EMELD Conference July 2004, Detroit 

Archiving Language Resource Objects in XML

Dafydd Gibbon, Thorsten Trippel & Ben Hell
Universität Bielefeld
Version 03

Contents

1 Archiving in XML **2**

2 Language Resources **2**

3 Getting abstract: types of Resource Object **4**

4 Abstract Resource Object implementation in XML **5**

5 Getting practical: the MoDeLex application **7**

6 Using an XML database: TAMINO **9**
6.1 Procedure 9
6.2 Database creation 9
6.3 Corpus data stored in the file system 10
6.4 Using a DBMS for storing Resource Objects 10
6.5 Selected use cases 12
6.6 Querying 13
6.7 Signal processing 14

7 Conclusion: evaluation and further work **14**

A Sample XQuery **16**

B Sample TASX formatted corpus **17**

1 Archiving in XML

The object of this contribution is to discuss a Database Management System (DBMS) selection and database design strategy which was developed for archiving XML annotated language resources in a number of research projects at Universität Bielefeld. The research projects work on the one hand with data which were collated and annotated from the start according to standard specifications of reusability, interoperability and interpretability and on the other hand with legacy data which were re-processed according to these criteria [Gibbon et al., 2004b]. The criteria are being used in the MoDeLex project on *Theory and Design of Multimodal Lexica*[1], to specify resources in the WALA metadata repository for West African Languages [Gibbon et al., 2004a], and in developing an M.A. curriculum in language documentation for West African languages[2].

Specifically, we focus on the criteria for selection and technical realisation of the TAMINO[3] XML oriented DBMS. As a commercial system, TAMINO does not fall into the Open Source category which would be appropriate for an Open Language Archive; however, for research purposes it is necessary to investigate the potential of such a system. An Open Source alternative is currently under development with the XINDICE application by the Apache Software Foundation[4].

For the purpose of discussing resource database design we discuss linguistic and formal aspects of resource description, and for this purpose introduce a distinction between four types of Resource Object:

1. General Resource Object (language data types specified in terms of standard metadata).
2. Specific Resource Object (actual data instantiation as a General Resource Object).
3. Abstract Resource Object (formal abstract data type specified in standard informatic terms).
4. Implementational Resource Object (concrete data types specified in data representation languages, knowledge representation languages and programming languages).

These distinctions are discussed in detail below.

2 Language Resources

Language resources are standardly classified along many metadata dimensions pertaining to the technical and interpersonal setting of their production, their physical and abstract formats, and the linguistic and situational categories of their content. Two proposals for such dimensions which have been optimised for linguistic information harvesting are are IMDI (Isle MetaData Initiative), and OLAC (Open Language Archive Community).[5]

In the present discussion we restrict our attention mainly to file sets containing information of the following types:

- written texts and spoken dialogue transcriptions,

[1] Funded by Deutsche Forschungsgemeinschaft.
[2] Funded by Deutscher Akademischer Austauschdienst.
[3] Produced by Software AG.
[4] see http://xml.apache.org/xindice/index.html
[5] Current versions of these proposals are easily found on the web.

- annotations:
  - time-stamped transcriptions of spoken dialogue,
  - marked up written text (primary text or transcription),
- signal recordings:
  - audio,
  - video,
  - laryngograph,
  - airflow,
- lexical information:
  - concordance (KWIC or annotation-based),
  - tabular lexicon with classic
    1. *macrostructure* as a list of entries,
    2. *microstructure* of entries with flat or shallow hierarchical arrangement of types of lexical information (data categories),
    3. *mesostructure* linking entries with sketch grammar information, cross-references and examples (the latter also as a concordance),
  - hierarchical (e.g. type or default inheritance) lexicon.

Annotated grammars and other linguistic documents presuppose more fundamental agreement on the linguistic category systems and text structures than is currently available, and will not be considered here.

The extensive resources created in various projects require efficient storage, management and procedures with the following features:

- structuring with XML,
- storing,
- accessing,
- updating:
  - monotonic changes (addition and versioning),
  - nonmonotonic changes (non-recoverable deletion),
- re-structuring.

The XML structure caters for varying types of metadata as well as internationalization issues such as the encoding of language features; character encoding uses Unicode.

For the particular application dealt with in the project *Theory and Design of multimodal Lexica* (MoDeLex) the goal is to integrate lexical information from different modalities in a lexical resource. This includes concordancing of these corpora, i.e. finding all occurrences of signs in various contexts in all modalities

(gesture and speech) and speech submodalities (locution and prosody). One central problem is that coarticulation across tiers prevents simple separation of the contexts from the sign. Another problem is that there are problems of unique definition and ambiguity of units. Further, indexing of multimodal data (whether manual or semi-automatic) is time consuming and error prone. Hence automatic signal concordancing is needed as far as possible.

In Section 3 we describe four different perspectives on Resource Objects before discussing implementation in XML in Section 4. Section 5 illustrates project requirements for the spoken language annotation project MoDeLex and in Section 6 the experimental use of an DBMS for XML databases, TAMINO, is discussed. In the conclusion in Section 7 we discuss evaluation and open issues. Two appendices are included, Appendix A, with a sample XQuery, and Appendix B, with a sample TASX formatted corpus, are included.

First, we will distinguish different levels of the structure of language resources and consider general issues of data representation. then briefly discuss the structure of typical language resource databases, followed by a report on our own work with the TAMINO XML DBMS and a conclusion.

