The TSNLP Database

— From tsct to tsdb —

Stephan Oepen, Klaus Netter
Judith Baur, Tom Fettig, Judith Klein, Fred Oberhauser

Report to LRE 62-089 (D-WP6.1)

Test Suites for Natural Language Processing (TSNLP)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

June 1995

Summary

Title	The TSNLP Database. From tsct to tsdb.
Project	LRE-62-089
Document ID	D-WP6.1
WP	Design and Implementation of Database
Date	June 1995
Authors	Stephan Oepen, Klaus Netter, Judith Baur, Tom Fettig, Judith Klein, Fred Oberhauser
Reviewers	The consortium
Status	final document

Contents

The TSNLP Database

— From tsct to tsdb —

1 Overview

The objective of TSNLP work package 6 (WP6) is to design and implement a relational database that will ease the storage, maintenance and retrieval of TSNLP test data. Key desiderata of the database design include:

- **usability**: to allow for the application of the methodology, technology and test data developed in TSNLP to a wide variety of diagnosis and evaluation purposes of very different applications by developers or users with varied backgrounds (e.g. academic vs. industrial);
- **suitability**: to meet the specific necessities of storing and maintaining natural language test data (e.g. in string processing) and to provide maximally flexible interfaces;
- **extensibility**: to enable and encourage users of the TSNLP database to add test data and annotations according to their needs without changes to the underlying database;
- **portability and simplicity**: to make the results of TSNLP available on several different hard- and software platforms free of royalties and easy to use.

In both test data construction and database design issues TSNLP could build on experiences gained in the DiTo project ([Nerbonne *et al.* 1993]) previously carried out at DFKI. The DiTo effort produced some 1500 systematically constructed and annotated sentences for four syntactic phenomena of German; the test data were organized into a simple relational database that was implemented by means of standard Un∗x text processing tools (viz. `sort(1)`, `uniq(1)` and `join(1)`) combined with a query engine (an `awk(1)` script) and a query processor (realized in `yacc(1)` and `lex(1)`).

The annotation schema and parts of the data developed in DiTo (for German at least) served as a starting point for work done in TSNLP. At the same time, however, the DiTo project revealed two important problems that prompted the TSNLP consortium to pursue more elaborate approaches:

(i) Data construction in DiTo was directly carried out in the internal format defined by the database: the test data authors had to know and understand some of the database internals and use a standard text editor to produce the appropriate format; the method, obviously, turned out to be inefficient, laborious and error prone;

(ii) although `awk(1)` as the query engine provided for very high-level string manipulation facilities, with growing database sizes processing efficiency became a problem since `awk(1)` is an interpreted language; additionally, even though the Un*x text processing tools deployed in DiTo are (at least) available for the Microsoft DOS universe as well, the hybrid combination of scripts with `awk(1)` and `yacc(1)` programs imposed a severe limitation on portability and simplicity.

In a sense, items (i) and (ii) from the above list mirror the structure of the present document: (i) because it seemed necessary to support the test data construction in the (ongoing) TSNLP work package 3 (WP3) with an intelligent, form-based input tool independent of the eventual database schema, DFKI implemented a Test Suite Construction Tool (called tsct; see section 2) and made it availabe to the consortium. Yet, in the original project proposal there was no work package allocated to this task such that the tool has not been documented in TSNLP reports so far. Since, however, the tsct data format in turn serves as the vantage point for (ii) the design and implementation of a portable Test Suite Database (tsdb; see section 3) as well as for the import procedure into the database (see section 3.3 and appendix C) we include the documentation of tsct as the first part of this report.

To allow for the dissemination and wide distribution of TSNLP results, obviously the technology developed has to be of almost industrial strength. Therefore, the dabase design primarily puts emphasis on engineering rather than on scientific aspects such that simplicity, robustness and documentation of the software play a very important role. Because the tools implemented in work package 6 will have to be documented to an extent that they can be used by outside parties, it is foreseen that a revised version of the present report will be made available after the completion of the implementation.

