observing the specifications of the destination node (schema).

Once the entry and exit points in the hierarchy have been determined and translation links have been devised, all the rest is done by the hierarchy itself augmented with the processing power in the manner described above. This way, the processing needed to arrive from the input to the output is computed by the hierarchy as sequences of serial and parallel processing steps, each of them supported in the hierarchy by means of specialised modules. Then the process itself is launched on the input file. It includes an initial translation phase, followed by a sequence of simplifications, pipelines and/or merges, as described by the computed path, and followed by a final translation, which is expected to produce the output file.

## 2.5 Features

In this section we will describe a set of features, important for environments working with linguistic resources and tools, that emerge from the proposed model. These features are important especially when considering the proposed integration of resources and tools belonging to the LT4eL project in an ALPE type hierarchy.

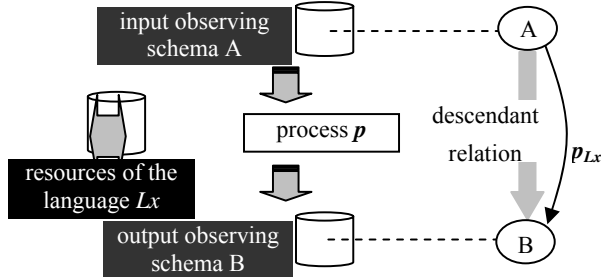
### Multilinguality

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 part-of-speeches 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.

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 labeled with processes are indexed with indices corresponding to languages. This way, to each particular language an instance of the graph can be generated, in which all edges keep 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<sub>x</sub>*, the edge corresponding to a POS-tagger has as index *L<sub>x</sub>*, meaning that it accesses a configuring resource file that is specific to language *L<sub>x</sub>* – the *L<sub>x</sub>* language model (Figure 1).

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 is available 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 lemmatizer while another one



**Figure 1.** Processes along edges are language indexed

realizes these operations independently, pipelining a POS-tagger with a lemmatization 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 inhibit pipelining operations along them, but are suited for simplification operations.

### Distributivity and access

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 in different locations than the hierarchy itself. This way, the whole augmented graph of annotation schemas could be distributed over the Web. However, as the unique accessing gate, a portal holding a representation of the entire graph, on which classification and navigation operations can be performed, must exist. By manual configuration and/or repeated classification accesses, the graph grows. Also navigation accesses are initiated by users and run on the portal. They leave the graph constant while returning the computed architectures, to be executed mainly remotely from the portal, by activating chains of processors which are not all located on the same machine, but which are pointed to by edges of the graph.

### Versioning of language resources

Each document can have multiple annotations, in correspondence with the nodes of the hierarchy. While some of the language resources may have been created by human annotators (therefore being taken as gold standards), others can be automatically created, some even using the augmented hierarchy. More than that, different versions of the same hub document may correspond to just one node in the hierarchy, mainly by being created both ways (manually and automatically). The versioning problem could be accommodated by the model through an indexing mechanism similar to the language indexing of edges, by allowing the attachment of different versions of the same

document to nodes of the hierarchy. When a computed architecture is run over an input file (corresponding to a start node), the output file (corresponding to the destination node) will be indexed identically with the input file.

### 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 descendent 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 particularize the annotation tools, which label edges of the graph, in the same way in which language specific resources particularize processing modules.

### IPR and cost issues

Intellectual property rights can be attached to documents and modules as access rights. Only a user whose profile corresponds to the IPR profile of a resource/tool can have access to it. 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 not corresponding to certain IPR profiles of certain component modules or resources they need.

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

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 fees for their services, it is foreseeable also an optimization calculus over

the set of paths that can be computed for a transformation with respect to the overall price.

### **Facing the diversity of annotation styles**

It is a fact that nowadays a huge diversity of annotation variants circulates and is being used in diverse research communities. It is far from us to believe that a Procrustes' Bed policy could ever be imposed in the CL or NLP community, that would aim for a strict adoption of standards for the annotated resources. On the other hand, it is also true that efforts towards standardization are continually being made (see the TEI, XCES, ISLE, etc. initiatives [8]). Moreover, Semantic Web, with its tremendous need for interconnection and integration of resources and applications on communicating environments, boosts vividly the appeal for standardization. It is therefore foreseeable that more and more designers will adopt recognized 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 with the standards.

