CHAPTER 11

Curating the Typological Database System

Menzo Windhouwer[a,1,2], Alexis Dimitriadis[b] and Vesa Akerman[c]
[a]Meertens Institute, [b]Utrecht University, [c]Data Archiving and Networked Services (DANS)

ABSTRACT

The Typological Database System (TDS), which provides integrated access to a dozen independently created typological databases, was launched in 2007. Due to the pace of change in web technologies, the original software has for some time been edging toward obsolescence. CLARIN-NL granted funding to the TDS Curator project to migrate this valuable resource to a more durable platform, archiving the data and converting its interface to a true web service architecture that can continue to provide interactive access to the data. This chapter describes the architecture of the new system, and the Integrated Data and Documentation Format (IDDF) on which it is based.

11.1 Introduction

The Typological Database System (TDS) is a web-based resource that provides integrated access to a collection of independently created typological databases. Typological databases are used for research in linguistic typology, 'the study of patterns that occur systematically across languages' (Croft, 2003), and consulted by linguists and others looking for a high-level, cross-linguistic view of particular phenomena. Combining several typological databases provides advantages in scale, as well as the opportunity to look for relations among features. The TDS was developed with support from NWO grant 380-30-004 / INV-03-12 and from participating universities, and launched in 2007. It provides access to and extended documentation for its component databases, through a uniform structure and search interface (Dimitriadis et al., 2009). However, web technologies evolve

[1] Corresponding author: menzo.windhouwer@meertens.knaw.nl
[2] At the time of the CLARIN-NL TDS Curator project (2011–2012), Menzo Windhouwer was working at the Max Planck Institute for Psycholinguistics.

How to cite this book chapter:
Windhouwer, M, Dimitriadis, A and Akerman, V. 2017. Curating the Typological Database System. In: Odijk, J and van Hessen, A. (eds.) *CLARIN in the Low Countries*, Pp. 123–132. London: Ubiquity Press. DOI: https://doi.org/10.5334/bbi.11.

rapidly and thus pose a challenge to software sustainability. Through its Project Call 1, CLARIN-NL granted funding to the TDS Curator project to save the valuable TDS resource in a durable, archival environment and convert access to it into a true web service architecture, thus safeguarding future access to the TDS data.

11.2 The Architecture of the Typological Database System

Figure 11.1 gives a global overview of the TDS architecture. Data from the component databases (at the bottom of the diagram) is pushed through an extensive, manually supervised Extraction, Transformation and Loading (ETL) processing chain and integrated into one data resource, based on knowledge in various knowledge bases (right side of the diagram). The end user interacts with the system to access the data and knowledge bases through its web interface (at the top). The knowledge bases contain all TDS knowledge about the component databases and the linguistic domain. This consists of various interlinked specifications: a set of database-specific ontologies, one global linguistic ontology and (currently) two topic taxonomies. Maintenance of the knowledge base is supported by the TDS Workbench (right) and other custom-built or general-purpose tools, including an ontology editor.

To guarantee continued access to the valuable data integrated into this system, the TDS Curator project addressed three weak spots in the architecture:

1. Although parts of the knowledge base already used W3C recommendations – the Web Ontology Language (OWL; W3C, 2016a) for the linguistic ontology, and the Simple Knowledge Organization System (SKOS; W3C, 2016b) for the topic taxonomies – the core of the system, consisting of the collection of data imported, merged and enriched from the component databases and the database-specific knowledge basis, relies on a proprietary format. This format had already been cleaned up, generalised and formalised as the XML-based Integrated Data and Documentation Format (IDDF; Windhouwer and Dimitriadis, 2008), but IDDF