3 Getting abstract: types of Resource Object

In order to reduce potential sources of confusion, we distinguish between *General Resource Objects* and *Specific Resource Objects* on the one hand, and *Abstract Resource Objects* and *Implementational Resource Objects* on the other.

General Resource Objects: resource object *types*, such as transcriptions, annotations, concordances, dictionaries.

Specific Resource Objects: specific *instances* of general resource objects.

Abstract Resource Objects: *abstract data types* which are appropriate for representing general resource objects, such as lists, tables, trees, networks (together with methods — algorithms — for processing these objects).

Implementational Resource Objects: *specific data types* in representation and programming languages which are appropriate for computing with abstract resource objects, such as strings, arrays, linked lists and other pointer structures (together with methods — procedures — for processing these objects).

Much relevant discussion of General Resource objects is found, for instance, in the Annotation Graph literature [Bird and Liberman, 2001].

We concentrate here on Abstract Resource Objects which are commonly used for representing the structure of language resources, for example:

1. strings (in general of arbitrary length),
2. sequences of strings (in general of arbitrary length),
3. structures defined over strings, typically: labelled trees (of arbitrary depth).
4. lists (often of arbitrary length),

5. tables (often of arbitrary depth, particularly in sketch grammars),
6. directed acyclic graphs, DAGs (of arbitrary depth),
7. cyclic graphs, CGs (of arbitrary depth)
8. numbers (integer and real).

The objects which are of particular interest for present discussion are trees, tables and directed acyclic graphs.

Example: trees. Trees are typically used in linguistics to represent hierarchical relations of two kinds:

1. paradigmatic relations (relations of classification, similarity), such as taxonomies, as in thesaurus organisation,
2. syntagmatic relations (relations of compositionality), such as constituent structures or dependency trees.

Example: tables. Tables, both flat and embedded, are typically used in linguistics to represent distinctive feature matrices and vectors, attribute-value structures, morphological generalisations (paradigm tables) and lexica.

Example: Directed Acyclic Graphs (DAGs). DAGs are used in linguistics to represent structures of all kinds, including trees and tables, but in particular structures which are not trees or tables, such as autosegmental lattices (tiers of partially linked 'autosegments') and coarticulated gestural patterns of the kinds found in the tiers of annotated speech signals.

4 Abstract Resource Object implementation in XML

For the storage of language resources, representations of General Resource Objects need to be specified in terms of Abstract Resource Objects, which are in turn realised by Implementational Resource Objects. The view on Abstract Resource Objects taken here is one which is common in Computational Linguistics.

The most popular implementation language for realising Abstract Resource Objects in the language resource field is currently XML. XML is frequently also used as an abstract specification language, but for this purpose it is not ideal as its formal semantics is not particularly clear. The markup language paradigm represented by XML embodies basic database notions such as *entities* and *relationships*:[6]

1. In the case of XML the *entities* are triples of
   (a) *names* (of *types*),
   (b) *attribute-value structures*,
   (c) *strings*.

[6]Note that the terms *Entity* and *Relationship* are database terminology (Section 6.4), and differ from the terminology of the XML paradigm.

2. In the case of XML the main *relationships* constitute tree structures and are defined syntactically by context-free grammars (DTDs, Document Type Descriptions); further relationships, for instance between documents or discontinuous parts of documents, are defined semantically by means of pointer structures.

Unlike traditional networks, which correspond in the main to DAGs, the relationships defined by DTDs may be recursive (as in the sentences of natural language), thus corresponding to Cyclic Graphs (CGs). The structure of a given (finite) document as defined by a DTD may always be represented by a DAG, in fact by a tree. Enhancements with pointer structures may require CGs.

A strict distinction between the tree structure of a *formal syntax* and the *semantics of pointer structures* must consequently be maintained.

If a DTD defines a non-recursive tree structure, i.e. a tree structure with finite depth, then this structure may in principle reduced to a regular grammar (finite-state transition network, FSN, FTN), and processed by a finite state automaton (FSA). Note, though, that tables of arbitrary depth (such as the paradigm tables of sketch grammars) are more complex than tree structures. Tree structures may be derived with grammars which describe Type 2 (context–free) formal languages with the structure

$$a^n \ b^n$$

A moment's reflection will show that an embedded table needs a sub-type of context-sensitive grammar, an indexed grammar, which describes Type 1 (context–sensitive) languages with the structure

$$a^n \ b^n \ ... \ c^n$$

where each $a, b, c, ...$ represents a row, and each exponent n represents the length of the row, which is, in the general case, equal to the length of the other rows. This kind of context–sensitive language is called an *indexed language*.

The discussion thus suggests a computationally useful scale of complexity in the development of data models for resources, analogous to the Chomsky hierarchy of formal grammars:

1. Type 0. Arbitrary networks.
2. Type 1. Context sensitive structures such as embedded tables.
3. Type 2. Tree structures such as named entities with attribute-value structures.
4. Type 3. Flat (possibly iterative) sequences.

This discussion has consequences for the use of XML. Taken as a context-free representation language, the formal *syntax* of XML presents no problems in representing trees of all kinds. The *compositional semantics* of trees — as paradigmatic classification relations, syntagmatic compositional relations or other hierarchical relations — is rather straightforward. But there are more complex structures which require additional specifications which are external to the syntax of XML. Tables and directed acyclic graphs of the autosegmental lattice (or annotation graph) type are examples of structures which go beyond tree complexity.