We have seen already that, by classification, any schema could be placed in the hierarchy. Of course, classification could increase in an uncontrollable 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 and attributes). 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 metadata scheme of 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 neighboring 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 behavior associated to the edges of the graph and is specific to wrappers. It describes a translation process, in which the annotation is not enriched, but rather names of tags and attributes are changed. Ideally, the processing abilities of the hierarchy should include also the capability to automatically discover the wrapping procedure. This task is not trivial since it would require that the hierarchy "understands" the intentions hidden behind the annotation, displaying an intelligent behavior which is not easy to implement, but could make an interesting topic for further research.

## **2.6 ALPE vs GATE and UIMA**

Since ALPE, GATE and UIMA are systems capable of performing similar tasks, the significant differences and, most important, the advantages of ALPE over the other two are presented below.

First of all, ALPE is intended primarily to facilitate user interaction with the system, allowing the common user to access integrated resources and tools. As a standalone linguistic processing environment, the user is presented with a visual representation of a hierarchy of annotation formats and has basically three main choices: he can either add a new resource to the hierarchy, a new processing tool or create and use a processing chain by specifying start and end nodes in the hierarchy and providing the input document. In comparison, GATE offers a user interface just for creating and using processing chains, and these have to be built manually, requiring at least a well informed user. UIMA is even more oriented to the CL specialist, offering very little in terms of visual user interaction.

Every one of the three main functionalities is easier to perform using ALPE. Both UIMA and GATE require some formal description to be written for each new resource integrated into the system, but ALPE generates these formal descriptions automatically. When adding a new processing tool, ALPE has much more permissive restrictions with regards to what tool can be integrated: it basically has to be either a webservice or a command line executable under Windows or Linux. GATE allows the user to integrate just Java and Perl based tools, and this is done by writing some dedicated code. UIMA allows only Java based tools to be integrated, and only after significant implementations and changes to the original code.

When creating and using processing chains, the most significant advantage of ALPE is the automatic creation of processing chains, and the fact they can be created between any two formats in the existing hierarchy (if the required modules are available). GATE and UIMA offer relatively simple ways to create and use processing chains, but the user has to be sure the required modules exist and have compatible input/output formats. Also, ALPE deals much easier with multilinguality, as it has a module that performs language identification automatically for each input file, then selects the corresponding tools and language resources, if available. GATE and UIMA are mainly focused on English, GATE having some modules for Romanian, but the user has to make sure to select those and not the English ones when building a processing chain for a Romanian document.

Let us consider a simple use-case: the user has two processing tools he wants to use on the same input file and merge the results in an output file. Using ALPE he just has to use the available functionality to integrate the two tools in any hierarchy (even if all the annotation formats involved are not currently available: new nodes will be created automatically), then input the file and specify the required output format (node). Using GATE, the user has to

implement the integration of the tools to make them available to the processing chain building interface, to build and run two processing chains, one for each tool, then merge the results manually (GATE does not allow parallel processing and merging of annotations). UIMA performs this task basically the same as GATE, requiring even more implementation when integrating the new tools, but can perform annotations merging.

### 3. ALPE and LT4eL

The model presented in the previous sections is partially implemented and will be used (in an intermediate version) in the framework of the LT4eL European FP6 project [5]. As the main objective of the project is to provide functionalities based on language technologies and to integrate semantic knowledge in Learning Management Systems (LMS), the first step was to create an environment for collecting and (semi) automatic exploitation of language resources and tools. For the 9 languages involved (Bulgarian, Czech, Dutch, English, German, Maltese, Polish, Portuguese and Romanian), a multilingual corpus, partially parallel, of almost 9 million words was collected, annotated and uploaded on the project's portal<sup>5</sup>. There are about 30 linguistic tools on the portal, corresponding to various processing steps, hence to edges of the hierarchy of annotation schemas. The following sub-sections will briefly describe the LT4eL formats and tools which will be involved in the adaptation of ALPE for LT4eL.