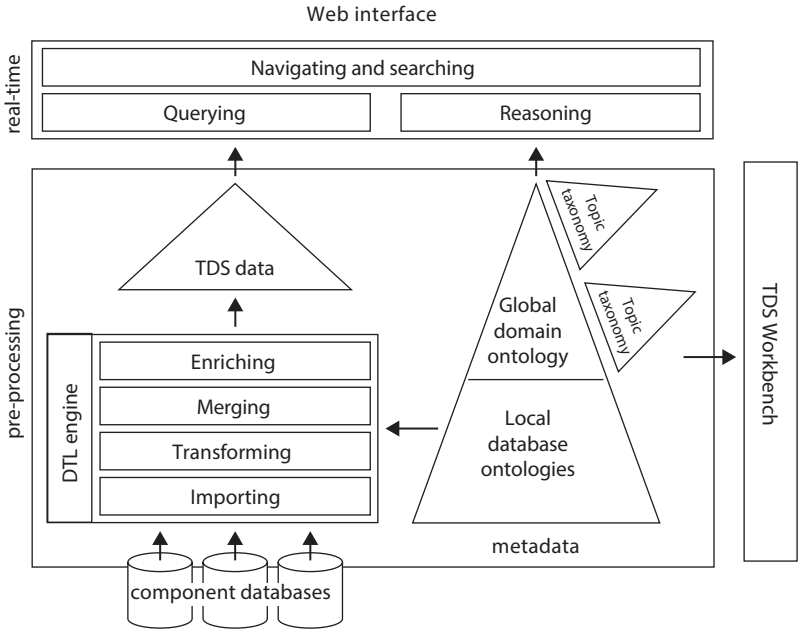

Figure 11.1: The TDS system architecture.

had not actually been implemented in the TDS system architecture. The TDS Curator project transformed the TDS data into the IDDF format, and they are now archived by the Data Archiving and Networked Services (DANS)[3] in their online EASY archiving system.

2. The original TDS engine did not fully implement a client-server architecture, i.e., a clean separation into a back end (the server) and interface components communicating through an explicit Application Programming Interface (API). Instead, features and technical idiosyncrasies of the framework chosen for the interface were deeply interwoven into the ragtag API. In the TDS Curator, an IDDF-driven web API was designed and implemented. This API can potentially be used by other tools besides the current TDS interface.
3. The TDS web interface, which predated HTML5, was based on the Backbase UI framework (Van Emde Boas and Ilinsky, 2009). Unfortunately, this turned out to be a poor technology bet, as support for this framework decreased rapidly and now has completely disappeared. In the TDS Curator project, the interface and the underlying technology have been renewed. However, the world of browser technology remains highly volatile, so unfortunately sustainability problems do remain.

The next sections discuss these problem areas and the implemented solutions in more depth.

11.2.1 Integrated Data Documentation Format

The aim of the TDS project was to integrate a number of typological databases, allowing access to and correlating their information using one uniform interface. The basic technological requirements for this integration are close to the Extraction, Transformation and Loading (ETL) phase in data warehousing. However, a not-so-common need was to add extensive documentation (part of the database-specific knowledge base) to the data and data structures loaded from the component databases. Many of the databases were built over the years by researchers for their own purposes, and had virtually no documentation. The linguistic knowledge designer of the TDS had to solicit this information from these researchers and make it digitally available to the end-users of the TDS (see Figure 11.2). This was done by storing the knowledge in the knowledge base and adding this

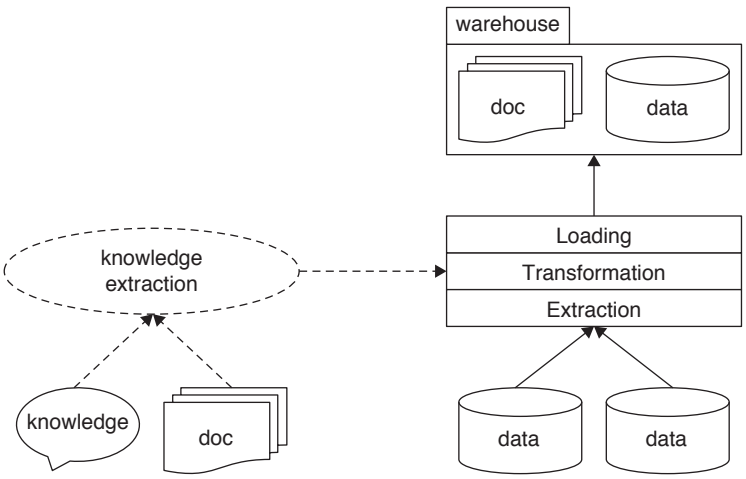

Figure 11.2: Extraction, Transformation and Loading (ETL).