First, the *syntax of embedded tables* cannot, in the general case, be fully represented in XML, since each XML DTD defines a context-free language, and embedded tables (as already shown) have to be described as non-context-free indexed languages. Consequently, in order to represent embedded tables in XML, *additional syntactic constraints* which recursively specify equality of row branching (equal fan-out factor) are required.

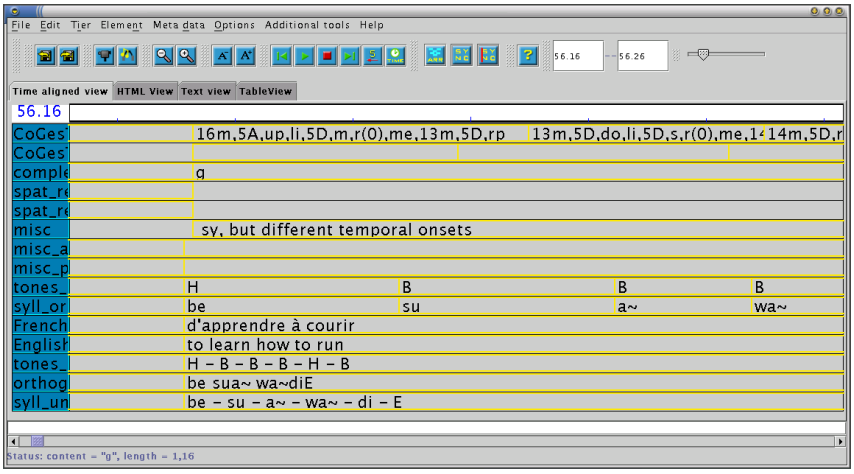

Figure 1: Multitier score labelled with TASX-annotator

Second, the *syntax of DAGs*, lattices, networks, etc. cannot be represented by XML syntax in the general case, since nothing more complex than trees can be derived by context-free grammars. In order to cater for the joining of edges in addition to the divergence of edges a *semantic approach* is required here, in which *compositional tree semantics* is enhanced by *pointer semantics*.

More complex devices — so called 'hedge grammars' or 'regular tree grammars' - are in fact being developed to cope with more complex structures. For practical purposes, these distinctions may seem super-subtle. However, for designing formal data models and methods for handling them they need to be recognised.

In practice, an XML implementation (and thus an XML oriented DBMS) will need to cope with these additional constraints and with non-tree-based semantic interpretations. This by no means an insuperable problem, but it is certainly an issue which needs to be made more explicit for data modelling purposes than has previously been the case. Specifically, implementations of DBMS methods need to be tested explicitly for their ability to handle these issues.

5 Getting practical: the MoDeLex application

The requirements for database storage in the MoDeLex project derive from the underlying issue of theory and design of multimodal lexica. Both textual data with complex structures and video and audio signal data have to be stored and accessed. For multimodal data this means in particular

- access to time aligned annotations containing
- a set of tiers, each with
- time-stamped labelled intervals (segments).

Figure 1 shows a screenshot of an annotation in the form of a multitier score with 15 tiers labelled at different linguistic annotation levels, created with the TASX-annotator [TASX-Annotator, 2004]. The levels of annotation are:

1. right–hand gesture (in CoGesT notation, [Gibbon et al., 2003]),
2. left–hand gesture (if different),
3. complex gesture (if relevant),
4. spatial relation 1 (if relevant),
5. spatial relation 2 (if relevant),
6. Comment,
7. Comment (annotator Alex),
8. Comment (annotator Karin),
9. Lexical and morphological tones,
10. syllable level segmentation orthographic segments,
11. French gloss,
12. English gloss,
13. phrase level segmentation tone sequence,
14. phrase level segmentation orthography,
15. phrase level segmentation syllable sequence.

Some tiers are hierarchically related, others represent different modalities (e.g. gesture vs. the oral modality) and submodalities (e.g. locution vs. prosody). The TASX storage format (*Time Aligned Signal-data eXchange format*, [Milde and Gut, 2002]) is defined in XML as one possible binding (implementation) of the Annotation Graph model [Bird and Liberman, 2001].[7]

As an Abstract Resource Object, the format is defined straightforwardly as follows:

```
<Annotation>   ::=  <Session> | <Session> <Annotation>
<Session>      ::=  <Tier> | <Tier> <Session>
<Tier>         ::=  TierName | <TierBody>
<TierBody>     ::=  <Segment> | <Segment> <TierBody>
<Segment>      ::=  Start End Label
```

`TierName` and `Label` are strings, `Start` and `End` are real numbers in the annotation time line.

In other words, the data model is segmented into sessions, which the MoMoLeX case are

1. recording sessions,
2. tiers, and
3. individual segments containing
4. annotations, which consist of labels and a pair of time-stamps.

[7]We thank Sandrine Adouakou for providing the Agni (triliteral SIL code ANY) data.

6 Using an XML database: TAMINO

6.1 Procedure

The procedures of resource storage and use can be divided into four different steps:

1. corpus creation,
2. database creation,
3. querying,
4. signal selection for further analysis.

