Published in: Sprache und Datenverarbeitung vol. 32 (2008) no. 1, pp. 35-46.

Roman Schneider

E-VALBU: Advanced SQL/XML processing of dictionary data using an object-relational XML database

Abstract

Contemporary practical lexicography uses a wide range of advanced technological aids, most prominently database systems for the administration of dictionary content. Since XML has become a de facto standard for the coding of lexicographic articles, integrated markup functionality – such as query, update, or transformation of instances – is of particular importance. Even the multi-channel distribution of dictionary data benefits from powerful XML database services. Exemplified by E-VALBU, the most comprehensive electronic dictionary on German verb valency, we outline an integrated approach for advanced XML storing and processing within an object-relational database, and for a public retrieval frontend using Web Services and AJAX technology.

1 Modelling Dictionary Data with XML and Databases

Modern practical lexicography, i.e. the production of dictionaries, has obviously entered a phase of manifold distribution and re-use of content. It was not so long ago, when lexicographers used computers mainly to produce digital master copies for printing machines. As recently as the World Wide Web introduced appropriate community portals, wikis, grids, and other publication channels, electronic dictionaries have to serve most different media. This evolution comes along with the widespread adoption of two core technologies: XML (Extensible Markup Language) for complex content authoring, and Database Management Systems (DBMS) for content administration. Both can be used together efficiently for the collaborative production of all kinds of lexica. The technological focus lies on the flexible structuring and portability, as well as on reliable high-performance processing, respectively.

As we just pointed out, the rich content of modern dictionary databases has to be made accessible for a variety of distributed applications, regardless of concrete lexicographic categorization – general-purpose or specialized, etymological or terminological, mono-, bi-, or multilingual. In order to demonstrate the advantages of a combined XML/DBMS approach, we will present step by step our solution for transforming a "traditional" print dictionary into a multi-functional, web-enabled resource: Section 2 provides some background information about the origin and micro-structure of VALBU, which is a monolingual valency dictionary on German verbs. Section 3 covers the implementation of a suitable database environment as well as some practical conversion strategies. Section

4 describes typical retrieval showcases and feasible technical strategies, followed by a compact outlook on future development scenarios in section 5.

2 Project Framework

The main objective of the VALBU project at the Mannheim Institute for German Language was the compilation of a monolingual dictionary on German verb valency. Completed in 2004, the print publication contains semantical and syntactical descriptions of 638 lemmata and more than 3.000 verb variants, together with detailed information about morphology, word formation, phraseology, and stylistics. The inventory is based on the requirements for the certificate "German as a Foreign Language" at the federal Goethe-Institut. Today VALBU (Schumacher / Kubczak / Schmidt / de Ruiter 2004) is the most comprehensive dictionary of its kind for German. Since the target audience comprises mainly of scholars and writers whose mother tongue is not German, quite a few examples of usage, based on the analysis of the institute's DEREKO reference corpus – the largest corpus of written German available at present time – are added to each article.

VALBU uses a semantic-oriented valency concept. Broadly speaking, every clause element that is essential to explain the meaning of a verb – or verb variant – is categorized as complement. This results in a maximum valency frame. The verb *vermieten* (engl. *to rent*), for example, opens up to five complement slots:

- Subject complement (the landlord)
- Accusative complement (the rental object)
- Dative complement (the tenant)
- Adverbial complement (the rental fee)
- Second adverbial complement (the term of lease)

For each verb variant, VALBU codes obligatory and facultative complements within the sentence structure (*Satzbauplan*). If obligatory complements can be left out under special circumstances, e.g. with a particular accentuation or by adding context elements, this fact is noted in a separate comment. Furthermore, VALBU contains information about possible realizations for each complement, considering main clause realizations as well as subordinate clause realizations with our without correlate. As regards content, complements are described with their semantic role and category. For this purpose, VALBU does not use deep case terminology, but captures the roles with surface-oriented explications like "the one who rents the object". Additional article sections cover passivation, lists of commonly used supplements, stylistics, and phraseology (construction, idioms etc.).

VALBU articles comprise two different subarticle formats, mainly for didactical reasons. Verb variants, for which the Goethe-certificate demands passive and active command, are described in long format; all other verb variants are discussed in short format. Each article is made up of at least one long subarticle, possibly followed by one or more other long or short subarticles. Within long subarticles, every type of information

is placed separately: semantic roles and categories (person, circumstance, event, etc.), syntactic rules, passivation, comments, examples of usage, etc. Semantic and morpho-syntactic information explicitly refer on each other. Within short subarticles, the semantic categories are placed inside the semantic explanation, which takes the form of a finite-verb clause. Complements are embedded as dummy elements (*something*, *someone*, *somewhere* etc.), so that the semantic role is implicitly recognizable. Information about passivation is not given.