[3] DANS was the (candidate) CLARIN centre partner in the project and provides the project's longterm archiving services.

information to the data source in the transformation stage of the ETL phase, resulting in a heavily interconnected combination of data and documentation in the data warehouse.

An IDDF document consists of two major sections: one for the documentation and one for the data. The IDDF schema is expressed in Relax NG (ISO, 2008) and Schematron (ISO, 2006a), both of which are validation languages for XML documents. The hierarchical nature of XML makes it very suitable for IDDF documents. To avoid having to manually overspecify hundreds of data fields and values, and in order to capture some of their interrelationships and higher-level organisation, data fields and values are grouped and documented in so-called semantic contexts. For example, one context contains all the fields related to language identification, while other contexts contain all the fields for particular linguistic phenomena. Data and its documentation will always have to be shown in its semantic context to allow proper interpretation, e.g. to know that this name is the name of a language and not the name of an author of a scientific article. As it is possible to embed a semantic context within another context, a hierarchy, i.e., a tree, of semantic contexts can be built.

Many typological databases consist entirely of information describing languages as a whole (e.g., 'basic word order'); such databases are effectively structured as a single relational table, and can be hierarchically represented by nesting semantic contexts, starting from the root context for *language*. However, more sophisticated typological databases contain data about multiple constructions per language, while other data sources contain information that is not restricted to single languages (e.g., a universal phoneme inventory). This requires more than one hierarchy, i.e., one for languages and one for each of these other entity types, which can refer to each other through foreign/primary key relationships. The data model thus becomes a network of hierarchies. In Figure 11.3 each type of semantic context is represented by a triangle with a specific colour and size. The documentation section describes the types of semantic context, while in the data section they are instantiated – here, the same type can appear multiple times. Relationships between entities are shown by directed edges.

The major aim of the documentation section is to describe the data, which will be stored in the data section. The basic documentation building blocks, to which data will get associated later on, are called *notions*: this term was introduced to make a distinction between the fields and their

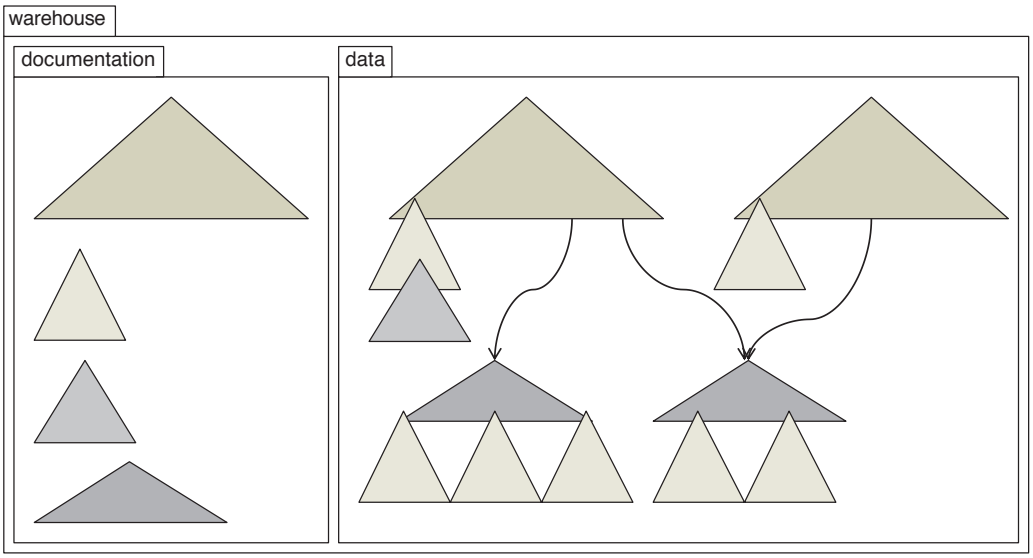

Figure 11.3: Integrated Data and Documentation.

values that get loaded from a component database, on the one hand, and more formal knowledge base building blocks like concepts and topics, on the other hand. Notions are grouped together in small hierarchies, which correspond to the previously introduced semantic contexts. For example, the notions *title*, *name*, *affiliation* and *email* could be grouped together under an *author* notion. Notions that are at the root of such a hierarchy are called *top* notions, i.e., *author* would be a top notion. Semantic contexts are themselves organised into a bigger hierarchy; for example, the author context can be reused in a bibliographic entry semantic context. Notions that are uppermost in these bigger hierarchies are called *root* notions and are required to have a primary key. (By their nature, as mentioned above root notions are also top notions.)