The annotations are created using annotation software, such as Praat, esps-waves+ (xwaves) for audio and the TASX–annotator for video. Unlike the other tools, the TASX annotator also allows the insertion of arbitrary Unicode characters (by selecting a particular character from a font table).

All of these tools are based on different data formats, so the first step in further processing is the normalization of the data, i.e. all annotations have to be converted into a single format. To allow the preservation of all information from the original data formats, including metadata, a relatively open approach was taken, taking into account the information on metadata, timestamps and technical information. The only format allowing for this was the *Time Aligned Signal data eXchange format* (TASX)[8], around which the TASX annotator was designed.

This open data format has a number of advantages:

- only time alignment is presupposed
- metadata can be recorded in AV format
- it is XML
- restrictions for annotations can be encoded using embedded trees
- hierarchical annotations can be embedded using namespaces and include this in the data format.

The TASX DTD is converted into XSchema. Using namespace definitions the content of the annotations can be validated against other document grammars than the annotation format grammar. Hence a syntactic validation is possible using all options of the XML encoding family, such as data type definition (e.g. dates, strings, but also extended, user defined data types). These features are also used for querying.

6.2 Database creation

For the creation of the database itself, two different options exist:

1. creating, storing and accessing the database on a file system, with *ad hoc* access tools,

[8] `http://tasxforce.lili.uni-bielefeld.de/`

2. storing the database in a *DataBase Management System* (DBMS)

It is possible to use a mixture of both approaches, such as storing the annotation data in a DBMS and storing the signal on a file system. For the time being we decided to use this hybrid approach in the initial proof-of-concept prototype, using the file system for signal files (because DBMS server extension applications for handling non–native formats were not available) and the DBMS for XML files. Future versions will integrate signal files into the DBMS.

6.3 Corpus data stored in the file system

Storing XML files in a file system such as a web server seems to be the most obvious way of storing the data. This method allows access to the data which can be either via the command line, which enables easy manipulation such as changing and updating, or via a suitable graphical user interface form (GUI form).

This proceedure, however convenient it might seem at first glance, is not at all convenient if the negative consequences are taken into account. One of these consequences is that the consistency of the data cannot be guaranteed, because manual interference is possible even if initially a validating parser can be used for checking the data. Some operating systems offer an access management to the filesystems which defines access restrictions to individual users and user groups. However, manual influence is highly problematic. Storing the material in this fashion requires considerable self discipline for everyone who has access to the data, which still, of course, does not guarantee consistency.

Another problem is performance, especially when querying large data repositories. Reading and searching large repositories on the file system requires full access to all files and all structures.

For signal data, however, initially the use of a file system as a storage structure does not seem to be so much of a problem. Modification of the data requires specialized tools; otherwise the danger of random changes is present, as the signal data is stored in binary formats which are not meant for direct eyeballing and are extremely hard to manipulate manually in any sensible way. We limit the consistency problem artificially to the question of file naming and persistence in a given directory. Nevertheless, the optimal solution for signal data is also ultimately provided by a DBMS.

Backup and restore options for file system storage techniques are provided by the conventional file system functions.

6.4 Using a DBMS for storing Resource Objects

A Database Management System allows efficient storage and access to a database. The definition of the databases allows preservation of the consistency of data, as updates and re-structuring are only permitted within the constraints of the DBMS. User–friendly Graphical User Interfaces (with 'forms' and 'filters') are provided in order to facilitate the manipulation of what might be quite complicated data.

A typical DBMS allows indexing of specified data types to optimize access to the data, and provides backup and restore mechanisms as well as database mirroring tools, access restriction, space restriction, system power restrictions, timeouts, etc.

The purpose of a DBMS is to provide a software environment for

- systematically defining and manipulating data in a formal data model,

- permitting safe storage of large quantities of data,
- with application independent storage,
- and permitting controlled access for maintenance and general use.

Data models are described in terms of *entities* and *relationships* between the entities which define the structure of the database (see also [Draxler, 1997]). The relationships may be

- *hierarchical*,
- *network*,
- *relational* (based on relation algebra using set theoretic *relations*, set operators *union*, *intersection*, and *difference*, relation operators *selection* for selecting rows in a table, *projection* for filtering attributes or columns in a table and *join* (for merging tables),
- *object-oriented*, containing class hierarchies as in a lexicon type or default hierarchy, taxonomy or 'ontology',
- *deductive*, as in a Prolog fact and rule base or in an expert system.

The standard kind of data model is relational. Hierarchical and network structures have for a long time been considered to be 'deprecated' ways of structuring data. However, this view has been overtaken by storage systems with arbitrarily structured and arbitrarily related objects in storage environments such as the World Wide Web, for which provision has to be made. XML, for instance, provides a hierarchical data model and, when enriched semantically with pointer structures, XML provides a network data model.

In Computational Linguistics and Artificial Intelligence more sophisticated object–oriented and deductive structures are preferred, since they provide a more structured concept of compositionality and permit the expression of scientifically valued 'paradigmatic' notions such as

- generalisation,
- underspecification,
- default,
- redundancy, and
- simplicity.

Object–oriented and deductive data models are not supported by the XML paradigm, though this defect is being remedied [Trippel et al., 2004].

There are thus different options for databases. However, the use of standard relational databases for semi–structured XML data (or simply: non–tree–structured data) seems not to be the best choice. Another option is the use of native XML databases. Such a Database is available with TAMINO (Software AG). Figure 2 shows the architecture of the TAMINO system as used for handling concordance queries.

