LREC 2016 Workshop

Translation Evaluation: From Fragmented Tools and Data Sets to an Integrated Ecosystem

PROCEEDINGS

Edited by

Georg Rehm, Aljoscha Burchardt, Ondrej Bojar, Christian Dugast,
Marcello Federico, Josef van Genabith, Barry Haddow, Jan Hajic,
Kim Harris, Philipp Koehn, Matteo Negri, Martin Popel,
Lucia Specia, Marco Turchi, Hans Uszkoreit

ISBN: 978-0-306-40615-7
EAN: 4 003994 155486

24 May 2016

Proceedings of the LREC 2016 Workshop
"Translation evaluation – From fragmented tools and data sets to an integrated ecosystem"

24 May 2016 – Portorož, Slovenia

Edited by Georg Rehm, Aljoscha Burchardt, Ondrej Bojar, Christian Dugast, Marcello Federico, Josef van Genabith, Barry Haddow, Jan Hajic, Kim Harris, Philipp Koehn, Matteo Negri, Martin Popel, Lucia Specia, Marco Turchi, Hans Uszkoreit

http://www.cracking-the-language-barrier.eu/mt-eval-workshop-2016/

Acknowledgments: This work has received funding from the EU's Horizon 2020 research and innovation programme through the contracts CRACKER (grant agreement no.: 645357) and QT21 (grant agreement no.: 645452).

Organising Committee

- Ondrej Bojar, Charles University in Prague, Czech Republic
- Aljoscha Burchardt, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany*
- Christian Dugast, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Marcello Federico, Fondazione Bruno Kessler (FBK), Italy
- Josef van Genabith, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Barry Haddow, University of Edinburgh, UK
- Jan Hajic, Charles University in Prague, Czech Republic
- Kim Harris, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Philipp Koehn, Johns Hopkins University, USA, and University of Edinburgh, UK
- Matteo Negri, Fondazione Bruno Kessler (FBK), Italy
- Martin Popel, Charles University in Prague, Czech Republic
- Georg Rehm, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany*
- Lucia Specia, University of Sheffield, UK
- Marco Turchi, Fondazione Bruno Kessler (FBK), Italy
- Hans Uszkoreit, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany

*: Main editors and chairs of the Organising Committee

Programme Committee

- Nora Aranberri, University of the Basque Country, Spain
- Ondrej Bojar, Charles University in Prague, Czech Republic
- Aljoscha Burchardt, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Christian Dugast, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Marcello Federico, Fondazione Bruno Kessler (FBK), Italy
- Christian Federmann, Microsoft, USA
- Rosa Gaudio, Higher Functions, Portugal
- Josef van Genabith, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Barry Haddow, University of Edinburgh, UK
- Jan Hajic, Charles University in Prague, Czech Republic
- Kim Harris, text&form, Germany
- Matthias Heyn, SDL, Belgium
- Philipp Koehn, Johns Hopkins University, USA, and University of Edinburgh, UK
- Christian Lieske, SAP, Germany
- Lena Marg, Welocalize, UK
- Katrin Marheinecke, text&form, Germany
- Matteo Negri, Fondazione Bruno Kessler (FBK), Italy
- Martin Popel, Charles University in Prague, Czech Republic
- Jörg Porsiel, Volkswagen AG, Germany
- Georg Rehm, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
- Rubén Rodriguez de la Fuente, PayPal, Spain
- Lucia Specia, University of Sheffield, UK
- Marco Turchi, Fondazione Bruno Kessler (FBK), Italy
- Hans Uszkoreit, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany

Preface

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam maximus euismod pharetra. Suspendisse tempor arcu urna, non laoreet ipsum maximus vitae. Duis pulvinar elit urna, a venenatis odio sodales nec. Suspendisse vel tortor lacus. Nullam venenatis ex neque. Integer id elit nec leo varius sollicitudin a at purus. Mauris placerat eros eget mauris commodo faucibus. Mauris tincidunt ultrices sodales. Ut at lorem in diam malesuada condimentum. Cras hendrerit at ligula sit amet finibus. Pellentesque pulvinar tincidunt massa, a vehicula sem aliquet eu. Donec ultricies molestie neque quis faucibus. Mauris suscipit nisl purus. Maecenas in convallis libero, eget aliquam ipsum. Suspendisse quis accumsan erat, sit amet pharetra purus. Quisque id dapibus diam.

Cras et enim et tellus rutrum egestas. Proin rutrum dolor eget mauris semper tempor. Praesent sit amet scelerisque metus. Vestibulum elementum neque non euismod iaculis. Pellentesque id pretium quam. Suspendisse vitae magna at diam luctus laoreet. Sed volutpat viverra justo, a efficitur ante ultrices non. Phasellus tincidunt urna quis pellentesque hendrerit. Morbi in dictum lorem, id iaculis nunc. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aenean lobortis malesuada ligula, a blandit leo dignissim sit amet. Maecenas tristique suscipit feugiat. Sed et elit est.

Nunc malesuada est et elit tristique, ac finibus sem auctor. Maecenas sed odio sodales, finibus neque sed, ultricies nunc. Nullam condimentum condimentum ornare. Mauris pharetra urna lacus, eget faucibus libero hendrerit sed. Proin id tempor est. Vivamus ac ornare risus. Integer quam ligula, blandit vitae nisi nec, sagittis porta orci. In sollicitudin metus a lobortis finibus.

Nulla tincidunt lectus non nibh venenatis, eget gravida lacus dignissim. Sed erat velit, venenatis et purus at, luctus molestie est. In tempor augue vitae posuere aliquam. Aliquam blandit magna eu cursus sollicitudin. Integer quis justo erat. Pellentesque dignissim neque quis ligula dignissim semper. Aliquam posuere auctor mi sit amet condimentum. Proin maximus libero id ultrices vulputate. Suspendisse gravida pellentesque odio vitae malesuada. Sed molestie enim ut porttitor condimentum. Maecenas et massa feugiat, rhoncus odio vitae, fermentum ipsum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Morbi nisl nulla, finibus non tempus sed, consectetur quis lacus. Phasellus dignissim interdum nulla, at congue sem euismod in. Donec luctus augue elementum vestibulum fringilla.

G. Rehm, A. Burchardt, O. Bojar, C. Dugast, M. Federico, J. van Genabith, B. Haddow, J. Hajic, K. Harris, P. Koehn, M. Negri, M. Popel, L. Specia, M. Turchi, H. Uszkoreit May 2016

Programme

Opening Session

09.00 – 09.10 Introduction

09.10 – 09.50 Author
Title (invited talk)

09.50 – 10.30 Author
Title

Session Title

11.00 – 11.45 Author
Title

11.45 – 12.30 Author
Title

12.30 – 12.45 Author
Title

12.45 – 13.00 Author
Title

Session Title

14.30 – 15.15 Author
Title

15.15 – 16.00 Author
Title

Closing Session

16.30 – 17.15 Author
Title

17.15 Discussion

Table of Contents

Managing Language Resources and Tools using a Hierarchy of Annotation Schemas

Dan Cristea
Faculty of Computer Science, University "Al. I. Cuza" of Iaşi, Romania
Institute for Computer Science, Romanian Academy, Iaşi, Romania
dcristea@info.uaic.ro

Ionut Cristian Pistol
Faculty of Computer Science, University "Al. I. Cuza" of Iaşi, Romania
ipistol@info.uaic.ro

Abstract

This paper describes the concept and usage of ALPE (Automated Linguistic Processing Environment) a system designed to facilitate the management and deployment of large and dynamic collections of linguistic resources and tools. ALPE can build linguistic processing chains involving the annotation formats and the tools integrated into a hierarchical structure. The particularities and advantages of integrating ALPE in a project involving the development and usage of multiple linguistic resources are the main topics of this paper.

1. Introduction

Making sure that corpora, resources and tools are reusable in different contexts than that of the originating project is one of the recent main topics of interest in the Natural Language Processing community. Re-using a resource initially developed for a specific project usually fails for one of two reasons: either the resource is not enough documented (the format is not known to the re-user), or the resource is not directly accessible (the location of the resource is not known to the re-user). Making sure a project's results are well organized and accessible ensures a better impact and a longer lasting significance, as more people will be able to use the developed resources and tools.

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

In this paper we describe ALPE, a system offering a new perspective to the task of exploiting NLP meta-systems, by helping a community of users to have an integrated look at a whole range of tools that are able to communicate on the basis of common formats.

For annotated linguistic resources several standardization efforts have been made, such as XCES [1] and TEI [2]. However, the proposed standardizations are not universally accepted, most research projects developing resources according to their own described formats. More recent developments, such as GOLD[3], propose unification methods for the various annotation formats. Due to such methods one can easily transform the name space of a corpus in order to make it compatible to her/his own targets. Several systems tried to facilitate the access to existing processing tools and to ease their usage. The more prominent ones are GATE [4] and UIMA [5]. Both systems make easier the access to a set of independently developed NLP tools which are already parts of an environment offering means to create and use processing chains intended to add linguistic metadata to an input corpus. GATE (Cunningham et al., 2002, Cunningham et al., 2003) 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 a single document or corpus. UIMA (Ferrucci and Lally, 2004) 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). Since the appearance 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, a new NLP meta-system still in development, allows a user, even with very limited programming capabilities, to automatically exploit already walked-on processing paths or to configure new ones on-the-spot, by exploiting the annotation schemas at intermediate steps. ALPE is based on the hierarchy of annotation schemas described in (Cristea and Butnariu, 2004). In this model, XML annotation schemas are nodes in a directed acyclic graph, and the hierarchical links are subsumption relations between schemas. In (Cristea et al., 2006) is described how the graph may be augmented with processing power by marking edges linking parent nodes to daughter nodes with processors, each realising an elementary NLP step.

Section two of this paper presents the theory behind the ALPE system, and section three describes the significant features of ALPE, relevant in the context of a large scale research project, employing multiple layers of annotation schemas and various tools. Section four makes a brief comparison between ALPE and the two most prominent NLP meta-systems (GATE and UIMA). The conclusions, as well as the further planned developments are described in section five.