In addition to documenting the data, it is also important to track their provenance. For this, the documentation section defines *scopes*. Each notion belongs to a scope, and only data sources with access to that scope can actually instantiate the notion. On first glance this seems to hinder integration, but actually scopes are nested, and it is allowed that a descendant scope instantiates notions from higher-level scopes. Scopes at the lowest level are tied to one of the actual data sources, while the higher-level scopes express semantic similarity.

```
<iddf:documentation>
    <iddf:scope xml:id="tds">
         <iddf:label>Typological Database System</iddf:label>
         <iddf:scope xml:id="pi">
              <iddf:label>Phoneme Inventories</iddf:label>
              <iddf:scope xml:id="upsid" type="datasource">
                   <iddf:label
>UCLA Phonological Segment Inventory</iddf:label>
              </iddf:scope>
         </iddf:scope>
    </iddf:scope>
    <iddf:notion xml:id="n1" scope="tds" name="identification"
     type="top">
       <iddf:label>Language identification</iddf:label>
       <iddf:link rel="datcat"
        href="http://www.isocat.org/datcat/DC-3932"/>
       <iddf:notion scope="tds" name="name">
            <iddf:label>Name</iddf:label>
            <iddf:values datatype="FREE"/>
       </iddf:notion>
    </iddf:notion>
    <iddf:notion scope="tds" name="language" type="root">
         <iddf:label>Language</iddf:label>
         <iddf:description
>One of the world's languages</iddf:description>
        <iddf:keys>
            <iddf:key>
                 <iddf:literal>l-iso-tba</iddf:literal>
                 <iddf:label>Aikan\~a</iddf:label>
            </iddf:key>
        </iddf:keys>
        <iddf:notion ref="n1"/>
    </iddf:notion>
</iddf:documentation>
```

The example above shows an IDDF documentation section. It declares scopes for TDS, phoneme inventories and the UCLA Phonological Segment Inventory Database (UPSID) data source,[4] and the top notion *identification* which is reused by the root notion *language*. The example also hints at

[4] For an overview of and brief introduction to the component databases included in the TDS see Dimitriadis et al. (2009).

some additional features, which help to extend the coverage of the documentation or make explicit the nature of the data:

- labels and descriptions;
- (key) value enumerations, which can be defined as total or partial;
- links from scopes, notions, (key) values or other documentation elements to (online) resources with further information, e.g. a data category specification or website;
- a hierarchy of relation types, which can be instantiated by links;
- a hierarchy of data types to be used by (key) values, which are in general of a semantic nature, e.g. the 'vernacular tier' of a glossed sentence, and can be used to trigger specific data renderers in the user interface;
- annotations, which can be used in the data section, e.g. to mark the confidence level of a data value or add a comment; and
- associated resources, e.g. an ontology or a taxonomy, which can be referred to by links.

Once notions have been declared, the data section is populated with instances of them, as shown below.

```
<iddf:data xmlns:tds=".../tds" xmlns:pi=".../pi"
  xmlns:sylltyp=".../sylltyp" xmlns:upsid=".../upsid">
    <tds:language key="l-iso-tba" iddf:srcs="upsid">
        <tds:identification>
            <tds:name iddf:src="sylltyp">
                <iddf:value ann="v1" src="sylltyp"
>Wari' (Tubar&#227;o)</iddf:value>
                 <iddf:annotation ann="v1" type="marker"
>UNSURE</iddf:annotation>
            </tds:name>
            <tds:name iddf:src="upsid">Huari</tds:name>
        </tds:identification>
    </tds:language>
</iddf:data>
```

Notice that the schema as defined in the documentation section is very loosely interpreted. For example, IDDF has no facilities to declare the cardinality of the notion *name,* and depending on the actual data loaded from the component databases there can be from zero to many instantiations in the data section. In this example there are two names, one ('Wari' (Tubarão)') provided by SyllTyp (Syllable Typology Database) and the other ('Huari') by the UPSID database. The only mandatory information is provenance information and keys for root notions. Some information can be expressed in multiple ways, so a canonical form is defined. This form can be created for any IDDF document and simplifies the development of tools, i.e., the tools can use the canonical path to this information and disregard the alternative paths.