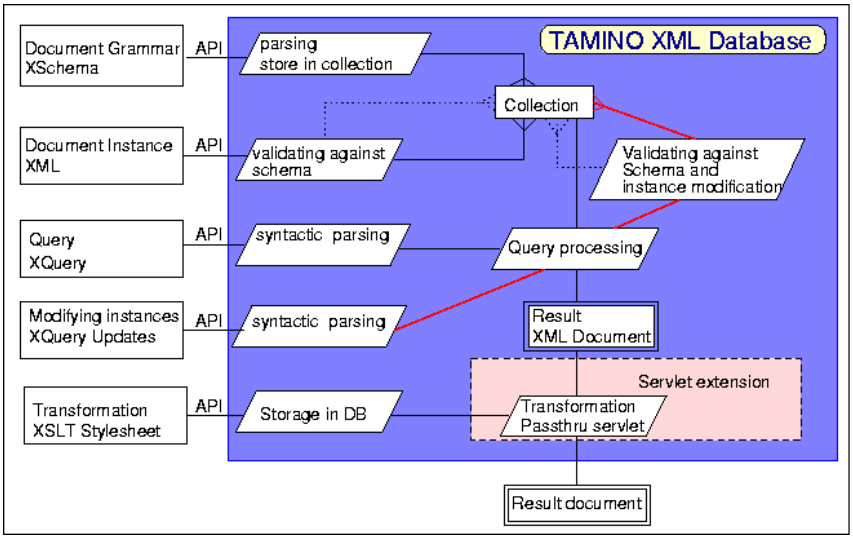

Figure 2: Architecture of the Tamino XML DBMS

6.5 Selected use cases

Creation: To store data in the database there are two options for the creation of data collections, which can either be binary, including binary large objects such as signal data, or textual XML data. To store XML data, the following information is required:

1. required Schema or Schemata (Schemas) as document grammars,
2. document instances.

On storing the data the XML files are validated against the document grammars.

Maintenance: The processing of the signal stored within the database can be done with database server extensions, using a predefined API. For this purpose we have provided a Perl API with CGI-based maintenance and access in mind [Hell et al., 2003]. The maintenance procedure for the XML data uses the new W3C standard XQuery language [Boag et al., 2003], in contrast to the SQL standard for relational databases, or to *ad hoc* scripting. TAMINO allows a subset of XQuery queries on XML data for updating, inserting and deleting databases. After processing, the document is validated again as a correct XML document before it is finally updated in the database, in order to ensure formal consistency.

Querying: Querying can be quite complex, and range from metadata driven to text searching procedures. An important concept, especially useful for concordancing, is granularity specification for corpus access, which can be compared to zooming into a picture. By zooming into the corpus detailed parts of the corpus up to the finest represented structures become available, and zooming out allow for a broader view seeing the wider context. This requires detailed description of and access to metadata at all levels [Trippel, 2004]. Querying is described in more detail below.

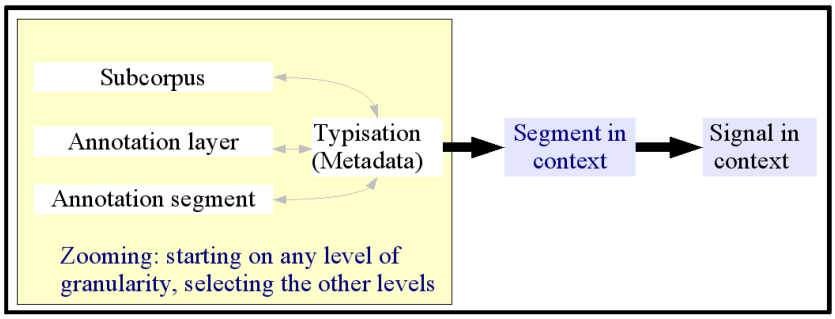

Figure 3: Concordancing a multimodal corpus with score annotation

6.6 Querying

For general querying purposes, full IMDI specified metadata are included; however, we restrict ourselves here to concordance–type querying. The data selection procedure starts with the identification of the subcorpus and the identification of the tier, both based on the metadata available inside the corpora. The annotation segment can be selected according to metadata or by its content. Different access strategies are possible as well, for example selecting the subcorpus that contains a particular word. Figure 3 illustrates the identification of a segment in context.

The process of the selection of the annotation segment results in a partial corpus which can be transformed into a required format.

For the query specification and execution procedure XQuery is the format of choice rather than *ad hoc* processing with other programming languages. The output of a query in XQuery is always one or many well formed XML documents. The XML document is transformed for the user inside the DBMS with XSLT, using a server extension such as the saxon XSLT processor[9]. XQuery allows the use of all document grammars available in the database. Namespaces can be used for accessing other embedded structures.

A sample XQuery selecting the contexts of a word is shown in Appendix A. A sample corpus fragment can be found in Appendix B. Table 1 describes the XQuery sample in an 'almost English' pseudo-code style.

This pseudo code hides the major workhorses of XQuery, which are the so called FLWOR expressions[10]:

For a number of elements, which is an iteration

Let some variables be bound to a certain value

Where some conditions apply (a kind of filter)

Order by some criterion (sorting of the return sequence)

Return a sequence of characters, strings, or elements

Like the XML transformation language XSLT and languages such as LISP and Scheme, XQuery is a

[9] http://www.saxonica.com/
[10] XQuery buffs pronounce this like the English word *flower*.