After the print version of VALBU was published in 2004, we noticed a growing demand for an electronic implementation, most preferably as an integrated module for the established web portals GRAMMIS (Schneider 2004) and OWID (Haß 2004). The electronic version (E-VALBU) should extend the original book with new and extended retrieval options (see section 4), and address a correspondingly broader audience. Eventually, the distinction between long and short subarticles should be abandoned in favor of a new integrated format. In a first step, XML was selected as the preferred coding option, and the available VALBU articles are transformed using text technological methods as well as manual post-editing. In order to define a microstructure for the new electronic articles, XML Schema, published as a W3C recommendation in May 2001, came into operation. An XML schema definition allows the precise specification of a set of rules to which an XML document – better: XML instance – must conform. For example, the following code fragment defines the element *sbp* (short for "Satzbauplan", engl. "sentence structure"):

```
<xs:element name="sbp"><xs:complexType><xs:sequence>
<xs:choice>
   <xs:element ref="komp"/><xs:element ref="fak"/>
</xs:choice>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:element ref="op"/>
   <xs:choice>
      <xs:element ref="komp"/><xs:element ref="fak"/>
   </xs:choice>
</xs:sequence>
</xs:sequence></xs:complexType></xs:element>

<xs:element name="fak"><xs:complexType><xs:sequence>
<xs:sequence>
   <xs:element ref="komp"/>
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="op"/>
      <xs:element ref="komp"/>
   </xs:sequence>
</xs:sequence>
</xs:sequence></xs:complexType></xs:element>

<xs:element name="komp"><xs:complexType>
<xs:attribute name="typ" use="required">
<xs:simpleType>
   <xs:restriction base="xs:NMTOKEN">
```

```
            <xs:enumeration value="akk"/>
            <xs:enumeration value="praed"/>
            <xs:enumeration value="verb"/>
            <xs:enumeration value="praep"/>
            <xs:enumeration value="dat"/>
            <xs:enumeration value="adv2"/>
            <xs:enumeration value="adv"/>
            <xs:enumeration value="gen"/>
            <xs:enumeration value="praep2"/>
            <xs:enumeration value="akk2"/>
            <xs:enumeration value="sub"/>
        </xs:restriction>
    </xs:simpleType></xs:attribute>
    </xs:complexType></xs:element>

    <xs:element name="op"><xs:complexType>
    <xs:attribute name="typ" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="Xoder"/>
            <xs:enumeration value="oder"/>
            <xs:enumeration value="und"/>
        </xs:restriction>
    </xs:simpleType></xs:attribute>
    </xs:complexType></xs:element>
```

As we see, *sbp* can be filled with an arbitrary number of *komp* (complement) elements, connected by *op* (operator) elements, and optionally clasped around with *fak* (facultative) markers. Complements are tagged with special attributes (*sub* for subject, *akk* for accusative etc.), and operators can be *und* (and), *oder* (or), and *Xoder* (exclusive or). This fine-grained specification allows the exact representation of possible sentence structures and, most crucial, serves as a foundation for reliable electronic processing and retrieval. A simple example for a sequence of subject complement, accusative complement, and facultative adverbial complement looks like this:

```
<sbp><komp typ="sub"/><op typ="und"/><komp typ="akk"/><op typ="und"/><fak><komp
typ="adv"/></fak></sbp>
```

3 Database Processing of XML Instances

For a considerable time, there is a discussion throughout the scientific community about appropriate and efficient ways of storing XML instances within database management systems. The anticipated benefit lies in the combination of XML's authoring flexibility with the database characteristics: security, data integrity, management of concurrent processes, backup and recovery mechanisms etc. Today most commercial products as well as some open source distributions have integrated XML features into their database engine. For E-VALBU, we decided to stick to the Oracle DBMS already in use for the institute's web information systems and lexicographic portals, so that all data is kept in

one place. Oracle provides on-board tools for all major XML operations, such as automatic generation of XML instances from relational database tables or external files, validation against Document Type Definitions (DTDs) or XML Schema documents, native XML storing, indexing, extraction, manipulation of instance nodes, XSLT transformation etc. Its SQL/XML extensions comply with ISO/IEC standard 9075-14 and allow the