Finally, validation of IDDF documents consists of two phases, implemented with the help of Relax NG, Schematron, NVDL (ISO, 2006b) and XSLT (W3C, 2007):

1. validate the documentation section against the general IDDF RELAX NG + Schematron schema;
2. validate the data section against the document-specific IDDF RELAX NG + Schematron schema created by a transformation to make explicit the schema implied by the documentation section.

The resulting XML-based IDDF provides a rich container for the collected semi-structured data and their documentation. An IDDF document is sufficiently self-describing to support data use and integrity checks, even without the software infrastructure described in this chapter.

In the TDS Curator project the ETL tool chain was left intact, i.e., the tool chain still generates the TDS internal format, with the resulting integrated data source then converted into IDDF.

11.2.2 An IDDF-based Application Programming Interface

The TDS interface is browser-based: the user's web browser interacts via HTTP with the back-end server. The back-end server provides access to IDDF documents, so it is natural to base the client-server interaction, the API, on concepts natural to IDDF. The following methods are provided by the API:

IDDF collection

documents()
: Gets all IDDF documents stored in the system, and their description.

IDDF documentation section

annotation(file, ann)
: Gets the definition of a specific annotation.

annotations(file)
: Gets the definitions of all annotations.

context(file, notion, keys=no, values=no, desc=no)
: Gets the definition of a specific semantic context identified by the top notion.

context-links(file, notion)
: Gets all the semantic contexts that refer to a specific top notion.

datatype(file, datatype)
: Gets the definition of a specific data type.

datatypes(file)
: Gets the definition of all data types.

fulltext-filter(file, notion, text, labels=no)
: Gets matches of the given text within the context of a specific root notion.

fulltext-search(file, text, labels=no)
: Gets notions which match the given text.

key(file, notion, key)
: Gets the definition of a specific key value of a specific notion.

keys(file, notion, random=25)
: Gets the definitions of all key values of a specific notion.

links(file)
: Gets an overview of all links between semantic contexts.

notion(file, notion, keys=no, values=no, desc=no)
: Gets the definition of a specific notion.

relation(file, rel)
: Gets the definition of a specific relation.

relations(file)
: Gets the definition of all relations.

root-links(file, notion)
: Gets the notions that refer to a specific root notion.

roots(file)
: Gets all the root notions.

roots-links(file)
: Get the notions that refer to a root notion.

scope(file, scope)
: Get the definition of a specific scope.

scopes(file)
Get the definition of all scopes.
value(file, notion, value)
Get the definition of a specific value of a specific notion.
values(file, notion, random=25)
Get the definition of all values of a specific notion.
IDDF data section
context-instance(file, id, labels=no, deep=no)
Gets a single instance of a specific top notion.
query(file, query, labels=no, pageSize?, startFrom=1)
Query the data section.
root-instances(file, notion, values=no, labels=no, pageSize?, startFrom=1)
Gets all instances of a specific root notion.

The eXist XML database management system (eXist Solutions, 2016) was selected as back-end technology, and the API was implemented using a mixture of XQuery (W3C, 2010) and XSLT. The API uses a Remote Procedure Call (RPC) approach with a single endpoint, i.e., the actual method invoked is specified as a parameter. For example, the first API method is invoked as follows:

```
tds.dans.knaw.nl/tds-services.xql?service=documents
```

11.2.3 A New Web User Interface

The old TDS web interface was deeply intertwined with the old TDS API, so a new web interface was implemented for the new, IDDF-based TDS API. Since HTML5 was still under development and its support in browsers was still immature in 2010, Adobe Flash was selected for the implementation of this front end.[5]

In this new interface, user interaction starts with the selection of an IDDF document to inspect. For inspection, there are two major options: explore the instances using the language browser (see Figure 11.4), or build and execute a query. The query interface follows the three-stage model that the old interface also used:

1. exploration of the notions to find the ones which can help answering the user's question;
2. formulation of the actual query by specifying selection and projection criteria; and
3. execution of the query, followed by the display and inspection of the results.