2. The Underlying Model

2.1 Linguistic Metadata Organised in a Hierarchy

We base our model on the direct acyclic graph (DAG) described in (Cristea and Butnariu, 2002), which

[1] www.xml-ces.org/
[2] www.tei-c.org/
[3] http://www.linguistics-ontology.org/gold.html
[4] http://www.gate.ac.uk/
[5] www.research.ibm.com/UIMA/

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. By interacting with the graph, a user 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 and processing needs. If there is an oriented edge linking a node A with a node B in the hierarchy (we will say also that A subsumes B or that B is a descendant 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 descendant 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 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 the language and application independence impose the reuse of specific modules in configurable architectures. In order for the modules to be interconnectable, their inputs and outputs must observe the constraints expressed as XML schemas.

When processes are placed 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.

In (Cristea et al., 2006) a graph (or hierarchy) of annotation schemas on which processing modules have been marked on edges is called augmented with processing power (or simply, augmented). The null process, marked Ø, is a module that leaves an input file unmodified.

2.3 Building the Hierarchy

Three hierarchy building operations are introduced in (Cristea et al., 2006): initialize-graph, classify-file and integrate-process. In this section we briefly present them.

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 or manually.

The *classify-file* operation takes an existing hierarchy and a document marked with an XML metadata and classifies the schema of the document within the hierarchy. The operation results in a (possibly) 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 found 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 labelling edges with processors, but also by adding nodes and edges and labelling the connecting edges.

Apart from these basic operations that allow building a hierarchy from scratch or modifying an existing one by exploiting the annotation incorporated in files, a graphical interface allows the user to also define new nodes manually, which ALPE will place at proper places in the hierarchy automatically. But building a hierarchy can be made independent of any explicit interaction with the system by a user. It is still not unusual that an interaction results also in an augmentation of an existing hierarchy with nodes, corresponding to user's input and/or output file. Through multiple interactions, an initial minimal hierarchy which is accessed by a community of users can thus be developed.

2.4 Operations on the Augmented Graph

Three main operations can be supported by the Cristea et al. (Cristea et al., 2006) model.

If an edge linking a node *A* to a node *B* (therefore *B* being a descendant of *A*) is marked with a process *p*, it is said 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*, it is said 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). In terms of processing, 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 the described 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 connected, there should always be at least one path connecting these 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. A **flow** is a combination of paths between the start and the destination node that configures the processing which 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 processing flows (combination of paths in the graph) have been devised, all the rest is done by the hierarchy 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 specialized modules. Then the process itself is launched on the input file.

2.5 ALPE

ALPE is a system implementing the described model. Besides implementing all the previously described features, ALPE brings several additions.

The core modules

ALPE includes 11 core modules, used in any ALPE hierarchy (the hierarchy augmented with processing power, as described) but not attached to any edge. These core modules perform built-in tasks such as language

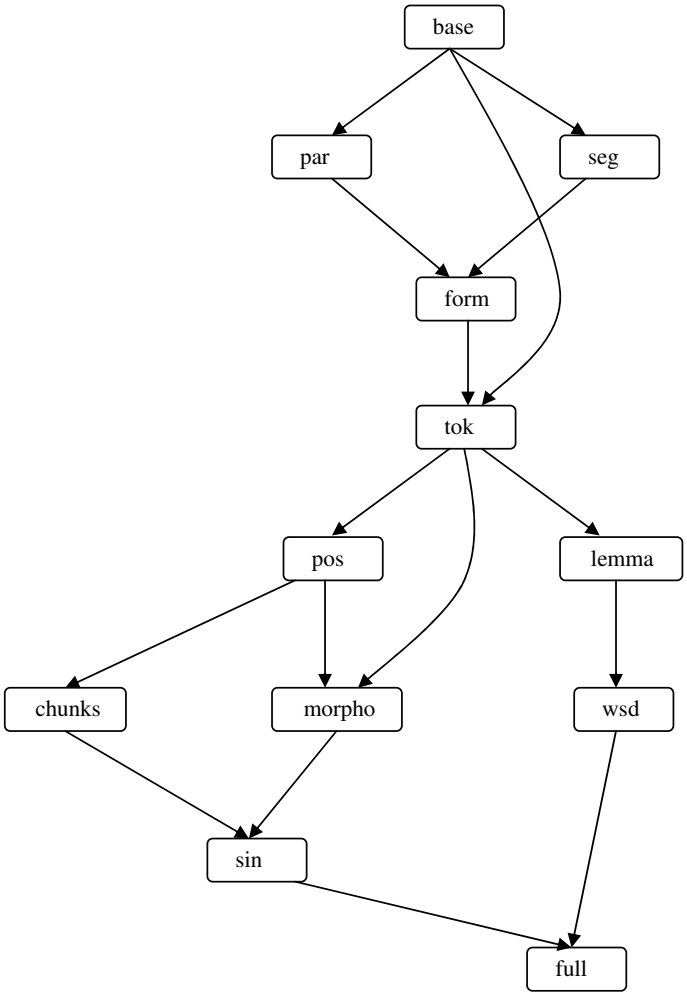

Figure 1: The ALPE core hierarchy

identification, but also implement the basic operations in the hierarchy (among others, flow computation, merging and simplifying). These core modules are used in any ALPE hierarchy and are not replaceable by user tools. They ensure that any ALPE hierarchy implements the basic behaviour, as described in this paper.

The core hierarchy

One of the main problems in developing a new NLP system is selecting a relevant and useful annotation format for the developed resources. Establishing a hierarchy of generally used XML metadata is not one of ALPE's main purposes, but having most annotated documents adhere to some common format brings obvious benefits both to the developer of new NLP software and to the user who would have an easier time finding the tools required for a particular annotation task. As base for any new ALPE hierarchy is offered a core hierarchy, with 12 annotation schemas ranging from basic XML format to a full XCES (Ide et al., 2000) linguistic annotation specification[6]. The intermediate formats are designed to conform to specific requirements for document annotation, such as tokenization, POS-tagging, NP-chunking, etc. as well as combination of these markings. Figure 1 shows the ALPE core hierarchy. All nodes are subsets of the XCES standard for annotated data, and the subsumption relation is observed between all pairs of nodes linked through an edge.

The 12 nodes in figure 1 correspond to XML annotation schemas as follows:

- *base*: subset of XCESAna including just *cesAna* tags – corresponding to a basic XML format;
- *par*: adds the *par* tag to the parent node – corresponding to an XML with marked paragraphs;
- *seg*: adds the *s* tag to the parent node – corresponding to an XML with marked sentences;
- *form*: a merge of the subsuming formats – corresponding to an XML with marked formatting (paragraphs and sentences) information;
- *tok*: adds the *tok* and *orth* tags to the parent node – corresponding to a tokenized text;
- *pos*: adds the *ctag* tag to the parent node – corresponding to a pos-tagged text;
- *lemma*: adds the *base* tag to the parent node – corresponding to a lemmatized text;
- *chunks*: adds the *chunk* and *chunklist* tags to the parent node – corresponding to a (Noun/Verb) phrase-chunked text;
- *morpho*: adds the *msd* tag to the parent node – corresponding to an XML displaying morphological metadata;
- *wsd*: adds a *wsd* tag for semantic disambiguation;
- *sin*: merges the parent nodes – corresponding to an XML displaying full syntactic information;
- *full:* merges all parent nodes.

The purpose of the core hierarchy is to offer both a starting point to any new hierarchy as well as anchors for any new linguistic annotation formats that a user would like to include. When the XML formats of the user's input and output files are not identical with schemas belonging to the hierarchy (for instance, due to differences in the tags name space or to configurations of attributes that convey in different ways the same information) then the user has to provide convertors (wrappers) able to accommodate his notations with those corresponding to nodes of the hierarchy.

The user's needs and the selection of flows

The ALPE augmented hierarchy can be used in many ways. Suppose a user wants to process an XML file from one input format to some output format. In principle, any such processing task involves a transformation by some module capable to receive the input format and to output the required final format. The ALPE philosophy details such a processing task in relation with the pair of input-output schemas by establishing the way these schemas interrelate from the point of view of the subsumption relation. Two cases can be evidenced: either the two schemas do observe a subsumption relation or not. When they do, then the node corresponding to the input file can be connected through a direct descending or ascending edge to the one corresponding to the output file. It will be descending if the output schema results from the input schema through some adds, and it will be ascending if in order to obtain the output, simplification applied to the input are required. When the two schemas are not in a subsumption relation, then there should be a node such that either both are subsumed by it, or both subsume it.

ALPE comes with a core hierarchy whose nodes act as a grid of fixed bench-marks with respect to which the locations of the input and output schemas are set out. When the pair of users' schemas matches two nodes of the core hierarchy, then processing can be drawn in terms of known (built-in) interconnected modules. When a match (modulo, as noticed above, the XML elements name space and/or differences in configurations of attributes still conveying the same information) of one or even both of user's schemas against nodes of the hierarchy is not possible, then the non-matching schemas should be seen as new nodes of the hierarchy. In this case it is the user's responsibility to locate also the processes which will be assigned to the new edges which will interconnect the new nodes onto the hierarchy.

ALPE designs a solution to the user's problem by first computing all possible chains of edges which link the input schema to the output schema and, if needed, executing them.

Each computed flow is characterized by a set of features. These features include properties such as: flow length (defined as number of processing steps involved), cost (for instance, if processing involving one or more modules presupposes financial costs), the estimated precision of execution, and the estimated time of execution. The user can then select and run the flow most suitable to his needs.

3. Features

In this section we will describe a set of features implemented in ALPE often wished for in environments working with linguistic resources and tools. We will see how these features emerge from the model described above. Many of these features are key elements of the

[6] http://www.cs.vassar.edu/XCES/dtd/xcesAna.dtd

future European linguistic infrastructure, as seen by CLARIN[7].