#### 3.1 The resources in the hierarchy of schemas

The linguistic resources - called learning objects (LO) - were first collected as documents corresponding to formats on the first layer of the ALPE LT4eL hierarchy (see Fig. 2), according to their language, format (*doc*, *pdf*, *plain text*, *html* or *other*), domain (broadly: the use of computers in education), and IPRs. In figure 2 an ALPE hierarchy, as described in this paper, includes the *sxml* node and all others below it. After their automatic conversion to XML, using a specially created converter [6], the objects were linguistically annotated to mark (tokens, part-of-speech, lemma), hence placed on the second layer of the hierarchy – Figure 2. After another conversion to the specific format used as input for the keyword extractor developed within the project, the resources are taken to the third layer, corresponding to the annotation of keywords and definitory contexts. This specific hierarchy was used in LT4eL as a

<sup>5</sup> [http://consilr.info.uaic.ro/uploads\\_lt4el/](http://consilr.info.uaic.ro/uploads_lt4el/)

repository form for the projects LOs. Practically by clicking on the node the users have access to all files observing the specific annotation. A dedicated portal was built that includes functionalities for upload and download the projects' LOs.

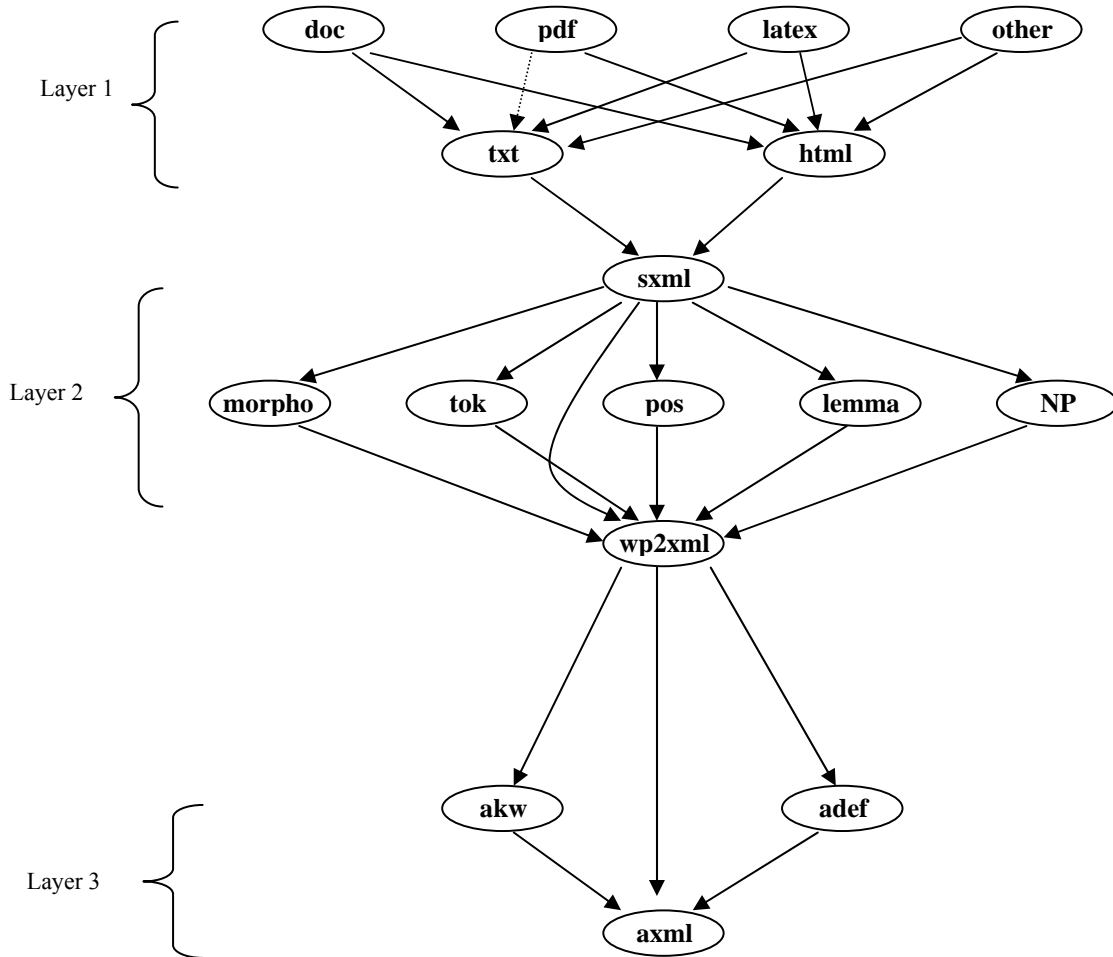
The formats (ALPE schema definitions) shown in figure 2 are:

- *doc*: Format of MS Word and OpenOffice text documents;
- *pdf*: Portable Document Format generated by any of the available software;
- *latex*: LaTeX document format;
- *html*: HTML format for web pages and documentations;
- *txt*: simple text format, without markups and viewable/editable with basic editors;
- *other*: formats other than the ones nominated above;
- *sxml*: XML format<sup>6</sup> with basic formatting information extracted from the txt and-or html formats;
- *morpho*: annotated format with added morphological information;
- *tok*: annotated format with added tokenisation;
- *pos*: annotated format with added part of speech information;
- *lemma*: annotated format with marked lemmas for words;
- *NP*: annotated format with marked Noun Phrases;
- *wp2xml*: XML format with morphological and syntactical information merged from the above formats;
- *akw*: XML with automatically generated keywords annotation;
- *adef*: XML with automatically generated definitions annotation;
- *axml*: XML format combining the automatically generated keywords and definitions annotations.

#### 3.2 The tools of LT4eL

The LT4eL corpora required extensive processing. The tools used for working with the LT4eL corpora are either existing tools, which were adapted for LT4eL purposes, or tools developed as part of the project.

<sup>6</sup> <http://ufal.mff.cuni.cz/~spousta/lt4el/html2xml/LT4ELBase.dtd>



**Figure 2.** The schema hierarchy for LT4eL

Initially, all collected documents were in various formats and they were converted to an XML format preserving some of the visual formatting (for further processing). This step involved the development of a conversion tool from an intermediate *html* or *txt* version of the original document (obtained using existing tools) to an XML format observing the LT4eL format. Basically, this tool, combined with the original (automated) conversion from the original format to the intermediate format, allows the user to input almost any kind of file in the processing hierarchy and obtain a required XML format. The conversion tool is configurable for various output formats, its source code is available, so this will become one of the core tools in the ALPE system.

For the next step, adding basic linguistic annotation to the XML corpora, existing tools were employed. Each partner language identified, adapted and used its own tools, and produced various types of annotated XML. All these formats were transformed according to a common DTD, to include linguistic information such as token markings, lemma information, POS tags, morpho-syntactical characteristics and noun phrase identifiers. Two tools were implemented to mark the keywords and definitions in the

corpora using this common XML standard for input files. The keywords and definitions were considered with respect to the LT4eL domain: teaching computer science and e-learning. All processing modules are under a continuous process of improvement. One of the final goals of the project is to fully develop these technologies with modularity and language-independence as two of the main characteristics, hence making LT4eL an ideal environment for practical testing a system such as ALPE.

### 3.3 ALPE-LT4eL

The LT4eL hierarchy represents the first significant deployment of an ALPE environment. The nature and requirements of the project impose the following characteristics of the system:

- Has to be able to handle resources/tools for 8 languages (the initial version includes tools for 3 languages, the others having to be added later).
- Has to work with files in 12 different formats. ALPE automatically identifies the file format and language.
- Has to be able to handle a wide variety of processing modules. Preliminary inquiries showed that most

modules are either Java or Perl based. Some modules are available only as executables.

- Has to be able to handle diverse processing configurations in different languages, from strictly serial to a combination between serial and parallel. In average, a processing chain involves 4 processing modules (and several of the ALPE core modules, like language identification and annotations merging).