The new interface lacks some of the more advanced features of the old web interface, e.g. geographical views of result sets and joins between different entity types (roots). This is due to a combination of limited development resources and a desire to avoid features that are particularly likely to stop working over time because of reliance on volatile browser technology or third-party services. Unfortunately, support for Adobe Flash as a client-side technology is also diminishing in favour of HTML5, and there is a possibility the former might disappear soon. But, while the web client may already be approaching the beginning of obsolescence, the clean IDDF-based API created in the TDS Curator project ensures that the server side is isolated, and that less work will be needed when the time comes to create the next, HTML5-based, web interface. However, even when based on HTML5 a state-of-the-art interface also depends on the interactivity provided by JavaScript; and this area is still ridden by browser differences, although less so than in the

[5] The Meertens Institute, the project partner who developed the new frontend, has extensive experience with Adobe Flash

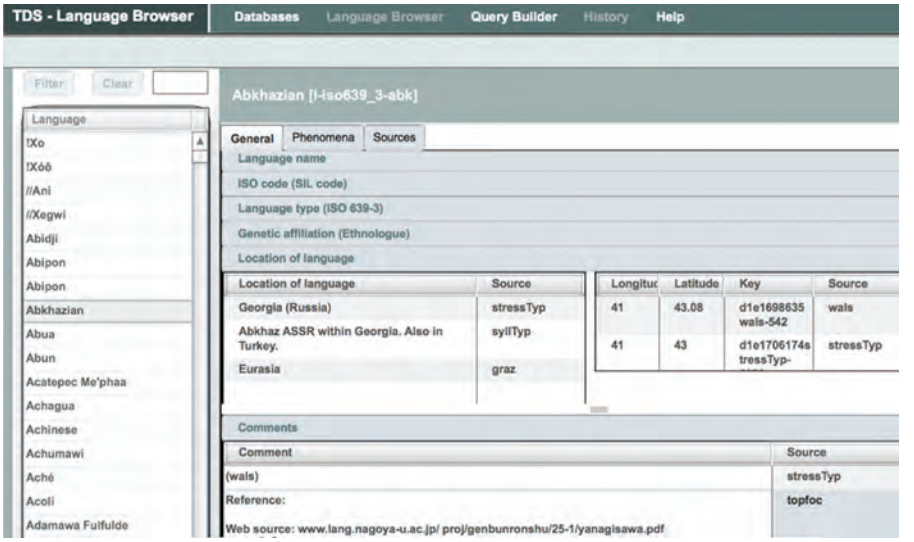

Figure 11.4: The TDS language browser (tds2.dans.knaw.nl).

times of the first TDS interface. Overcoming these differences is hardly achievable for a small-scale development team, so one has to rely on major frameworks. However, the availability and persistence of these frameworks is still largely ruled by hypes, i.e., it remains a challenging problem to select a technology/framework with a long-term horizon.

The approach of the TDS Curator project was to design a generic back-end data structure and API, so that an economy of scale can eventually be achieved through reuse of these components for other resources. Even if changes in browser technology outstrip resources for keeping the TDS web service in operation, the self-documenting format of the IDDF at least preserves and makes available the accumulated data in static, portable form.

11.3 Conclusion

CLARIN-NL's TDS Curator project provided crucial support for the sustainability of this data source. The collected data of the TDS have been safely archived at DANS as a single, integrated IDDF document and in the form of a separate IDDF document for each database. The IDDF-based web service is operational, making it possible to browse and query these documents interactively for the medium-term. Unfortunately the fast cycles of appearance and disappearance of client-side technology are already catching up with the new interface, so that a more sustainable solution is still needed in this area.

In CLARIAH, the successor to the CLARIN-NL project, work is planned to integrate new component databases into the TDS. This will likely involve curation of the tools used in the ETL phase. Work on the web interface is not yet foreseen in CLARIAH.