Multilinguality

In modern NLP, algorithms are separated from linguistic details. This way, a module designed to perform a specific task can be put to work on any language if fuelled with appropriate language resources. This is the case, for instance, with POS-taggers (see, for instance, TNT (Brants, 2000)), which are powered by specific language models (frequency of n-grams of POS tags). A syntactic parser should be powered by the grammar of a language to be effective in parsing sentences of that language. A shallow parser, which usually implements an abstract automata machinery, could recognize noun phases of one language if powered by a resource consisting of a set of regular expressions specific to that language.

To implement 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, regular expressions, etc.) which are specific to different languages. This can be achieved if the edges of the graph labelled 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 *Lx*, the edge corresponding to a POS-tagger has as index *Lx*, meaning that it accesses a configuring resource file that is specific to language *Lx* (that language model).

It is a fact that different languages have different sets of processing tools developed, English being perhaps the richer, presently. Ideally, the blame for the lack of a tool in a specific language should be put on 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 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

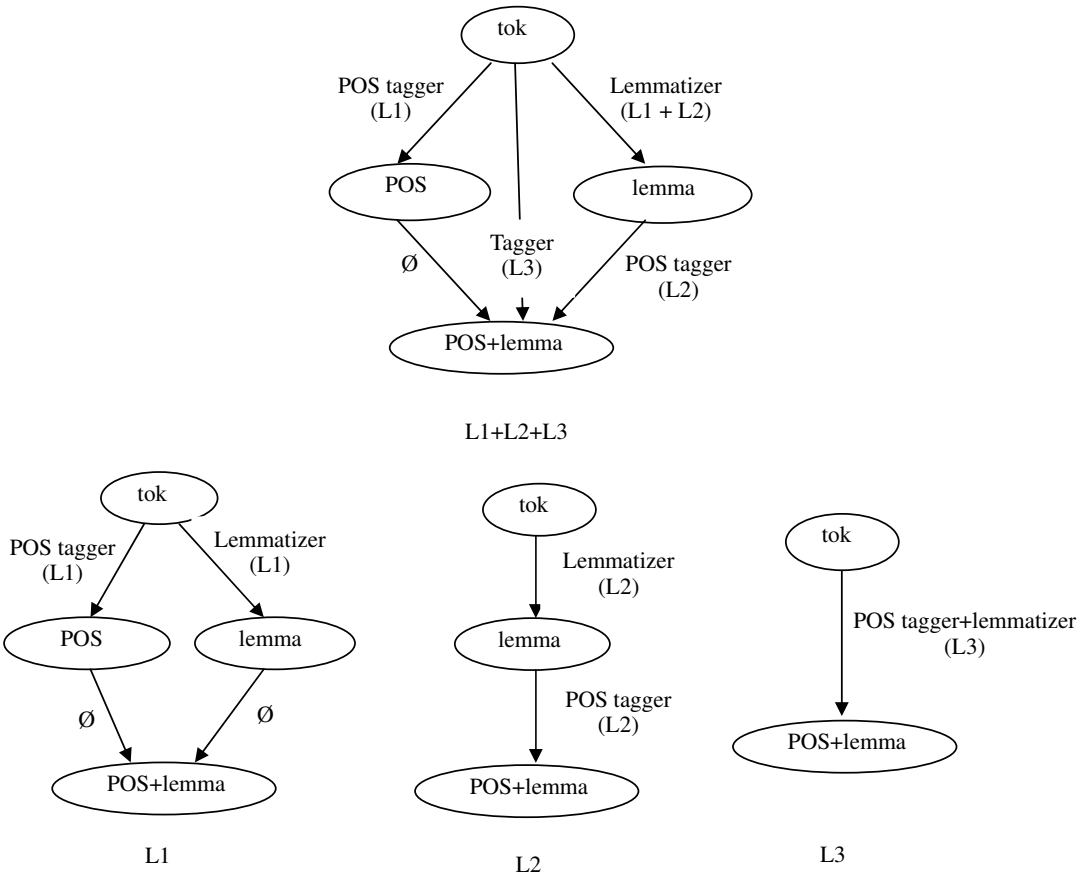

Figure 2: Computation of different flows for specific languages

[7] http://www.clarin.eu

along them, but are suited for simplification operations.
In figure 2 is given a simple example of how ALPE handles multiple languages integrated in the same hierarchy. The first hierarchy (marked as L1+L2+L3 in the figure) has four nodes (annotation schemas):

- *tok*: XML which marks lexical tokens;
- *POS*: XML marking tokens and their part-of-speech;
- *lemma*: XML marking tokens and their lemmas;
- *POS+lemma*: XML with tokens, POS and lemma information.

These four nodes correspond to simple processing stages for linguistically annotated documents. The ALPE hierarchy fragment representation (shown on the L1+L2+L3 section of Figure 2) indicates the subsuming relations between the respective nodes and the attached tools. For each tool, in parenthesis, it is indicated the languages for which the tool is available. In the sections marked L1, L2 and L3, respectively, of Figure 2 are sketched the corresponding instantiations of this sub-hierarchy for the three languages.
The user can provide an input document (XML with marked lexical tokens) and specify the required output format as being the final node (suppose *POS+lemma*). ALPE determines the language of the input document (as being L1, L2 or L3). If the input document belongs to the language L1, the computed flow will include only tools available for that language. Thus the only possible flow will use the *POS tagger* and the *Lemmatizer* tools, then merge their results into the output format. For the second language the flow will use a different *POS tagger* tool, one that requires as input a file corresponding to the *lemma* node. So the computed flow will run first the *Lemmatizer*, then the *POS tagger* on the result. For the third language, a tool is available that can directly annotate an input file in the *tok* format up to the required output.
We can look at the ALPE hierarchy as having three layers, one for each language. The three language specific hierarchies can look completely different for each language, but are still able to compute and run the same flows as the combining hierarchy. The three layers are aligned by nodes which display the same XML structure.

Manual versus automatic annotation
We have seen how automatic annotation is supported by the augmented graph. But how can manual annotation be accommodated within this 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 subsumtion 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.

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 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 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 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 fees for their services, it is foreseeable also an optimization calculus with respect to the overall price over the set of paths that can be computed for a required processing.

Facing the diversity of annotation styles
It is a fact that, today, a huge diversity of annotation variants circulates and is being used in diverse research communities. It is far from us to belief that a Procustean 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). 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).
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 and is specific to wrappers. It describes a translation process, in which the annotation is not enriched, but rather names of XML elements and attributes are changed. Ideally, the processing abilities of the hierarchy should include also the capability to automatically discover wrapping procedures. This task is not trivial since it would require that the hierarchy "understands" the intentions hidden behind the annotation, displaying, this way, some kind of semantic processing capabilities which is not easy to implement. However, recent initiatives as GOLD make us believe that significant steps forward in this direction are near us.

4. Evaluation

4.1 ALPE vs. GATE and UIMA

In this section we will compare functionalities of ALPE with those of GATE and UIMA, systems which can give very similar results with our.

First of all, ALPE is intended primarily to facilitate the user's interaction with the system, allowing for an programming non-expert to integrate 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: s/he can add a new resource to the hierarchy (for example enabling an already integrated processing module to work for another language by adding a corresponding language model), add a new processing tool (attached to an existing edge, or attached to a newly created edge) or compute and use a processing chain (providing the input file and selecting the output format). GATE offers a user interface adequate for creating and using processing chains. Chains have to be built manually and presuppose an intimate knowledge of the system. UIMA is even more oriented to the NLP professional, offering little in terms of visual user interaction. A direct comparison that would put on stage quantitative evaluations is difficult to be made for these kinds of systems. Perhaps a better prospect would be a qualitative comparison performed by a significant pool of users, providers as well as consumers of language resources and tools. In the following, we make just an estimative comparison, but a qualitative evaluation versus human performance is planned.

Every one of the three main functionalities (adding a new resource, adding a new tool, and computing and using a processing chain) is easier to perform in ALPE. Both UIMA and GATE require some formal description to be written for each new resource integrated into the system, while ALPE generates these formal descriptions automatically. When adding a new processing tool, ALPE has much more permissive restrictions with regard 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 at least Java and Perl based tools, and this is done by writing some dedicated code, a task which is however above the capabilities of some users. UIMA is even more restrictive, allowing only C++ based tools to be integrated, and only after significant implementations and changes to the original code. However, an extension allowing modified Perl, Python and TCL modules to be integrated is available.

An evident advantage of ALPE over both GATE and UIMA is that the processing chains in ALPE are automatically computed, therefore requiring no human intervention. Moreover, they can be created between any two formats defined in the hierarchy (providing the modules decorating the connecting edges are available, otherwise there are signalled as missing). ALPE deals with multilinguality, thanks to its core module that performs language identification for each input file, then selects to corresponding tools and language resources, if available. GATE and UIMA are mainly focused on English (GATE incorporating also modules dedicated to some other languages), but the user has to make sure to select the proper modules when designing a processing chain for a document in other language than English.

Let us consider the example of a use-case in which the user has two processing tools s/he wants to use on the same input file and to merge the results in an output file. Using ALPE, this user has to specify the input/output formats of the modules, then let the system integrate the tools as arches linking the corresponding nodes in the hierarchy (in the case when one of both of these formats are not currently part of the hierarchy, they will become as such), 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, then build and run two processing chains, one for each tool, then merge the results outside GATE (since it does not allow parallel processing and merging of annotations). UIMA performs this task basically in the same way as GATE, requiring even more implementation when integrating the new tools, but allows annotation merging.

4.2 Qualitative evaluation

In order to evaluate ALPE versus human computational linguistic specialists, we have developed an ALPE augmented hierarchy configured for a current research project involving documents in 9 European languages (Bulgarian, Czech, English, German, Dutch, Maltese, Polish, Portuguese and Romanian) and using a significant number of language processing tools[8]. All documents have to be annotated according to 6 main annotation formats (and 8 optional ones), resulting a significant hierarchy of standards. This hierarchy is already implemented and serves as a management and access facility for the collected documents.