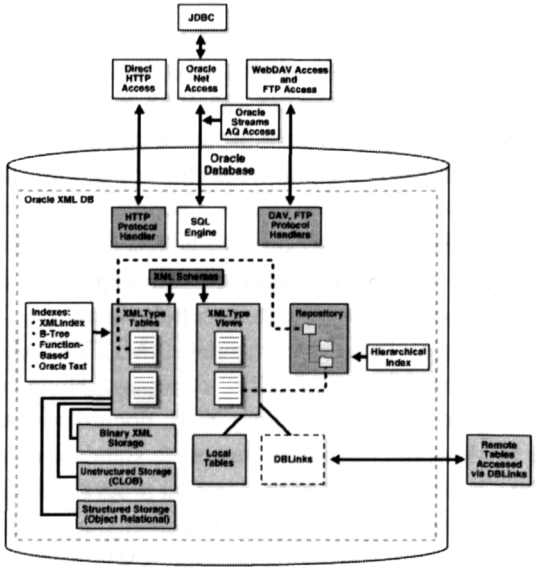

Figure 1: The XML database repository and its protocol handlers

processing of XML instances with SQL (Structured Query Language) commands. An integrated XML database repository (see figure 1) enriches the database with content management features for XML documents, and can be accessed via a variety of protocol handlers such as http, ftp, or WebDAV.

Native XML support within databases demands an appropriate data model for the reliable storing and reconstruction of XML instances. Instances should be treated as logical units independent from the concrete storage technique. Traditional approaches would either use Large Object datatypes (LOBs) or decompose and map XML structures to relational tables. Instead, Oracle works with a special datatype called XMLTYPE for the handling of native XML data. Three different storage options can be distinguished: binary, document centric (unstructured), and object-relational (structured).

The latter calls for a mandatory registration of XML Schema definitions inside the XML database repository. The system then generates proper object types that are used for the fine-grained storing within object-relational tables. Well-formedness and validity are checked automatically; node structure and relations between the nodes remain unchanged (DOM fidelity). This option needs less storage space, because whitespaces, element names, and attribute names are extracted from source code and stored only once as object types. Furthermore, single nodes can be quickly modified with DML statements.

The price for these advantages lies in a somewhat limited flexibility: When modifying the original XML schema, one has to act with the same consideration regarding constraints and datatypes as when modifying a table with an "ALTER TABLE" statement. The document centric option, in contrast, impresses with unlimited flexibility in terms of revising the underlying XML structure, as well as with rapid updating of complete instances. XML documents remain unchanged with exact character match (document fidelity). The binary option, finally, stores XML in a compact binary format. It is XML schema aware, but can also be used without schemas. So its advantage over the object-relational option is that one does not have to know the XML structure in advance, and that even multiple structures can be used for the same database column. It is also more efficient than the document centric option, regarding updating, indexing or selecting XML fragments.

The decision for a specific storage option – for E-VALBU we chose the binary XML format – does not affect any of the following processing steps, i.e. it is totally transparent to the programmer. One can always work with the whole set of SQL/XML commands for manipulating the XML content. In order to populate XMLTYPE-columns, the straightforward way would be to use SQL's "INSERT" command. However, since E-VALBU had to incorporate a large quantity of existing XML documents stored within the file system, the alternate track would be to either resort to standard database tools for bulk loading (SQL*Loader), or to make use of the internal programming language PL/SQL. Initially, this variant expects a so-called SQL directory, i.e. access to a user-defined folder in the server's file system. Herein the external XML instance can be opened and copied into the destination table with the help of the DBMS_LOB package. So populating a rudimental destination table (TB_LEX) that is made up of two columns (CO_ID for the primary key, and CO_XML for the XML instances) can be accomplished with this simple script:

```
CREATE DIRECTORY "xmldir" AS 'C:\VALBUFILES';
DECLARE
  v_file BFILE := BFILENAME('xmldir', 'abfahren.xml');
  v_content CLOB := ,';
BEGIN
  DBMS_LOB.fileOpen (v_file, DBMS_LOB.file_readonly);
  DBMS_LOB.CREATETEMPORARY(v_content, TRUE, 2);
  DBMS_LOB.loadFromFile (v_content, v_file,
  DBMS_LOB.getLength(v_file), 1, 1);
  DBMS_LOB.fileClose (v_file);
  INSERT INTO tb_lex VALUES (1, XMLTYPE(v_content));
  COMMIT
END;
```

Further processing, e.g. the extraction of XML nodes, the computation of values, the creation of task-specific views of XML fragments etc., can optionally be placed within INSERT or UPDATE triggers.