2 The Test Suite Construction Tool (tsct)

2.1 Motivation

Very generally, the motivation for the development of a Test Suite Construction Tool (tsct) particular to the TSNLP project has already been sketched: the basic idea in building tsct was to facilitate and guide the production of systematically and consistently annotated test data in three languages by several test data authors at four geographically distant sites. More specifically, the objectives of tsct are:

- to provide a mask-based graphical editor (i.e. an input form fitted to the annotation schema) for TSNLP test data;
- to promote the systematic production of test data by tailoring tsct to the TSNLP annotation schema developed in work package 2.2 (see [Estival *et al.* 1994]);[1]
- to partly automate the writing of test data through the reuse and adaptation of previously constructed data;
- to fit the tool to the needs of the test data author by providing specialized editing functionality (i.e. in string manipulation and the on-line computation of dependent field values);
- to control the test data construction through the form-based input mask to reduce erratic variation and ensure a basic degree of formal consistency;
- to allow for convenient browsing and navigation in TSNLP test data files, thus facilitating the exchange and comparison of data among different test data authors and sites.

Besides, tsct provided the consortium with a uniform and well-defined format of storing the TSNLP test data (see section 2.9) well before the scheduled start of the database design work package. The machine-readable tsct data format greatly simplifies the automated import procedure into the TSNLP database, such that the labour that was originally allocated for this task can be transferred into data construction and consistency checking.

2.2 Implementation

The tsct implementation is based on the ANSI C and X11 industrial standards; hence, the tool is efficient, highly portable and can be made free software. To minimize the development time and to provide for easy adaptation and extension, tsct was built on top of the Widget Creation Library

[1] Although several of the attribute names and part of the grouping have changed in-between the work package 2.2 report, the tsct and, in turn, the tsdb implementations respectively, it should become obvious how the individual parts relate to each other by simple comparison of the three representational stages.

(Wcl) available from the public domain (e.g. through the Usenet group `comp.sources.x` and most major `ftp(1)` servers). Following is a quotation from the Wcl 2.0 distribution:

> Wcl is the Widget Creation Library. It allows all Look and Feel aspects for an application to be specified in Xrm resource files, leaving your application code to consist of a simple main() and callback procedures. Wcl greatly reduces the learning curve for developing applications with Widget based user interfaces, yet does not restrict the developers to platforms, widget sets, nor development languages.
>
> Wcl is widget set independent, but Xt Intrinsics dependent. Wcl can be used on Xt Intrinsics from X11R3, Motif 1.0 (X11R3.5 as with SCO ODT), X11R4, and X11R5. Some capabilities are not available on Xt Intrinsic versions before X11R5.
>
> Wcl is portable: the distribution has been built and tested on platforms with SVR3, SVR4, SunOS, and other varieties of UNIX. Some embriotic support for VMS is also provided. Systems with dynamic linking (SunOS and SVR4) which have dlopen() and dlsym() can also invoke dynamically bound callbacks and actions from shared libraries.
>
> Support for Athena, Cornell, Motif, and OpenLook widgets are provided with campanion libraries (widget set registration functions) and resource interpreters with many example resource files.

— Accordingly, tsct comes as a single executable file (the resource interpreter) together with four separate resource definition files corresponding to the organization of the TSNLP annotation schema into four basic windows (see sections 2.4 to 2.7). The command line synopsis is:

`tsct -rf` *resource file*
    `-file` *data file*
    [`-intelligent`]
    [`-offset` *n*]
    [*generic X11 options*]

where

- *resource file* is one of the Wcl resource description files corresponding to the four tsct windows (i.e. one of `item.ari`, `phenomenon.ari`, `set.ari` or `profile.ari`; see appendix A);
- *data file* is a data file in tsct format (see section 2.9); if the file already exists it is opened in appending mode and the original version is preserved as a backup copy (file name with '~' suffix);
- `-intelligent` is only applicable to the *Test Item* window and enables the automatic computation of the *Length* and *Instance* (in the embedded *Analysis* table) fields (see section 2.4);
- *n* is an integer setting the initial browsing position to the n^{th} record in *data file*; and
- *generic X11 options* is a set of standard X11 options such as `-title`, `-geometry` or `-font` (see `X(1)` for details).

The current version of tsct (viz. 1.0) is stable and in use by all partners since April 1995; although it is not envisaged that further development of tsct will become necessary for TSNLP, bug reports and suggestions for improvement are welcome by electronic mail to `tsct@cl.dfki.uni-sb.de`. Currently, tsct binaries are available for Sun and HP workstations as well as for Intel x86 based machines running the Linux operating system. Versions for most any Un∗x flavour can be made available on demand.