At the time of writing this paper, an ALPE core hierarchy specific to the mentioned project is implemented for English and Romanian.

5. Conclusions

We think that the model we propose and its first implementation, as the ALPE system, encapsulate different organisational, standardisation and processing features which make it interesting for the goals of a project like CLARIN.

In this proposal we have been concerned with the following features of functionality, also identified as of

[8] LT4eL – an FP6 project (www.lt4el.eu)

primary importance in CLARIN[9]:

- **unique access gate and distributivity**: although distributed in different places, LR and LT could be, in the vision described in this paper, identified through a single access gate;
- **metadata policy**: primary text and speech documents should be given the possibility to be accompanied by metadata describing human and/or automatic annotation over them. The ALPE conventions allow for the metadata to have a form which make it easily removable when the primary raw documents are needed of being recuperated;
- **independence of representation**: it is clear that the XML representation adopted by ALPE allows for LR to be manipulated in such a way as to benefit of the same treatment irrespective of the particular metadata conventions;
- **quick access**: ALPE comes very close to the objective that CLARIN LR and LT be accessed instantaneously from all over Europe;
- **conversion services**: the ALPE approach incorporates features that allows easy conversion operations from and onto different representations;
- **processing services**: the ALPE portal provides processing services for enrichment and or simplification of metadata attached to LR;
- **versioning**: the portal allows manipulation of different versions of data as well as of the metadata accompanying the texts;
- **multilinguality**: the structure allows uniform treatment of documents in different languages, as well as of parallel texts;
- **IPR issues**: the structure provides means of dealing with IPR.

In this paper we have described a model of dynamical building of processing architectures based on a hierarchy of XML schemas and an implementation – called ALPE. We have argued that ALPE brings some advantages over other known systems with similar objectives, mainly coming from a plus in manoeuvrability and complete automation of the configuring tasks. It is also shown how ALPE, has brought already significant advantages in the context of a multilingual research project. In this context ALPE has automatically configured complex processing chains involving several modules and documents in different languages. The features brought by the addition of an ALPE type hierarchy into a complex project contribute significantly to acquire multilinguality, distributivity, versioning of language resources, automatic and manual annotation, management of IPR and cost issues, as well as managing diversity of annotation styles, features that the CLARIN project considers of extreme importance.

One important further development of ALPE will be a web-service allowing users to build, configure and use ALPE hierarchies on the web, either as a limited password-protected resource or a global linguistic resources collection. This type of hierarchy is able to manage multilingual resources as well as resources which require a fee to be paid before usage. Each user will be able to contribute its own tools and annotated resources, as well as using processing chains adapted to its specifications, both in terms of input and output formats and cost and performance issues.

[9] We foresee that other requirements, as, for instance, discovery of resources and tools, preservation of resources, archiving services, content discovery, distribution, authentication and authorization, could also be designed around the structure we propose.

Acknowledgments

Part of the work for the paper was supported by the ROTEL (CEEX project) AMCSIT contract no. 29/03.10.2005, the CLARIN INFRA-2007-2.2.1.2 project, and the FP6 LT4eL project.

References

T. Brants (2000): TnT: a statistical part-of-speech tagger. In Proceedings of the sixth conference on Applied Natural Language Processing, Seattle, Washington, pag: 224 – 231.

D. Cristea, C. Butnariu (2004): Hierarchical XML representation for heavily annotated corpora. In *Proceedings of the LREC 2004 Workshop on XML-Based Richly Annotated Corpora*, Lisbon, Portugal.

D. Cristea, C. Forăscu, I. Pistol. (2006): Requirements-Driven Automatic Configuration of Natural Language Applications. In Bernadette Sharp (Ed.): *Proceedings of the 3rd International Workshop on Natural Language Understanding and Cognitive Science - NLUCS 2006*, in conjunction with ICEIS 2006, Cyprus, Paphos, May 2006. INSTICC Press, Portugal. ISBN: 972-8865-50-3.

H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. (2002): 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)*. Philadelphia, US.

H. Cunningham, V. Tablan, K. Bontcheva, M. Dimitrov. (2003): Language engineering tools for collaborative corpus annotation. *Proceedings of Corpus Linguistics* 2003, Lancaster, UK.

D. Ferrucci and A. Lally. (2004): UIMA: an architectural approach to unstructured information processing in the corporate research environment, *Natural Language Engineering* 10, No. 3-4, 327-348.

N.Ide, Bonhomme P., Romary L. (2000) : XCES: An XML-based Encoding Standard for Linguistic Corpora, *Proceedings of the Second International Language Resources and Evaluation Conference*. Paris: European Language Resources Association

Sustainable operability: Keeping complex resources alive

Menzo Windhouwer[a], **Alexis Dimitriadis**[a,b]

[a]University of Amsterdam, [b]Utrecht institute of Linguistics OTS
M.A.Windhouwer@uva.nl, alexis.dimitriadis@let.uu.nl

Abstract

The data contained in a typological database are difficult or impossible to use on their own. Sustainability must include not only preservation of the data, but also of the interface designed to present them—or a reasonable substitute. The *Typological Database System* project (TDS), which originated as a way to address issues of fragmentation and interoperability of typological databases, also points the way to a model of sustainability beyond the lifetime of a database's host application.

1. Introduction: Obstacles to the sustainability of complex resources

While the sustainability of language resources such as corpora and dictionaries can be largely safeguarded by relying on documented, standard formats for their encoding, the approach does not scale well for resources with more complex internal structure, for which no general standard can be sufficient. Such complex resources have the characteristic that they require a certain software tool for their proper utilization; and that this software tool is not generic (e.g., an audio player, text editor, or linguistic annotation tool that supports the storage format of the resource), but is made specifically for the resource in question: Databases, in particular, are typically accessed through a custom-made user interface. A second, interacting problem is that much of the information needed to properly navigate and interpret such data is encoded in its user interface, not with the data itself. We consider the case of typological databases, and describe our approach to their integration and long-term sustainability.

Consider, as a concrete example, a typological database consisting of several linked tables, accessible over the internet through a web interface comprising several forms. Numerous such databases exist today, and more are being created at a rapid pace.[1] Once they are completed, such databases are subject to the usual perils afflicting electronic linguistic resources: Gradual obsolescence of their encoding formats or host software; sudden disappearance due to incompatible software updates, retirement of a "legacy" server, or as URLs change and links fail to be updated; gradual fall into unusability with the dissipation of the insider knowledge often needed to usefully operate a poorly documented resource; etc.

To render such a database sustainable, it is not enough to export its tables in some format guaranteed to be readable (e.g., tab-separated files in a Unicode encoding, or even an SQL dump in some portable format). Doing so is insufficient in two important respects:

a. The meaning of the table contents, and their interrelationships, are not explicitly given in the data tables; this is the familiar problem of documentation for a resource, but exacerbated (compared to corpora or dictionaries) by the complexity and variability of database structures, and by the relatively abstract level of linguistic description involved.

b. Even if accompanied by full documentation, a static collection of data is difficult, tedious, or even impossible to utilize without a suitable software tool. The forms and menus created by the original developers to operate a database are essential to its use, but they cannot be exported along with the data. We will term this consideration, which has not received as much explicit attention as issues of format and access, as the problem of sustainable *operability*.

To appreciate the scale of the operability problem, consider the difficulty of using a general-purpose table browser (a spreadsheet application, for example) to navigate the contents of a database consisting of several tables. Table columns (attributes) typically contain numeric values expressing different properties (whose meaning is, at best, explained in a separate document).[2] The tables are linked to each other by means of numeric keys with no intrinsic meaning. The process of navigating such data is tedious and error-prone, and likely to deter all but the most motivated researchers.

Lack of operability also has a detrimental impact on resource discovery: Summary metadata can only give an approximate indication of the utility of a resource for any particular task. A future researcher who will need to evaluate a large number of potentially useful resources will be hindered by the inability to inspect their contents without a large investment of effort.

1.1. The limits of data-only formats

The vast majority of existing typological databases are stored in relational database management systems. The

[1]Web-accessible databases include the Graz Database on Reduplication, at *http://reduplication.uni-graz.at/*; the databases of the Surrey Morphology Group, at *http://www.smg.surrey.ac.uk/*; the Typological Database of Intensifiers and Reflexives, at *http://userpage.fu-berlin.de/~gast/tdir/*; the Stress Typology Database, at *http://stresstyp.leidenuniv.nl/*; the Berlin-Utrecht Reciprocals Survey, at *http://languagelink.let.uu.nl/burs/*; etc.

[2]In proper relational design, numeric values can be indices into a separate table that matches numeric codes to a text equivalent. In practice, however, the meaning of numeric values is often embedded in the user interface; and prose documentation can be nonexistent or out of date.

relational structure itself is a sort of encoding standard, and would seem to provide a basis for standardization: While SQL implementations are too variable for database dumps in SQL format to be themselves portable, some version or extension of standard SQL could conceivably be chosen as the standard for data archiving. Even if the obstacles to unifying the many extant flavors of SQL could be overcome, however, the result would allow implementation-independent data storage but would still not render databases operable. The SQL schema of a database is insufficient in the same respects already mentioned:

First, it is an incomplete description of the database, since it does not include those parts of the database logic that are encoded in the user interface: Documentation and instructions to the user, business rules (explicit or implicit), and, in many cases, the text equivalents of values and menu options that are stored as small integers in the database. In the language of the OAIS Reference Model (ISO 14721, 2003), an SQL dump of a typological database is rarely "independently understandable."[3]

Second, general-purpose browsers for relational databases are too low-level; they allow viewing of one table at a time, but do not automatically perform appropriate joins or aggregations of records in one view—and, even with knowledge of foreign key declarations, have no way of determining which joins or aggregations are "appropriate." Simply put, the user interface of a database is underdetermined by its relational schema.