A bit more challenging is the task of assembling XML fragments by querying remote or local database tables. For this purpose, SQL/XML offers a set of specialized functions that deliver XMLTYPE content. To name a few (see also Scardina / Chang / Wang 2004): XMLELEMENT creates a single XML element, and expects element name and optionally element content as argument parameters. XMLFOREST converts each of its argument parameters to XML, and then returns an XML fragment that is the concatenation (a so-called "forest") of these converted arguments. XMLCONCAT takes a series of XMLTYPE instances as input, concatenates the series of elements for each row, and returns the concatenated series. XMLAGG, finally, is an aggregate function that takes a collection of XML fragments and returns an aggregated instance. With these tools, one can collect, sort, and format any relational data in order to build new XML instances or to add value to existing documents. Let us assume that some information about verb inflection is originally placed within two separate tables, connect by the canonical form as unique identifier, and – for the sake of exemplification – not normalized:

TB_STAMMFORM		
CO_LEMMA	**CO_FORM**	**CO_NAME**
abfahren	fährt ab	Indikativ
abfahren	fuhr ab	Präteritum
abfahren	ist abgefahren	Partizip Perfekt

TB_KONJUGATION	
CO_LEMMA	**CO_MUSTER**
abfahren	stark

The following SQL/XML statement would build a suitable XML fragment:

```
SELECT
  XMLELEMENT ("eintrag",
  XMLFOREST(t2.co_lemma as "lemma", t2.co_muster as
  "konjugationsmuster"),
  XMLELEMENT ("stammformen",
  XMLAGG(
  XMLCONCAT(
  XMLELEMENT("formname",t1.co_name),
  XMLELEMENT("form",t1.co_form)))))
FROM tb_stammform t1, tb_konjugation t2
WHERE t1.co_lemma=t2.co_lemma;
```

This statement produces valid XML code that can be inserted directly into our dictionary table's column CO_XML:

```
<eintrag>
<lemma>abfahren</lemma>
<konjugationsmuster>stark</konjugationsmuster>
<stammformen>
<formname>Indikativ</formname><form>fährt ab</form>
<formname>Präteritum</formname><form>fuhr ab</form>
<formname>Partizip Perfekt</formname><form>ist abgefahren</form>
</stammformen>
</eintrag>
```

Lexicographic XML content – as well as any content in continuous projects - remains almost never static. As project research attain new perceptions, one may have to update the value of certain elements or attributes. Sometimes even the customization of the internal XML structure will be inevitable. SQL/XML offers dedicated functions that manipulate XML nodes. The most prominent is certainly UpdateXML(), other DOM-related operations can be conducted by the self-explanatory functions InsertXMLBefore(), InsertChildXML(), AppendChildXML(), and DeleteXML(). Transformation of whole instances with XSLT technology can be accomplished with XMLTransform, as we will see in the next section.

4 Retrieval with SQL and XPath

Once the E-VALBU articles are stored as native XML instances, efficient retrieval ranks high on the agenda. Besides simple lemma listings and the display of complete articles, users are mostly interested in the extraction of single XML node values. For this purpose, the commonly accepted XPath (XML Path Language) technique comes into play. XPath operates on the logical structure of XML instances and treats them as hierarchical tree (Document Object Model, DOM). An XPath expression consists of axis specifiers and/ or node tests and/or predicate notations. For example, given the above XML fragment, the following expression selects the second "formname" element that is a child of a "stammformen" element directly underneath the "eintrag" element:

```
/child::eintrag/child::stammformen/child::formname [position()=2]
```

or (abbreviated):

```
/eintrag/stammformen/formname[2]
```

Furthermore, a comprehensive set of axis specifiers like "ancestor", "descendant", "following-sibling" etc. can be integrated into an XPath expression for both powerful and accurate access to XML fragments. XPath retrieval is implemented within the database engine in the shape of functions like existsNode(), extract(), and extractValue(). As its name indicates, existsNode() finds out whether the specified node exists or not, and gives back 1 or 0, respectively. Extract() extracts XML fragments as XMLTYPE (e.g.

<formname>Präteritum</formname>), whereas the output of extractValue() is the pure content of an element or attribute node (e.g. "Präteritum").