Another interesting direction for the TDS data is the Linked Data approach. The steadily growing Linguistic Linked Open Data (LLOD) cloud (Chiarcos, Hellmann, Nordhoff et al., 2012; LIDERproject, 2016) already contains many valuable linguistic resources, and the TDS would add typological knowledge to it. While the original TDS project predated the rise of Linked Data technologies, the use of semantic contexts in IDDF resembles the use of components in the Component Metadata Data Infrastructure (CMDI; Broeder, Windhouwer, Van Uytvanck et al., 2012) as used by CLARIN. A successful mapping of CMDI to Linked Data has already been created (see chapter 8), and the patterns learned can most likely be used to map IDDF to Linked Data.

Acknowledgements

The CLARIN-NL TDS Curator project was a collaboration between the University of Utrecht, DANS, the Meertens Institute and the Max Planck Institute for Psycholinguistics. The authors would like to thank these partner institutions and especially their colleagues: Marjan Grootveld, Marc Kemps-Snijders and Rob Zeeman.

References

D. Broeder, M. Windhouwer, D. Van Uytvanck T. Goosen and T. Trippel (2012). CMDI: a Component Metadata Infrastructure. In the *Proceedings of the Metadata 2012 Workshop on Describing Language Resources with Metadata: Towards Flexibility and Interoperability in the Documentation of Language Resources*. LREC 2012 Istanbul, Turkey, May 22, 2012.

C. Chiarcos, S. Hellmann, S. Nordhoff, S. Moran, R. Littauer, J. Eckle-Kohler, I. Gurevych, S. Hartmann, M. Matuschek and C.M. Meyer (2012). *The Open Linguistics Working Group*. LREC 2012. Istanbul, Turkey, May 23-25, 2012.

W. Croft (2003). *Typology and Universals*, 2nd edition. Cambridge University Press, 2003

A. Dimitriadis, M. Windhouwer, A. Saulwick, R. Goedemans and T. Bíró (2009). How to integrate databases without starting a typology war: The Typological Database System. In S. Musgrave, M. Everaert and A. Dimitriadis (eds), *The Use of Databases in Cross-Linguistic Studies*. Mouton de Gruyter, March 2009.

G. van Emde Boas and S. Ilinsky (2009). *Backbase 4 RIA Development*. Packt Publishing, 2009.

eXist Solutions (2016). *eXistdb - The Open Source Native XML Database*. exist-db.org. Accessed on June 15, 2016.

International Organization for Standardization (ISO, 2006a). *Information technology — Document Schema Definition Language (DSDL) — Part 3: Rule-based validation — Schematron*. ISO/IEC 19757-3. Geneva, June 1, 2006.

International Organization for Standardization (ISO, 2006b). *Information technology — Document Schema Definition Language (DSDL) — Part 4: Namespace-based Validation Dispatching Language (NVDL)*. ISO/IEC 19757-4. Geneva, June 1, 2006.

International Organization for Standardization (ISO, 2008). *Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*. ISO/IEC 19757-2. Geneva, December 15, 2008.

LIDER project (2016). *Linguistic Linked Open Data*, linguistic-lod.org. Accessed on January 18, 2016.

M. Windhouwer and A. Dimitriadis (2008). Sustainable operability: Keeping complex resources alive. In A. Witt, G. Rehm, T. Schmidt, K. Choukri and L. Burnard (eds), Proceedings of the LREC 2008 Workshop *Sustainability of Language Resources and Tools for Natural Language Processing* (SustainableNLP08). Marrakech, Morocco, May 31, 2008.

World Wide Web Consortium (W3C, 2007). M. Kay (ed), *XSL Transformations (XSLT) Version 2.0*. January 23, 2007.

World Wide Web Consortium (W3C, 2010). S. Boag, D. Chamberlin, M.F. Fernández, D. Florescu, J. Robie and J. Siméon (eds), *XQuery 1.0: An XML Query Language (Second Edition)*. December 14, 2010.

World Wide Web Consortium (W3C, 2016a). *Web Ontology* Language (OWL) w3.org/OWL. Accessed on June 15, 2016.

World Wide Web Consortium (W3C, 2016b), *SKOS Simple Knowledge Organization System*. w3.org/skos. Accessed on June 15, 2016.