We doubt that these problems are restricted to relational databases. Similar issues doubtless arise with other complex resources developed with their own interface, and with other data models besides relational databases.

1.2. Toward a solution

The difficulty of achieving sustainable operability can be summarized as follows: Complex resources require ad hoc software that cannot be maintained over the long term; so operability can only be ensured by relying on generic software that can be maintained, and periodically replaced, in a cost-effective manner. But traditional data archiving practices do not provide enough information for generic software (or even human specialists in many cases) to reconstruct the proper structuring and presentation of the data.

It can be seen now that to fully meet the goal of sustainable operability, the archived data must first be "independently understandable." We can distinguish here between user-oriented metadata (documentation), which helps users interpret the data when it is presented, and formal, system-oriented metadata that is machine-understandable and can describe not only the encoding and relational structure (narrowly considered), but also appropriate ways of managing and presenting the data to the user.

[3]The OAIS Reference Model charges conforming archives with ensuring that archived information be "independently understandable" by its designated target community, i.e., interpretable without recourse to hard-to-access resources, including the individuals who created it. This is considered necessary for long-term data preservation. We thank an anonymous reviewer for calling this point to our attention.

What is needed, minimally, is a software platform that provides operability of typological databases with diverse structures. While no tool could probably be fully generic and at the same time achieve operability without a prohibitive amount of configuration, the problem is not intractable when restricted to one application domain at a time—in our case, to the data models applicable to typological databases. But no software platform can make up for the lack of information that is essential to managing or understanding a resource; this problem must be addressed by ensuring that the required information is collected, and is suitably utilized by the software platform in question.

Sustainable operability, in short, requires two things: sufficiently rich metadata and documentation for the data to be not only "independently understandable" by its end-users, but also for automatically determining appropriate ways of rendering it; and a software tool, or a series of software tools over a long period of time, that utilize this information to provide the actual operability.

To provide operability of an open-ended collection of resources in a practical way, there must be a way for a generic application (or several) to be used with all of them. Because the native storage formats (usually relational) are insufficient to describe typological databases to a degree that allows operability via a generic tool, we adopt a hierarchical, semi-structured data model that combines the data itself with rich documentation of database contents and of the linguistic properties being described. We will term this the *Integrated Data and Documentation Format* (IDDF). Sustained operability is then a matter of mapping resources to the IDDF format at the time of archiving, and maintaining a generic tool, or tools, that support searching and browsing over IDDF resources. This approarch accomplishes operability of the databases in the narrow sense, and also provides access to the documentation needed by the end user to properly interpret the available data.

Eventually, even the generic software will approach obsolescence due to changes in web technology, host operating systems, and the like. At that point it will need to be replaced by new IDDF-aware software with analogous functionality. The self-describing nature of IDDF documents is meant to support their migration to new access tools (or even the addition of new tools next to existing ones) without any changes to the resources themselves.

But long-term operability is more than a matter of keeping the software running. A proper solution should also support other considerations of sustainability. In particular, it should be positioned within a scenario involving data archiving and its complement, resource discovery.

The Typological Database System (TDS), described in more detail in section 2., is a working implementation of such an architecture. The TDS provides integrated access to a collection of independently developed typological databases through a single, generic web interface. Databases are imported into the system through a process that combines rich documentation of all aspects of the data with automated transformation the data itself into a common, hierarchical data space. The result is a unified data structure (the IDDF data tree) that can be searched or

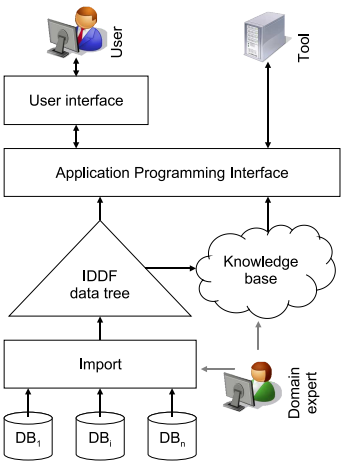

Figure 1: The TDS architecture

browsed over the web through the TDS webserver.[4]
While the process can easily be performed on each database separately, the approach has the added benefit of allowing the integration of multiple databases into a unified resource. (This is in fact the primary goal of the TDS). Arguably, integration is not essential for sustained operability of the resources; but it greatly enhances their usefulness, efficiency of utilization, and ease of resource discovery.
Data archival inadvertently exacerbates the problem of operability, because archives cannot commit to long-term hosting and maintaining a caleidoscope of diverse database applications; rather than wait for obsolescence of the software or hardware, operability threatens to be lost at the moment of uploading the static content of a resource to a digital archive. From our perspective, this can be seen as a blessing in disguise: Sustainability problems can be addressed while the original technical infrastructure is still operational, and the custodians of a resource still possess the required knowledge (either in their heads or as offline documentation).

2. The Typological Database System

The Typological Database System is a web-based service that provides integrated search access to a collection of independently developed typological databases. The system consists of a data integration module and a web server that provides access to the integrated data.[5] At the intersection of the two parts is the IDDF, a hierarchical data model that integrates data and metadata from multiple databases into a unified data space.
Figure 1 shows the TDS architecture. The primary data input to the system comes from the component databases. A domain expert creates an import schema that includes a mapping of each database into a unified hierarchy, enriched by documentation of the data and its relationship to the common TDS knowledge base. On the basis of this schema, data and documentation from multiple databases are integrated into a single hierarchical structure, the *IDDF data tree.* A separate component of the system, the TDS webserver, supports querying, browsing, and resource-discovery functions over the collected data.
The entire system is XML-based and relies on a number of (commercial) open source or freely available libraries. It is written largely in Java, XSLT, XQuery and a XML pipelining language specific to the application server 1060 NetKernel.[6]
With around a dozen databases currently in the TDS, the total number of parameters in the system is well over a thousand; hence the system follows a two-stage access model, consisting of resource discovery followed by query formulation and execution. During the resource discovery stage, users search or browse the combined metadata to discover database fields of interest. The user interface supports integrated search, display and navigation of the metadata, presenting users with the information necessary to assess both the relevance and the correct interpretation of a field. Selected fields are accumulated using a shopping basket model. In the second stage, the user constructs and executes a query on the basis of the fields in the query basket.

2.1. The integration process

The import schema is defined in a special-purpose language developed by the TDS project, the *Data Transformation Language* (DTL). [7] The TDS import engine interprets the DTL specifications, and uses an appropriate software plug-in to extract data from a copy of the original database (which can be in a variety of database formats) and transform it into the IDDF tree.
Typically, the documentation provided with a database is insufficient to make its semantics and logical structure fully explicit, and the creation of the DTL specification involves repeated interaction between the TDS domain expert and the creators of the database. The required metadata, which often lives only in the heads of the database's creators, is in this way elicited and recorded. The process is non-trivial but necessary for the sustainability of the data. Because the developers of the component databases have devoted much time and effort to collecting information in their databases, each component database represents a valuable resource; and therefore the time investment is justified.
In any event the process is reusable: Once the transformation schema has been defined, new data added to the database can be imported with minimal human intervention. In this way a database can be mapped to the IDDF before the data collection is finished and its data frozen.

[4] *http://languagelink.let.uu.nl/tds/*.

[5] The TDS is a project of the Netherlands Graduate School of Linguistics (LOT). It is supported by a grant from the Netherlands Organization for Scientific Research (NWO), and by funds from the participating universities (University of Amsterdam, Utrecht University and Leiden University). For more information on the TDS, see (Saulwick et al., 2005; Dimitriadis et al., 2005; Dimitriadis et al., 2008).

[6] *http://www.1060.org/*.

[7] The DTL is a non-procedural language that allows an IDDF schema to be specified and annotated, and the resulting data tree to be populated from the database contents. It was designed for use by linguists with no special technical background. See (Saulwick et al., 2005; Dimitriadis et al., 2008).

Only if the database schema is modified is it necessary to modify the transformation schema.
It should be added here that while it is necessary to have a working understanding of a database's semantics in order to integrate it into the TDS, much of the documentation collected and recorded into the IDDF tree is not explicitly encoding-related, but intended for the benefit of the end-user. For example, a TDS component database gives the number of basic color terms in some languages as "4.5". As a matter of encoding it is enough to know, as its documentation explains, that color term counts can be fractional numbers, and that 4.5 means "between four and five". But what does "between four and five" *mean*? It might indicate a dialectal split, inconsistencies between speakers, the presence of a marginal or dubious color term, uncertainty about the facts, or all of the above. The answer is of interest to potential users of the database, and only its creators can provide it.
Conceptually, the DTL is just one means of carrying out this transformation;[8] what is important from our present perspective is that the DTL, or an equivalent, defines a mapping of a data resource into an IDDF tree; and that the result comprises a combination of data and relevant documentation. Our vision of the IDDF is as an open format, which can be generated and manipulated by other tools. Section 3. gives more details on its structure, and on the way other components of the TDS architecture can be generalized.