Table 1: Description of an XQuery query for a specific corpus (see Appendix A).

Within a new `tasx` element iterate over all `session` elements and within these over all `meta/desc` elements

where the session metadata AV pairs have a particular value (i.e. it has to have a `SessionTitle` name and `Anyi_Story_17` value pair)

return a sequence of a `session` element with the following content:

- iterate over all `tier` elements in these sessions
- without further restrictions
- return a sequence with the following content:
  - iterate over all `event` elements in these tier element
  - where the text of this element is equal to the search word
  - return a sequence with the following content:
    - iterate over all tiers that are child elements of the 'grandparents' of the found search word
    - without any further selection
    - return a `tier` element with the following content:
      - iterate over the `event` elements in these tier
      - where the `start` and `end` attributes of these have a certain value (i.e. if they are in a certain interval of the`start` and `end` attribute values of the found searchword)
      - return a sequency of these events.

functional programming language. A Perl API is available for using XSLT from Perl CGI-script interfaces for web GUI applications.

6.7 Signal processing

Files which do not have an XML format, the native format of the TAMINO database, require APIs for access and processing. We have implemented a set of Perl programms for this purpose. This applies particularly to binary audio (or video) files, for which specialised tools are needed for access and processing. The main operation required is the creation of audio clips by cutting out specified signal chunks which match the time-stamps of selected annotation segments.

As already noted, currently signal files are stored in the file system and not in the database, though treatment in the database is being considered. The tools required for handling audio files are described in [Trippel and Gibbon, 2001]. The audio clips are used for further analysis, including the creation of oscillograms, spectrograms and pitch tracks, as well as for direct audio feedback.

7 Conclusion: evaluation and further work

We started with a need for XML oriented database storage of multimodal resources, in particular multitier annotations. In order to specify the required design we introduced a distinction between General Resource Objects (types of resource) and their instances, Specific Resource Objects; in order to relate these to resource storage strategies we also distinguished Abstract Resource Objects (abstract data types in the informatic sense), and Implementational Resource Objects (concrete data types offered by specific

representation languages like XML or programming languages like XSLT or XQuery). On this basis we discussed the specification of resourcce storage application implemented using the TAMINO DBMS.

The initial evaluation of the approach takes the form of a proof of concept implementation. The Specific Resource Object is currently a TASX-based audio corpus in the TAMINO environment, accessed via a Perl API[11]. The audio signal processing uses modules from the PAX environment [Trippel and Gibbon, 2001].

Problems which are not solved in the proof of concept implementation and which remain to be addressed include the following:

- The complex web graphical user interface, which currently creates problems if metadata are incomplete.
- Potential inconsistency in the file-system storage of signal data in corpora, i.e. the content of the corpora, requiring validation of the corpora themselves. This calls for integration into the DBMS by means of server extensions.
- Because of non-optimized queries the processing time is considerable, since XQuery is used for the creation of the output format, not XSLT; format transformation should be left to XSLT, however, and performed in the passthru servlet.

Further work includes the generation of other views on the data for more fine-grained use cases, with access functions which are not dependent on the order of selecting first a subcorpus, then a layer, then a segment. Work on the user interface is also required, and queries need to be optimized for performance.

References

[Bird and Liberman, 2001] Bird, S. and Liberman, M. (2001). A formal framework for linguistic annotation. *Speech Communication*, 33(1,2):23–60.

[Boag et al., 2003] Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J., and Siméon, J. (2003). XQuery 1.0: An XML query language. URL: http://www.w3.org/TR/xquery/ , checked February 2004. W3C Working Draft 02 May 2003.

[Draxler, 1997] Draxler, C. (1997). Appendix H: Database Management Systems. In Gibbon, D., Moore, R., and Winski, R., editors, *Handbook of Standards and Resources for Spoken Language Systems*. Mouton de Gruyter, Berlin.

[Gibbon et al., 2004a] Gibbon, D., Ahoua, F., Gbery, E., Urua, E.-A., and Ekpenyong, M. (2004a). WALA: a multilingual resource repository for West Africa Languages. In *Proc. LREC2004*, Paris. ELRA.

[Gibbon et al., 2004b] Gibbon, D., Bow, C., Bird, S., and Hughes, B. (2004b). Securing Interpretability: The Case of Ega Language Documentation. In *Proc. LREC2004*, Paris. ELRA.

[Gibbon et al., 2003] Gibbon, D., Gut, U., Hell, B., Looks, K., Thies, A., and Trippel, T. (2003). A computational model of arm gestures in conversation. In *Proceedings of Eurospeech 2003*, Geneva.

[11] `http://www.spectrum.uni-bielefeld.de/modelex/implementation/concordance.html`

[Hell et al., 2003] Hell, B., Trippel, T., and Gibbon, D. (2003). TaminoAPI: An application programmer's interface to the Tamino XML database server. http://www.spectrum.uni-bielefeld.de/modelex/publication/techdoc/taminoapi-0_91/.

[Milde and Gut, 2002] Milde, J.-T. and Gut, U. (2002). The TASX-environment: an XML-based toolset for time aligned speech corpora. In *Proceedings of LREC 2002*, pages 1922—1927, Las Palmas.