Irrespective of whether an XML instance actually contains the requested node, these functions construct a DOM tree for all processed documents. In order to improve the speed of SQL/XML operations such as `SELECT CO_ID FROM TB_LEX t WHERE extractValue(t.CO_XML,'/eintrag/stammformen/formname') = 'Präteritum'`, the creation of database indexes is highly recommend. Good choices for frequently used expressions are function-based B-tree indexes that are similar to traditional B-tree data structures in the sense that they work with balanced index trees. Oracle allows multiple parallel function-based indexes for one XMLTYPE column. As the case arises, the built-in cost-based optimizer decides which index fits best for the interpretation of a specific WHERE clause. Alternatively, one can use the specialized index type "XMLIndex". It automatically creates shadow tables for XPath expressions, row IDs, node content, and hierarchical position information like parent-child, ancestor-descendant, and sibling relations. Range queries as well as wildcard search are supported. XMLIndexes are best-suited for structured XML documents where it is hard to predict which ad hoc requests will be carried out in the future. If the project setting allows the restriction of search space, smaller indexes – and therefore faster index operations – can be achieved with XPath subsetting. For example, the following statement creates an XMLIndex that only includes the two nodes explicitly specified as XPath parameters: `CREATE INDEX idx_foo ON TB_LEX(CO_XML) INDEXTYPE IS XDB.XML_INDEX PARAMETERS ('PATHS (INCLUDE (/eintrag/lemma) (//formname))')`.

Additionally, it should be noted that database-oriented XML retrieval not necessarily has to use SQL/XML features. Project scenarios will often demand for a combination of XML queries with advanced full text search. This is the domain of Oracle's Text component and its highly complex CONTEXT index type. It focuses on multi-format text retrieval, offering a variety of strategies including keyword search, theme-based search, linguistic analysis etc. For the integration of E-VALBU into the institute's web information systems and lexicographic portals, we adopted the CONTEXT index type that covers our most prominent retrieval needs:

- Content filtering: A built-in filter recognizes different document formats like plain text, XML, HTML, PDF, Office formats etc.
- Section processing: Customizable section groups map XML elements and attributes to addressable sections.
- Word processing: A lexer module separates the sectioner's output into words or tokens. It deals with compound words, alternate spellings, skipjoins etc. Besides, stopword lists and stemming preferences can be defined.
- Semantic retrieval: Integration of thesauri or other semantic technologies for the retrieval of synonyms, hyponyms, meronyms etc. We used this feature to connect E-VALBU with our domain specific ontology of German grammar, thereby redu-

cing lexicographic information costs (i.e. possible difficulties and inconveniences of dictionary users) regarding terminology, abbreviations etc.
- XPath retrieval: Large-scale integration of XPath syntax into full text queries.

Generally, the CONTEXT index type turns out to be beneficial for all retrieval situations that do not use the complete node value as search criterion. This category comprises substring or prefix search, inflectional search, or fuzzy search; a detailed discussion would extend the scope of this paper. Even regarding retrieval speed of XPath expressions, we found out that our CONTEXT index implementation performs exceptionally well in comparison with the specialized XMLIndex. In order to illustrate some XML functionality of CONTEXT-based indexes, here comes a list of advanced queries:

- `SELECT CO_ID from TB_LEX t WHERE CONTAINS (t.CO_XML, 'Pertinenzdativ WITHIN anmerkung')>0` finds all XML instances that contain the word "Pertinenzdativ" within the XML section group "anmerkung".
- `SELECT CO_ID from TB_LEX t WHERE CONTAINS (t.CO_XML, 'Pertinenzdativ inPath (//anmerkung)')>0` is an alternate variant explicitly using the appropriate XPath expression.
- `SELECT CO_ID from TB_LEX t WHERE CONTAINS (t.CO_XML, 'hasPath (//sbp/komp[@typ=''sub''])')>0` tests for the existence of an XML element "komp" which is a child of "sbp" and possesses a "typ" attribute with the value "sub".

The integration of XML-based database queries into E-VALBU's public retrieval frontend is achieved by implementing dedicated Web Services and – on the client side – AJAX (Asynchronous JavaScript and XML) scripts that manipulate the DOM tree of the HTML search page. Figure 2 provides an insight into our practical approach. As soon as specific complement types are selected, the browser sends a first request to the database server and retrieves a list of matching sentence structures in a customized XML format. This list is then embedded into the corresponding HTML drop-down list without the need to reload the whole page. After the search button is pressed, again a web service is called that takes an XPath expression (e.g. //sbp/komp[@typ="sub"]) as input parameter and delivers an XML fragment containing ID and lemma of matching E-VALBU instances. This list is processed by another AJAX script, and displayed with the dictionary entries as hypertext anchors. Finally, a click on a lemma hotword calls a server-side procedure that transforms the appropriate sections of the selected XML instance into HTML, and sends the output to a local DOM container at the bottom of the search page.