2.2. What is transformed

Independently created data resources differ in a variety of ways, which need to be addressed during the integration process. The TDS makes an important distinction between differences in encoding (in the broad sense) and differences stemming from deeper theoretical or practical considerations. The former include variation in font encodings or notational conventions such as interlinear gloss labels, codes for Boolean values (*true/false* vs. *0/1*, etc), the organization of information into fields and tables, etc. The deeper differences are ultimately differences in meaning (semantics): They stem from considerations such as the theoretical commitments of a research group (including the associated terminology), the specific classificatory categories and coding decisions adopted during the construction of a database, etc.
While standardization efforts might one day lead to more uniformity in structure and encoding among databases, they will have no effect on the divergence of theoretical viewpoints and research traditions that constitutes the most intractable source of heterogeneity. These diverse viewpoints are not only dearly held by their practitioners: They are the subject matter and outcome of linguistic analysis, and cannot (should not) be replaced by any uniform, agreed-upon framework. While it might seem like a good idea to transform data into some "standard" terminology, the abstract nature of typological data collections makes this impossible. First of all, two theoretical terms are rarely if ever exactly co-extensive; even if they were, the terminology

```
<iddf:warehouse
 xmlns:iddf="http://.../ns/iddf">
  <iddf:meta>
    <iddf:scope id="tds" type="warehouse">
      ...
    </iddf:scope>
    <iddf:notion id="n1" name="language"
     scope="tds" type="root"
     key-datatype="enum">
      <iddf:label>Language</iddf:label>
      <iddf:description>
         One of the world's languages
      </iddf:description>
      ...
    </iddf:notion>
    ...
  </iddf:meta>
  <iddf:data xmlns:tds="..." ...>
    <tds:language iddf:notion="n1"
     key="...">
      ...
    </tds:language>
    ...
  </iddf:data>
</iddf:warehouse>
```

Figure 2: The top-level structure of the IDDF.

adopted by a researcher is often the result of a deliberate process, and can be felt to be as much a part of a linguistic analysis as its empirical claims. To substitute terminology under such circumstances would be a form of misrepresentation.
Accordingly, the TDS approach is to compensate for encoding differences wherever possible, by transforming the source data to adhere to, or at least be relatable to, a uniform design ("object model"); but semantic divergences are maintained, and are made explicit by suitable documentation and careful construction of relationships between various levels of metadata.
Because the various component databases each have their own schema and focus, i.e., they are heterogeneous, the aggregated IDDF data is semi-structured. To assist in the process of resource discovery by end-users, the TDS metadata includes links to a unified knowledge base, consisting of an ontology of linguistic terms and several taxonomies that provide quick domain-oriented entry points.

3. Sustainable operability with the IDDF

At the heart of the TDS, and of our vision for sustainable database operability, is the IDDF data tree. It organizes data and metadata into a unified structure that provides sufficient information for generic resource discovery, query operations, and interactive browsing tools.

3.1. The IDDF data structure

The IDDF data structure consists of two parts, a metadata schema and a data part. The metadata part defines and annotates the schema to which the data part conforms.[9] We use the term "data tree" to refer to the entire structure, since the

[8] One could, for example, convert data into XML and transform it by means of hand-written XSLT, as the TDS did during the pilot phase of the project.

[9] The IDDF can be conceptually considered as the concatenation of two documents. The document as a whole is valid XML,

two parts are closely interrelated. An abbreviated example is shown in figure 2. (A detailed example is given in the Appendix).
Figure 3 provides an informal overview of the conceptual structure of the IDDF data tree. It can be informally understood as a hierarchy of nodes (called Notions), which serve a variety of functions.

(namespace)
Grouping Notion ← (documentation)
(more groups)
Data Notion ← (documentation)
(Data) ← (value documentation)

Figure 3: Conceptual organization of the IDDF data model.

At the leaves of the tree are *Field Notions,* which correspond to fields of the component databases.[10] When the tree is built ("instantiated") by importing the databases, these Notions are populated with the data. (Note that the documentation remains in the schema portion of the IDDF, as shown above).
There are also *Grouping Notions*, which contain other Notions (either of the data or the grouping kind) and thus define the hierarchical structure of the IDDF data tree. Fields from several databases can be mapped to the same part of the tree, even the same Notion; for example, the attribute *Language Name* is a single Notion used for all databases. (The TDS organizes data according to topic, regardles of its database of origin; one could easily adopt a different policy, and map each database into a dedicated part of the hierarchy).
To facilitate management of all this data from diverse sources, Notion definitions are overlaid with a system of namespaces, which can be nested; Notions defined in a particular namespace can only be used within its scope. For example, the TDS project defines a top-level "tds scope" that provides the upper levels of semantic context, such as *clause-level phenomena.* The component databases can then define database-scoped Notions as descendants of appropriate points in the global hierarchy.
Besides its content, each Notion is associated with documentation and format information (which are stored in the schema part of the IDDF, as detailed below). Grouping Notions can be associated with a description of the kind of data they dominate, including summaries of the linguistic theory and terminology of the data providers; Field Notions can be associated with a description as well as an enumeration of possible values, which can themselves have associated documentation.

validated against a Relax NG schema that essentially ignores the data section. Validation as an IDDF document requires two passes: After the initial minimal validation, an XSLT 1.0 stylesheet is run on the metadata section to generate a complete Relax NG schema. This is then used to validate the entire IDDF document.

A sample IDDF document, and the required schema and stylesheet, are available at *http://languagelink.let.uu.nl/tds/iddf/*.

[10]The relationship to the original database fields is not one-to-one. Some Notions are in fact created by splitting up or combining several database fields.

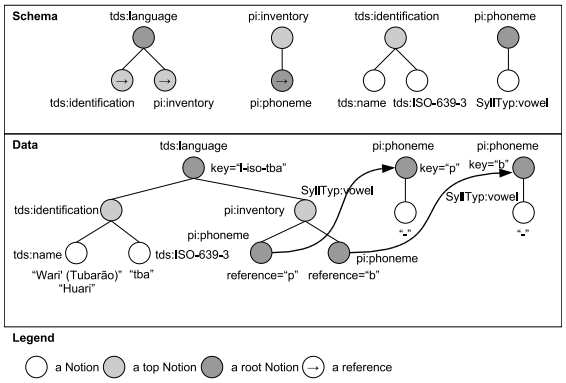

Figure 4: Graphical representation of the example IDDF schema and data tree

In many cases, a database uses a number of fields for information that belongs together and should be considered as a whole. For example, geographic latitude and longitude together make up geographic coordinates, and these together with language name, ISO code, and other essential information make up the *Language Identification* group. Each such group of fields is mapped to a subtree of the IDDF, which is identified as a *semantic context* by means of a special label assigned to its root Notion. These *Top Notions,* as they are called, are treated specially by the TDS search and browsing interface.
Larger hierarchies can be built by reusing these semantic contexts and nesting Notions inside each other. There can be multiple separate hierarchies, each with its own top-level root (called a *Root Notion*). Hierarchies can be linked to each other by establishing a primary/foreign key relationship between a Root Notion and another Root Notion. The role of Root Notions in the IDDF data model can be compared with tables in the relation model.
Figure 4 shows how the hierarchical definitions in the schema, `tds:language`, `tds:identification`, `pi:inventory` and `pi:phoneme`, are utilized during the instantiation process of the data tree. The reference leaves indicate the valid ways of linking these hierarchies together; e.g., `pi:inventory` is nested in `tds:language`; through the `pi:phoneme` reference in `pi:inventory`, the hierarchies `tds:language` and `pi:phoneme` are related.
Each of these building blocks, i.e., Notions, scopes and values, can be extensively described in the metadata. The metadata part of the IDDF document shown in the Appendix starts with describing four scopes: `tds`, `pi`, `SyllTyp` and `UPSID`. Due to space limitations, we do not discuss scopes further. A Notion schema can contain the following information:

1. an identifier;
2. a scope;
3. (optional) a label;
4. (optional) a description, possibly formatted using XHTML;
5. (optional) one or more typed links to the knowledge base;
6. (optional) one or more links to other Notions;

7. (optional) semantic data type;
8. (optional) semantic value data type and/or key data type
9. (optional) an enumeration, possibly partial, of the possible values or key values; and for each (key) value:
   (a) the literal (key) value as it appears in the data;
   (b) (optional) a label;
   (c) (optional) a description;
   (d) (optional) one or more links to the knowledge base;
   (e) (optional) one or more links to other Notions.

The example in the Appendix includes several Notions that illustrate some of these documentation units: `tds:ISO-639-3` has a description marked up with XHTML to include a link to the standards website; `pi:phoneme` and `SyllTyp:vowel` have links to concepts in the ontology, such as *segment* and *vowel*; `pi:inventory` has the semantic data type UPPC (Universal Phoneme Positioning Chart, see (Dimitriadis et al., 2008)); the metadata of Root Notions `tds:language` and `pi:phoneme` contain enumerations of their possible key values, while `SyllTyp:vowel` contains an enumeration of its values (see Figure 5).

```
<iddf:notion id="n7" name="vowel"
  scope="SyllTyp">
 <iddf:label>Vowel</iddf:label>
 <iddf:description>
  Is the segment a vowel?
 </iddf:description>
 <iddf:link type="concept" rel="as"
   href="http://...owl#vowel"/>
 <iddf:link type="concept" rel="to"
   href="http://...owl#vocalicFeatureNode"/>
 <iddf:values datatype="enum">
  <iddf:value>
   <iddf:literal>+</iddf:literal>
   <iddf:description>
    The segment is a vowel.
   </iddf:description>
  </iddf:value>
  <iddf:value>
   <iddf:literal>-</iddf:literal>
   <iddf:description>
    The segment is not a vowel.
   </iddf:description>
  </iddf:value>
 </iddf:values>
</iddf:notion>
```

Figure 5: Example of IDDF metadata associated to a notion.

3.2. The data

Since there are multiple top-level Root Notions, the data tree is actually a forest of trees, each of them an instantiation of a hierarchy dominated by a Root Notion. These trees are linked to each other using the `key` and `ref` attributes (see the Appendix). As Notions (with the exception of Top and Root Notions) can't be uniquely identified by just the combination of the scope and the identifier, each node in the tree also specifies which Notion is being instantiated, using the `iddf:notion` attribute.

Each instantiation is based on data from at least one component database. The source of a node in a tree is indicated by the `iddf:srcs` attribute. When data loaded from various databases are in agreement, they are instantiated as a single node and this attribute lists all these database scopes. But databases may also disagree. For example the Syllable Typology Database uses the name "Wari' (Tubarão)" for a certain language, while UPSID uses "Huari." Both names are stored in the IDDF document, but each in its own `iddf:value` node with a `srcs` attribute indicating its origin.[11]

3.3. The IDDF surroundings

3.3.1. The metadata and data source

The IDDF, as already mentioned, is an ordinary XML format. There are no barriers to creating valid IDDF documents with tools other than the DTL engine; one might wish, for example, to design a description language with a different syntax and primitives, perhaps for resource types that are very different than the typological databases we have been working with. Another possibility might be for a (complex) database application to directly support IDDF as an export format, without the intervention of a description language. In this case, the descriptive metadata might still need to be manually supplemented. This indicates that there could be a need for specific IDDF metadata editors. It is easy to visualize the use of a specific GUI to annotate Notions, and perhaps even to create the semantic hierarchies (contexts).