[TASX-Annotator, 2004] TASX-Annotator (2002–2004). TASXAnnotator. Software, download at http://tasxforce.lili.uni-bielefeld.de.

[Trippel, 2004] Trippel, T. (2004). Metadata for time aligned corpora. In *Proceedings of the Workshop: A Registry of Linguistic Data Categories within an Integrated Language Repository Area, at LREC 2004*, Lisbon. ELRA.

[Trippel and Gibbon, 2001] Trippel, T. and Gibbon, D. (2001). PAX- an annotation based concordancing toolkit. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 238–244, Philadelphia.

[Trippel et al., 2004] Trippel, T., Sasaki, F., and Gibbon, D. (2004). Consistent storage of metadata in inference lexica: the MetaLex approach. In *Proc. LREC2004*, Paris. ELRA.

A Sample XQuery

The following is an XQuery expression selecting a part of a corpus around a search word, which in this case is a gesture transcripition; the expression is to be found in a specifyed session and on a specific annotation layer. The context is given by a time interval around the search word.

This query has been used and tested with TAMINO (version 4.1.4.1) and saxon (version 7 and 8). The only necessary change is the input function which is `input()` in TAMINO but `doc()` in the current version of the Working draft (W3C Working Draft 12 November 2003).

```
<tasx>{
for $session in input()/session,
$sessionmeta in $session/meta/desc

let     $word := "16m,5A,up,li,5D,m,r(0),me,13m,5D,rp",
$layermetadataval := "CoGesT-rp",
$layermetadatacat := "layer name",
$sessionmetadataval := "Anyi_Story_17",
$sessionmetadataname := "SessionTitle",
$interval := 2
where $sessionmeta/name= $sessionmetadataname and
$sessionmeta/val = $sessionmetadataval
return <session type="subsession">
{
for $layer in $session/layer
return
   for $event in $layer/event
   where $event/text() = $word
   return
     for $targetlayer in $event/../../layer
     return
        <layer l-id="{$targetlayer/@l-id}" type="sublayer">
{
for $targetevents in $targetlayer/event
```

```
where $targetevents/@start  >= $event/@start - $interval  and
$targetevents/@end <=  $event/@end + $interval
return $targetevents
}
</layer>
}
</session>
}</tasx>
```

B Sample TASX formatted corpus

The following is a selection of a larger corpus, encoded using the *Time Aligned Signaldata eXchange format* (TASX). It contains a number of tiers (layers), including gesture (annotated using the CoGesT annotation scheme), prosody, syllable structure, English and French gloss. The language of the corpus is Anyi, spoken in Ivory Coast.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<tasx>
   <session type="subsession">
      <layer type="sublayer" l-id="CoGesT-rp">
         <event e-id="14" start="54.385" end="56.265">
 16m,5A,sy</event>
         <event e-id="5" start="56.265" end="56.55">
 16m,5A,up,li,5D,m,r(0),me,13m,5D,rp</event>
         <event e-id="6" start="56.55" end="56.75">
 13m,5D,do,li,5D,s,r(0),me,14m,5D,rp</event>
         <event e-id="7" start="56.75" end="57.080000000000005">
 14m,5D,ri/up,li,5C,xs,r(0),me,14r,5D,rp</event>
         <event e-id="8" start="57.080000000000005" end="57.425000000000004">
 14r,5D,do/le,ar,5A,s,r(0),me,16m,5A,rp</event>
      </layer>
      <layer type="sublayer" l-id="CoGesT-lp">
         <event e-id="9" start="56.265" end="56.49">16m,5A,lp</event>
         <event e-id="10" start="56.49" end="56.72">
 16m,5A,up,li,5D,s,r(0),me,14m,5D,lp</event>
         <event e-id="11" start="56.72" end="56.96">
 14m,5D,le/do,ar,5D,xs,r(0),me,14m,5D,lp</event>
         <event e-id="12" start="56.96" end="57.425000000000004">
 14m,5D,do/ri,ar,5A,s,r(0),me,16m,5A,lp</event>
      </layer>
      <layer type="sublayer" l-id="complex gestures">
         <event e-id="14" start="54.385" end="56.265">p</event>
         <event e-id="3" start="56.265" end="57.425000000000004">g</event>
      </layer>
      <layer type="sublayer" l-id="spat_rel_1">
         <event e-id="14" start="54.385" end="56.265">
 hands between thighs, contact of palms</event>
      </layer>
      <layer type="sublayer" l-id="spat_rel_2">
         <event e-id="14" start="54.385" end="56.265">
 source=(contact,palm-r,palm-l),
 (contact,hands,inside of thighs)</event>
      </layer>
      <layer type="sublayer" l-id="misc">
         <event e-id="13" start="56.265" end="57.425000000000004">
 sy, but different temporal onsets</event>
```