As already mentioned, for the transformation from one markup language into another – e.g. from VALBU's tailor-made XML format into HTML – XSLT is the commonly preferred technique. The Oracle database offers an integrated XSLT processor that is activated with the XMLTransform() function. It works in a virtual XSLT machine and compiles XSLT stylesheets into byte code, stored within an internal cache. This signi-

ficantly speeds up repeated requests with the same transformational setting. For the E-VALBU frontend, we developed two separate stylesheets. The first one produces a condensed synopsis of the article, as seen in figure 2. Besides general information like pronunciation or conjugation paradigm, this overview presents the available verb variants as numbered list. A click on the corresponding hypertext anchor calls a second stylesheet that extracts the complete XML fragment for the verb variant, converts the content to HTML, and replaces the original overview with an elaborated presentation.

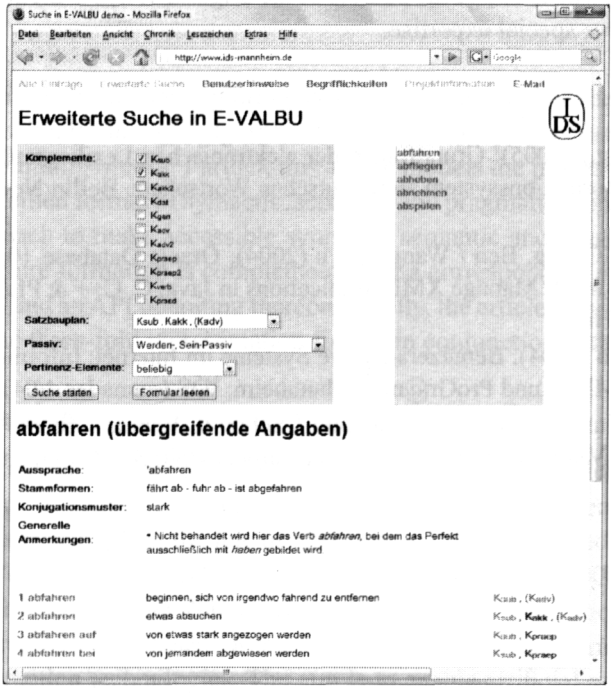

Figure 2: Example retrieval frontend to VALBU using Web Services and AJAX

5 Conclusion and Outlook

All data in one place: Our implementation presents an integrated framework of tools and techniques that serve as foundation for the database-driven processing of lexicographic content. Based on the assumption that both collaborative authoring as well as dictionary aggregation will continuously gain in importance, we see various applications for this approach. Web Service-based integration of XML exchange formats into lexicographic portals and web information systems will be most likely a key feature of these solutions. Given the fact that established database management systems, like Oracle or its competitors from IBM and Microsoft, by now offer free licenses for academic research, they establish a reliable and affordable alternative to their open source counterparts. Our future work will focus on elaborating the present approach, especially by incorporating

distributed dictionary content, and by developing more sophisticated retrieval interfaces that allow the software-independent linking of dictionary resources to heterogeneous linguistic portals.

Acknowledgment

I offer sincere thanks to Jaqueline Kubczak from the Mannheim Institute for German Language (IDS), who is one of the authors of the original VALBU dictionary, for patient assistance with her special expertise.

References

Haß, Ulrike (Hrsg.) (2005): Grundfragen der elektronischen Lexikographie. *elexiko* - das Online-Informationssystem zum deutschen Wortschatz. Berlin/New York: de Gruyter.

Scardina, Mark / Chang, Ben / Wang, Jinyu (2004): Oracle Database 10g. XML & SQL: Design, Build & Manage XML Applications in Java, C, C++ & PL/SQL. New York: McGraw-Hill.

Schneider, Roman (2004): Benutzeradaptive Systeme im Internet: Informieren und Lernen mit GRAMMIS und ProGr@mm. Mannheim: IDS (=amades 4/04).

Schumacher, Helmut / Kubczak, Jacqueline / Schmidt, Renate / de Ruiter, Vera (2004): VALBU - Valenzwörterbuch deutscher Verben. 1040 S. - Tübingen: Narr, 2004