3.3.2. Links to external semantic resources

As figure 1 shows, the IDDF document can be linked to a knowledge base. In the case of the TDS this consists of an OWL ontology, developed during the course of the project, and a number of SKOS taxonomies. This allows the TDS to semantically extend queries by following the formal relationships in the ontology. The taxonomies provide alternative organizations of entry points into the data schema. Other forms of encoding knowledge, e.g. in the form of a tag cloud, could also be associated with the IDDF schema. In the TDS project, developing the metadata and the knowledge base went hand in hand. In applications of IDDF where the metadata is readily available one could also extract the knowledge base, or part of it, by mining the metadata (Feldman and Sanger, 2006; Cimiano, 2006). To get enough input for the mining algorithms one might use other related inputs, e.g., in the case of scientific databases the articles written on the basis of the data. One could also bootstrap the mining process by manually creating an initial domain-specific knowledge base.

3.3.3. Standards

Because the data in typological databases is overwhelmingly about languages, data aggregation depends crucially on reliably identifying the language that data is about. The TDS protocol relies on ISO 639-3 language codes (ISO

[11] Note that the IDDF could have also allowed each database to be mapped to a separate hierarchy, avoiding any chance of an overlap or clash.

639-3, 2007), internally and externally, to identify the language described and carry out data integration. ISO language codes are used internally as part of the key, and they are always utilized for data integration, if available. For databases or records that do not provide them, the TDS domain experts attempt to add them (by means of the DTL script), on the basis of language names and the assistance of the database creators. Again, this is a labor-intensive process but is justified in view of the value of the data, and unavoidable if the language described is to be unambiguously identified. (Once again the result is enrichment of the original data through the transformation process). In alternative application domains where cross-database integration of records is not a goal, such issues are less of a concern.

To control the proper handling of the various kinds of integrated data, the IDDF tracks the data type of each variable; the primitive types *free text* and *enumeration* can be overlaid with an open set of other (semantic) types, which are defined dynamically in the IDDF schema (that is, through the DTL) and typically apply to a group of related Notions rather than to a single one. The TDS web interface, for example, has special renderer modules for the semantic types *interlinear glossed text* (consisting of aligned morphemic tiers, a translation, etc.) and *phoneme inventory*[12]

To fully exploit this approach, it should be possible for Notions (atomic or complex) to be associated with standard data types or controlled vocabularies. Thus the ISO language code can be linked to the namespace of the appropriate authority, which provides a controlled vocabulary shared by other tools; fields conforming to other controlled vocabularies can be linked to the appropriate "data category" registered in the future ISO Data Category Registry (ISO 12620, 2008; Kemps-Snijders et al., 2008), etc. Other encoding types such as MIME types, complex structures like interlinear glossed text, etc., should similarly be reported in a standard way, and/or linked to an appropriate URI to allow their identification.

In effect, this approach extends the notion of standard data types beyond simple numeric, text and enumerated types, to more complex aggregations of data. There still work to be done in the domain of registering such resource types (the ISO Data Category Registry is designed to cover only unitary data types, not hierarchies), but the IDDF can be positioned to utilize such advances when they occur.

4. The generic user interface

The rich structure of the IDDF has made it possible to develop a generic data browser service for the typological database domain, available through the TDS server.

The TDS server is divided (somewhat imperfectly, at the moment) into an Application Programming Interface (API) and a web interface. While the web interface is closely tied to the state of today's web browsers and associated technology (including JavaScript support, etc.), the API is considerably more stable. By untangling these two better, an API can be created that provides services to multiple generations of other tools.

[12]The phoneme inventory type triggers a specific table-based rendering of a full or partial phoneme inventory.

The data browser is generic, in the sense that it does not incorporate schema or data information about any of the component databases; all such information resides in the IDDF. The browser is limited, however, by the kind of data models and displayable objects one expects to find in typological databases. Much of the data in typological databases can be displayed as tables of short values, and therefore such tables are prominent in the browser interface. There are special provisions for presenting interlinear glossed text and tables of phonemic inventories, and a mapping module for displaying data values at the geographic location of the language in question. On the other hand, there is currently no provision for displaying video streams, or (more importantly) any provision for managing data aligned to particular portions of a video stream.

While more such display modules can be developed as necessary, the browser remains generic only in the limited context of the intended application domain. For very different kinds of resources (such as experimental measurements, corpora, annotated multimedia data, etc.), one can imagine a completely different data browser that is suited to the structure of that application domain. The IDDF itself can encapsulate a wide variety of such formats.

The structure of the IDDF also makes partial compliance possible: An IDDF-aware tool, for example, could extract and manipulate interlinear glossed text from a larger resource whose full structure is not supported by the tool.

Finally, it must be acknowledged that the TDS interface (and probably any conceivable generic equivalent) is not as effective in presenting data as the best custom-built typological database interfaces; but it is more than sufficient for providing operability of the data, and other generic browsers over the IDDF data could undoubtedly do even better. In any event, several of the component databases of the TDS had no autonomous interface at all, or only a very primitive one; and the TDS interface is immensely more effective than these.

5. The IDDF in broader context

The issues we have discussed are not new, of course. We have already mentioned OAIS, the Open Archival Information System Reference Model (ISO 14721, 2003), which provides definitions of terms related to data archiving and defines roles and responsibilities in the contect of a functional model. The OAIS document discusses in some detail the requirement that archived resources should be *independently understandable* to their target community of users, and also acknowledges the issue of operability, mentioning that the native user interface sometimes encodes information essential for its understandability and noting that "maintaining Content Information-specific software over the Long Term has not yet been proven cost effective due to the narrow application of such software." In this context, our approach can be seen as a way to achieve an economy of scale, by transferring the burden of operability to domain-wide generic tools which manage the generic IDDF format. This will reduce the burden of maintaining operability in a very scalable way, and will *hopefully* prove to be acceptably cost-effective. Whether this expectation will be realized can only be determined in the long term.

The OAIS also devotes attention to issues of archiving for the *Long Term,* defined as a period long enough to raise issues of adapting to new technology or a changing user community. The latter issue, of a changing user community, is not one we address directly; our user-oriented documentation is intended to make data independently understandable to present-day linguists, not future ones. However, there is sufficient creativity and variation in today's linguistic theories that even for understandability by contemporary linguists, they must be documented in some detail. Thus the documentation that is necessary today will serve as a good basis for understandability in the future.

Mapping a database to IDDF format requires manual enrichment of the resource with metadata that cannot be automatically automatically computed from its schema. Typically, the creators or maintainers of the original resource are asked to provide supplementary information (concerning both formal and user-oriented metadata) that is not embedded in the native data dump. While this is necessary if the resource is to be independently understandable (and is therefore indispensable to real data preservation), it means that the approach is applicable only to data of sufficient value to merit this sort of intervention. For very large-scale data collection projects, this kind of attention to each incoming resource might well be impossible. In such cases, the IDDF architecture can still support operability at a lower level, comparable with that provided by present-day solutions: The resource, along with whatever documentation is available, is imported in a form that simply mirrors the relational structure of the original database. Such data cannot be rendered in the most appropriate way, but can be browsed and manipulated at the relational table level by suitable generic software. This gives a level of functionality equivalent to viewing a database with a DBMS administration tool.

For large-scale data integration, then, the IDDF "dumbs down" to a level of functionality comparable to that provided by some existing large-scale archiving solutions. For example, (Heuscher et al., 2004) addresses the task of archiving the records of the Swiss Federal Administration, which are reported to be growing at a rate of some twenty terabytes per year. The SIARD project achieves "software-invariant" archiving of relational databases via transformation, at time of import, to a consensus SQL model (SQL-3). "On principle, functionality (i.e. software, hardware) is not archived" (Heuscher et al., 2004, p. 1). Archived data can be browsed at the relational table level by reloading into a conforming DBMS. The Chronos system (Brandl and Keller-Marxer, 2007) maintains data in its original dump format and provides low-level user access, again at the level of browsing the relational structure and tables, by supporting "on-the-fly migration" from an ever-growing collection of dump formats. This approach, while allowing archives to be maintained on a very large scale, does not provide high-level operability, especially for complex data of the type we have been concerned with. The IDDF architecture allows higher levels of operability to be achieved where this is practical, but can be (under)utilized to yield low-level operability for large volumes of complex data.

Roles and responsibilities

The architecture described relies on software support at two levels: On the input side, there must be tools to support the creation of IDDF documents. On the access side, there must be a generic data browser for any supported application domain. The two levels of tools have different maintenance requirements:

Once a resource has been mapped to the IDDF format, input-side software is not needed for its continued operability (unless, of course, the original resource changes and needs to be re-imported). An archive that stores resources in IDDF format need only ensure the continuous availability of appropriate data browsers on the access side. As such browsers become outdated or unmaintainable, they must be replaced by new IDDF-aware browsers with analogous functionality.

For IDDF-based archiving to be practical, however, suitable conversion tools are necessary. In the TDS architecture, IDDF generation is carried out by the TDS import engine, which is driven by DTL schemas and relies on plugins that grant it access to various database and dump formats.[13]

In principle, responsibility for maintaining IDDF generation tools (or using them) need not rest with the archive. A resource provider can arrange to export their data in IDDF format, perhaps via a DTL-like transformation module or in some other way. If the format should become widespread, one could even expect general-purpose DBMS applications to support such conversions. For the meantime, however, archives relying on the IDDF architecture must also address the problem of bringing data to IDDF form.