```
      </layer>
      <layer type="sublayer" l-id="transcr_conte17_alex.misc">
         <event e-id="22" start="54.364254579802093" end="54.853538232274737"/>
         <event e-id="23" start="54.853538232274737" end="56.257569582848404"/>
         <event e-id="24" start="56.257569582848404" end="57.19359048323085"/>
      </layer>
      <layer type="sublayer" l-id="transcription_conte17_karin.Problems">
         <event e-id="10" start="55.484546133343194" end="56.257569582848404">H: ?</event>
      </layer>
      <layer type="sublayer" l-id="conte17_Fr_En_syll_tone.tones">
         <event e-id="53" start="54.364254579802093" end="54.853538232274737">silence</event>
         <event e-id="54" start="54.853538232274737" end="55.043686993855687">H</event>
         <event e-id="55" start="55.043686993855687" end="55.231467542737541">H</event>
         <event e-id="56" start="55.231467542737541" end="55.342993701309524">H!</event>
         <event e-id="57" start="55.342993701309524" end="55.484546133343194">H</event>
         <event e-id="58" start="55.484546133343194" end="56.020494539061787">H</event>
         <event e-id="59" start="56.020494539061787" end="56.257569582848404"/>
         <event e-id="60" start="56.257569582848404" end="56.440058101654607">H</event>
         <event e-id="61" start="56.440058101654607" end="56.623612496803943">B</event>
         <event e-id="62" start="56.623612496803943" end="56.739428123013305">B</event>
         <event e-id="63" start="56.739428123013305" end="56.92173049760212">B</event>
         <event e-id="64" start="56.92173049760212" end="57.039690857630177">H</event>
         <event e-id="65" start="57.039690857630177" end="57.19359048323085">B</event>
         <event e-id="66" start="57.19359048323085" end="57.424468917551188">???</event>
         <event e-id="67" start="57.424468917551188" end="58.102204804257845"/>
      </layer>
      <layer type="sublayer" l-id="conte17_Fr_En_syll_tone.syllables">
         <event e-id="53" start="54.364254579802093" end="54.853538232274737"/>
         <event e-id="54" start="54.853538232274737" end="55.043686993855687">be</event>
         <event e-id="55" start="55.043686993855687" end="55.231467542737541">NU~</event>
         <event e-id="56" start="55.231467542737541" end="55.342993701309524">nI~</event>
         <event e-id="57" start="55.342993701309524" end="55.484546133343194">ni</event>
         <event e-id="58" start="55.484546133343194" end="56.020494539061787">ba~:</event>
         <event e-id="59" start="56.020494539061787" end="56.257569582848404"/>
         <event e-id="60" start="56.257569582848404" end="56.440058101654607">be</event>
         <event e-id="61" start="56.440058101654607" end="56.623612496803943">su</event>
         <event e-id="62" start="56.623612496803943" end="56.739428123013305">a~</event>
         <event e-id="63" start="56.739428123013305" end="56.92173049760212">wa~</event>
         <event e-id="64" start="56.92173049760212" end="57.039690857630177">di</event>
         <event e-id="65" start="57.039690857630177" end="57.19359048323085">E</event>
         <event e-id="66" start="57.19359048323085" end="57.424468917551188">eh</event>
         <event e-id="67" start="57.424468917551188" end="58.102204804257845"/>
      </layer>
      <layer type="sublayer" l-id="conte17_Fr_En_syll_tone.French">
         <event e-id="21" start="54.364254579802093" end="54.853538232274737">silence</event>
         <event e-id="22" start="54.853538232274737" end="56.020494539061787">
les deux ont decidé</event>
         <event e-id="23" start="56.020494539061787" end="56.257569582848404"/>
         <event e-id="24" start="56.257569582848404" end="57.19359048323085">
d'apprendre à courir</event>
      </layer>
      <layer type="sublayer" l-id="conte17_Fr_En_syll_tone.English">
         <event e-id="21" start="54.364254579802093" end="54.853538232274737">silence</event>
         <event e-id="22" start="54.853538232274737" end="56.020494539061787">
the two have decided</event>
         <event e-id="23" start="56.020494539061787" end="56.257569582848404"/>
         <event e-id="24" start="56.257569582848404" end="57.19359048323085">
to learn how to run</event>
      </layer>
      <layer type="sublayer" l-id="conte17_Fr_En_syll_tone_korrigiert2.ns_tones_segmented">
```

```
        <event e-id="21" start="54.364254579802093" end="54.853538232274737"/>
        <event e-id="22" start="54.853538232274737" end="56.020494539061787">
 H - H - H! - H - H:</event>
        <event e-id="23" start="56.020494539061787" end="56.257569582848404"/>
        <event e-id="24" start="56.257569582848404" end="57.19359048323085">
 H - B - B - B - H - B</event>
     </layer>
     <layer type="sublayer" l-id="conte17_Fr_En_syll_tone_korrigiert2.ns_syll">
        <event e-id="21" start="54.364254579802093" end="54.853538232274737">silence</event>
        <event e-id="22" start="54.853538232274737" end="56.020494539061787">
 be NU~ nI~ ni ba~:</event>
        <event e-id="23" start="56.020494539061787" end="56.257569582848404"/>
        <event e-id="24" start="56.257569582848404" end="57.19359048323085">be sua~ wa~diE</event>
     </layer>
     <layer type="sublayer" l-id="conte17_Fr_En_syll_tone_korrigiert2.ns_syll_segmented">
        <event e-id="21" start="54.364254579802093" end="54.853538232274737">silence</event>
        <event e-id="22" start="54.853538232274737" end="56.020494539061787">
 be - NU~ - nI~ - ni - ba~:</event>
        <event e-id="23" start="56.020494539061787" end="56.257569582848404"/>
        <event e-id="24" start="56.257569582848404" end="57.19359048323085">
 be - su - a~ - wa~ - di - E</event>
     </layer>
  </session>
</tasx>
```