The ALPE-LT4eL hierarchy can be constructed manually, first by defining the schema definitions graph, then adding LT4eL modules to the edges. The fact that the formats and modules involved are already available and are not subject to significant changes allowed the hierarchy to be fully build prior to its actual use. The ALPE-LT4eL hierarchy allows a user to input a document in any LT4eL format and automatically obtain any of the other formats in the hierarchy, except those above the *txt* and *html* level.

The integration of LT4eL tools and formats in an ALPE hierarchy makes processing and adding resources to the LT4eL corpus a much simpler and quicker task. All the transitions are performed automatically, as well as the detection of the input file, required resources and modules.

The manual configuration works faster only for simple processing chains. When the number of integrated modules increases, the advantage of the automated system becomes visible. Using the ALPE type of processing is more justified in complex projects involving large numbers of data formats and processing modules. Also, this type of processing can give access for non-specialist users easy to resources and tools, being able to test any LT4eL module using any file available in the many possible input formats.

Moreover, ALPE-LT4eL automatically identifies the language and checks whether a XML file (input for one of the modules) conforms to the required DTD assuring the correct execution of the processing flows.

The conceptual design of the ALPE hierarchy makes possible, in a later stage, to include new nodes (formats) and modules, as well as modules and resources for processing files in other languages.

## 4. Conclusions

In this paper we have argued for augmenting the theoretical model of an automatic configuration of NLP architectures, introduced in [1] and [2], with new features that can accommodate multilinguality, distributivity, versioning of language resources, manual versus automatic annotation, IPR and cost issues, as well as the diversity of annotation styles. For the first time, the ALPE environment has found an application field in an European project dedicated to applying linguistic processing to e-learning. Although only a part of the functionality of the ALPE framework has been exploited in this context, since the hierarchy itself was considered given and therefore not dynamically built, the integration of ALPE in the LT4eL LMS can bring

versatility for the user with respect to the file format of the input and easiness to expand the functionalities of the system for other languages.

Other envisaged deployments of ALPE hierarchies will be used in Question Answering, Textual Entailment and Anaphora Resolution systems. Eventually, ALPE will be deployed as a global NLP hierarchy dynamically built and usable on the net as a webservice.

## 5. Acknowledgments

Part of the work described in this paper was funded through the LT4eL project, STREP 027391 in FP6-2004-IST-4.

## 6. References

- [1] D. Cristea and C. Butnariu. Hierarchical XML representation for heavily annotated corpora. Proceedings of the LREC 2004 Workshop on XML-Based Richly Annotated Corpora, Lisbon, Portugal. 2004
- [2] D. Cristea, C. Forăscu, I. Pistol. Requirements-Driven Automatic Configuration of Natural Language Applications. B. Sharp (Ed.): Proceedings of the 3rd International Workshop on Natural Language Understanding and Cognitive Science - NLUCS 2006, in conjunction with ICEIS 2006, Cyprus. INSTICC Press, Portugal. 2006
- [3] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In Proceedings of the 40th Anniversary Meeting of the ACL (ACL'02), US. 2002
- [4] H. Cunningham, V. Tablan, K. Bontcheva, M. Dimitrov. Language engineering tools for collaborative corpus annotation. Proceedings of Corpus Linguistics 2003, Lancaster, UK. 2003
- [5] P. Monachesi, L. Lemnitzer, K. Simov. Language Technology for eLearning. Proceedings of EC-TEL 2006, in Innovative Approaches for Learning and Knowledge Sharing, LNCS 0302-9743, pp. 667-672. 2006
- [6] I. Pistol, D. Trandabăț, A. Iftene, D. Cristea, C. Forăscu. Processing Romanian linguistic resources in the LT4eL project (in Romanian). Proceedings of the Workshop Linguistic Resources and Tools for Processing Romanian Language, C. Forăscu, D. Tufiş, D. Cristea (eds.). University Al.I. Cuza Publishing House. 2006
- [7] D. Ferrucci and A. Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering 10, No. 3-4, 327-348. 2004
- [8] N. Ide, L. Romary, E. Clergerie. International standard for a linguistic annotation framework. In Proceedings of the HLT-NAACL'03 Workshop on the Software Engineering and Architecture of Language Technology, Canada. 2003