6. Conclusions

As we have seen, the problem of sustained operability of complex resources is ultimately traceable to the limitations of common storage and interchange formats, which do not provide sufficient information for generic navigation. By focusing on the particular (but broad) domain of typological databases, we have shown that the rich IDDF architecture can integrate sufficient information for a generic data browser adapted to the types of data common in typological databases. The approach is extensible and suitable for alternative application domains, as long as there is some homogeneity in the kind of data that is being collected (regardless of how each resource has chosen to present it). In effect, the idea of storing resources in a standard format that can be managed with generic tools is extended to families of complex formats that represent similar data collections.

A notable aspect of the TDS is its focus not only on metadata pertaining to encoding formats and operability, but also on documentation intended for the end-user. Because of the abstract nature of linguistic analysis, such user-oriented

[13]Note that while a diverse collection of such formats must be specifically supported, there is no need to support long-obsolete formats. When an archive no longer plans to archive databases stored on eighty-column punched cards, there will be no need to maintain support for this format (or the associated hardware). Once a resource is converted to IDDF, the original format is irrelevant to operability.

documentation is essential for the proper interpretation of high-level resources like typological databases.

More generally, by collecting and centralizing metadata and documentation, the TDS archival procedure safeguards the interpretability (and therefore true operability) of the archived data.

In the context of an archival environment, the IDDF architecture also solves the problem of versioning and citeability of evolving resources: Instead requiring resource creators to maintain multiple versions of their database, an archive can simply host multiple versions of a resource, and make them available (and operable) as if they were separate databases. Hence the archive can provide a versioned, operable mirror of the database without the need for any versioning provisions in the database schema itself.

In short, the rich IDDF format can support sustainable operability of complex resources, by allowing a critical mass of such resources to be managed through generic (but domain-specific) tools.

7. References

Stefan Brandl and Peter Keller-Marxer. 2007. Long-term archiving of relational databases with Chronos. In *First International Workshop on Database preservation (PresDB '07)*.

Philipp Cimiano. 2006. *Ontology Learning and Population from Text*. Springer-Verlag, Berlin.

A. Dimitriadis, A. Saulwick, and M. Windhouwer. 2005. Semantic relations in ontology mediated linguistic data integration. In *Proceedings of the E-MELD Workshop on Morphosyntactic Annotation and Terminology: Linguistic Ontologies and Data Categories for Linguistic Resources*, Cambridge, Massachusetts, July.

A. Dimitriadis, M. Windhouwer, A. Saulwick, R. Goedemans, and T. Bíró. 2008. How to integrate databases without starting a typology war: The Typological Database System. In S. Musgrave and M. Everaert, editors, *The Use of Databases in Cross-Linguistic Studies*. Mouton de Gruyter. To appear.

Ronen Feldman and James Sanger. 2006. *The Text Mining Handbook*. Cambridge University Press.

Stephan Heuscher, Stephan Järmann, Peter Keller-Marxer, and Frank Möhle. 2004. Providing authentic long-term archival access to complex relational data. In *European Space Agency Symposium "Ensuring Long-Term Preservation and Adding Value to Scientific and Technical Data"*.

ISO 12620. 2008. Terminology and other language resources – Data categories – Specification of data categories and management of a data category registry for language resources. To appear.

ISO 14721. 2003. Space data and information transfer systems – Open archival information system – Reference model.

ISO 639-3. 2007. Codes for the representation of names of languages – Part 3: Alpha–3 code for comprehensive coverage of languages.

M. Kemps-Snijders, M. Windhouwer, P. Wittenburg, and S.E. Wright. 2008. ISOcat: Corralling data categories in the wild. In *Proceedings of the International Conference on Language Resources and Evaluation*, Marrakech, Morocco, May.

A. Saulwick, M. Windhouwer, A. Dimitriadis, and R. Goedemans. 2005. Distributed tasking in ontology mediated integration of typological databases for linguistic research. In *Proceedings of the International Workshop on Data Integration and the Semantic Web*, Porto, Portugal, June.

Appendix: A longer IDDF example

We include here a sample IDDF structure. The first part (`<meta/>`) integrates data schema and documentation, while the `<data/>` element contains the sparse data.

```
<iddf:warehouse
   xmlns:iddf="http://.../ns/iddf">
 <iddf:meta>
  <iddf:datatype id="UPPC"/>
  <iddf:scope id="tds" type="warehouse">
   <iddf:label>
    Typological Database System
   </iddf:label>
   <iddf:scope id="pi">
    <iddf:label>
     Phoneme Inventories
    </iddf:label>
    <iddf:scope id="SyllTyp" type="database">
     <iddf:label>
      Syllable Typology Database
     </iddf:label>
    </iddf:scope>
    <iddf:scope id="UPSID" type="database">
     <iddf:label>
      UCLA Phonological Segment
      Inventory Database
     </iddf:label>
    </iddf:scope>
   </iddf:scope>
  </iddf:scope>
  <iddf:notion id="n1" name="language"
     scope="tds" type="root">
   <iddf:label>Language</iddf:label>
   <iddf:description>
    One of the world's languages
   </iddf:description>
   <iddf:keys datatype="enum">
    <iddf:key>
     <iddf:literal>
      l-iso-tba
     </iddf:literal>
     <iddf:label>Aikan&#227;</iddf:label>
    </iddf:key>
    ...
   </iddf:keys>
   <iddf:notion ref="n2"/>
   <iddf:notion ref="n5"/>
  </iddf:notion>
  <iddf:notion id="n2" name="identification"
     scope="tds" type="top">
   <iddf:label>
    Language identification
   </iddf:label>
   <iddf:notion id="n3" name="name"
```

```
    scope="tds">
  <iddf:label>Name</iddf:label>
  <iddf:values datatype="free"/>
 </iddf:notion>
 <iddf:notion id="n4" name="ISO-639-3"
    scope="tds" >
  <iddf:label>ISO 639-3 code</iddf:label>
  <iddf:description
     xmlns:xhtml="http://...">
   The code as assigned to the
   language in the
   <xhtml:a href="http://...">
    ISO 639-3 standard
   </xhtml:a>.
  </iddf:description>
  <iddf:values datatype="enum">
   <iddf:value>
    <iddf:literal>tba</iddf:literal>
   </iddf:value>
   ...
  </iddf:values>
 </iddf:notion>
</iddf:notion>
<iddf:notion id="n5" name="inventory"
   scope="pi" type="top"
   datatype="UPPC">
 <iddf:label>
  Phoneme inventory
 </iddf:label>
 <iddf:notion ref="n6"/>
</iddf:notion>
<iddf:notion id="n6" name="phoneme"
   scope="pi" type="root">
 <iddf:label>Phoneme</iddf:label>
 <iddf:link type="phoneme" rel="as"
    href="http://...owl#segment"/>
 <iddf:keys datatype="enum">
  <iddf:key>
   <iddf:literal>p</iddf:literal>
  </iddf:key>
  <iddf:key>
   <iddf:literal>b</iddf:literal>
  </iddf:key>
  ...
 </iddf:keys>
 <iddf:notion id="n7" name="vowel"
    scope="SyllTyp">
  <iddf:label>Vowel</iddf:label>
  <iddf:description>
   Is the segment a vowel?
  </iddf:description>
  <iddf:link type="concept" rel="as"
     href="http://...owl#vowel"/>
  <iddf:link type="concept" rel="to"
     href="http://...owl#vocalicFeatureNode"/>
  <iddf:values datatype="enum">
   <iddf:value>
    <iddf:literal>+</iddf:literal>
    <iddf:description>
     The segment is a vowel.
    </iddf:description>
   </iddf:value>
   <iddf:value>
    <iddf:literal>-</iddf:literal>
    <iddf:description>
     The segment is not a vowel.
    </iddf:description>
   </iddf:value>
  </iddf:values>
 </iddf:notion>
</iddf:notion>
</iddf:meta>
<iddf:data
  xmlns:tds="http://.../ns/iddf/tds"
  xmlns:pi="http://.../ns/iddf/pi"
  xmlns:SyllTyp="http://.../ns/iddf/SyllTyp"
>
 <tds:language iddf:notion="n1"
    key="l-iso-tba"
    iddf:srcs="SyllTyp UPSID">
  <tds:identification iddf:notion="n2"
     iddf:srcs="SyllTyp UPSID">
   <tds:name iddf:notion="n3"
     iddf:srcs="SyllTyp UPSID">
     <iddf:value srcs="SyllTyp">
      Wari' (Tubar&#227;o)
     </iddf:value>
     <iddf:value srcs="UPSID">
      Huari
     </iddf:value>
   </tds:name>
   <tds:ISO-639-3 iddf:notion="n4"
     iddf:srcs="SyllTyp UPSID">
     <iddf:value
        srcs="SyllTyp UPSID">
      tba
     </iddf:value>
   </tds:ISO-639-3>
  </tds:identification>
  <pi:inventory iddf:notion="n5"
     iddf:srcs="SyllTyp UPSID">
   <pi:phoneme iddf:notion="n6" ref="p"
     iddf:srcs="SyllTyp UPSID"/>
   <pi:phoneme iddf:notion="n6" ref="b"
     iddf:srcs="UPSID"/>
   ...
  </pi:inventory>
 </tds:language>
 ...
 <pi:phoneme iddf:notion="n6" key="p"
    iddf:srcs="SyllTyp UPSID">
  <SyllTyp:vowel iddf:notion="n7"
     iddf:srcs="SyllTyp">
   <iddf:value srcs="SyllTyp">
    -
   </iddf:value>
  </SyllTyp:vowel>
 </pi:phoneme>
 <pi:phoneme iddf:notion="n6" key="b"
    iddf:srcs="SyllTyp UPSID">
  <SyllTyp:vowel iddf:notion="n7"
     iddf:srcs="SyllTyp">
   <iddf:value srcs="SyllTyp">
    -
   </iddf:value>
  </SyllTyp:vowel>
 </pi:phoneme>
 ...
</iddf:data>
</iddf:warehouse>
```