2.3 Navigation in tsct

Each of the four tsct windows (see sections 2.4 to 2.7) displays the contents of one data file one record at a time. To navigate through the list of data records (e.g. individual test items in the case of the *Test Item* window) there are Previous and Next buttons moving the current browsing position through the data file backwards and forwards respectively. Clicking the left, middle or right mouse buttons on Previous or Next moves by one, five or ten data records respectively. To indicate the current browsing position in the list of records, the upper right corner of each of the windows displays the active record and the total number of records (in the format 'Data Set # m (of n)' where m is current and n total). Since the *Test Item* and *Phenomenon* windows have embedded *Analysis* and *Parameters* parts that in turn are list (or table) valued, for these there are additional movement buttons labelled prev and next.

Navigation through the individual input fields is by mouse or cursor or tabulator key movement (see appendix B for details); within any of the text input fields line editing facilities are similar to the GNU Emacs key bindings.

In addition to the global navigation means each of the tsct windows has the following buttons:

- Reset — to distinguish between fields that typically remain unchanged (or static) throughout one tsct session (e.g. the *author* and *date* information) and those that vary for each individual data record each of the four windows has a solid horizontal line delimiting the former set of fields (the upper part of the window) from the latter (the lower and typically larger part of the window); the Reset button deletes all field values of the current record except for those from above the delimiter line (i.e. the non-static ones);
- Delete removes the current data record from the list of records;
- New creates a new data record as a copy of the current record (by duplicating all its information) and appends it to the list of records;
- New + Reset is similar in effect to the sequence of New and Reset: it creates a new but empty (except for the static fields) data record;
- Save writes the current list of records to disk (i.e. into the *data file* specified at startup);

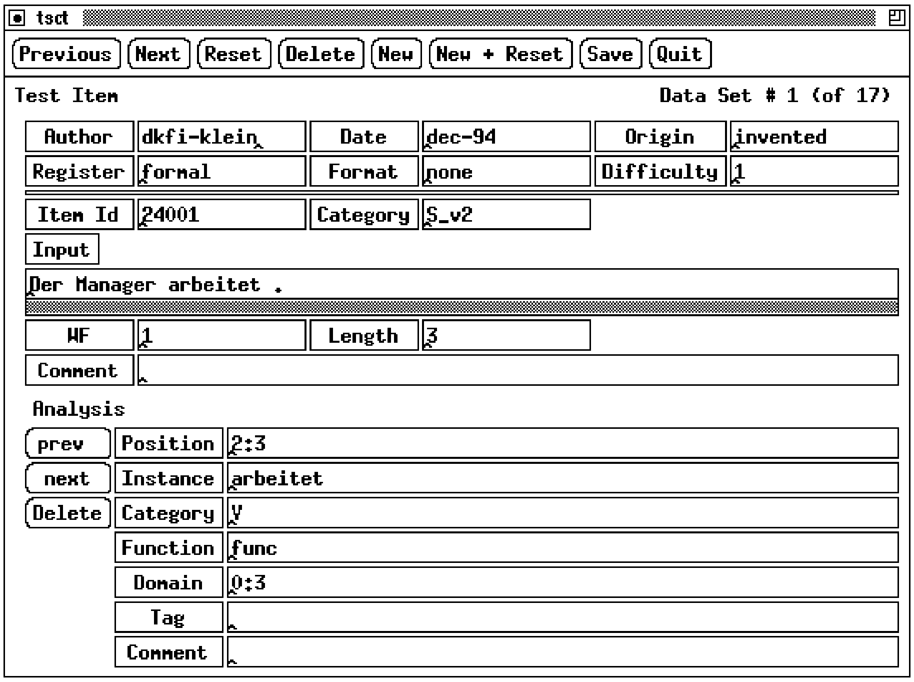

Figure 1: The *Test Item* window of tsct.

- Quit, finally, allows to exit from tsct without saving the current state; if, however, the active list of data records and the *data file* differ, a popup dialogue will request confirmation of the quitting first.

2.4 Test Item Window

Figure 1 shows the layout of the *Test Item* window of tsct that accommodates the **core data** of any test suite, viz. the individual test items (sentences, phrases *et al.*) together with all general, categorial and structural information that is independent of phenomenon, application and user parameters. Besides the actual string of words, annotations at this level include (i) bookkeeping records (date, author, origin and item id), (ii) the item register, format, length, category and difficulty and wellformedness codes, (iii) the (morpho-)syntactic categories and string positions of lexical and — where uncontroversial — phrasal elements, and (iv) an abstract dependency structure.

2.5 Phenomenon Window

The tsct *Phenomenon Window* (see figure 2) serves a dual purpose: firstly, it is the home for the descriptions of individual phenomena, and, secondly, it allows one to associate test items (and their parameters) with phenomena; in this sense, it holds the vast majority of **phenomenon related data**.

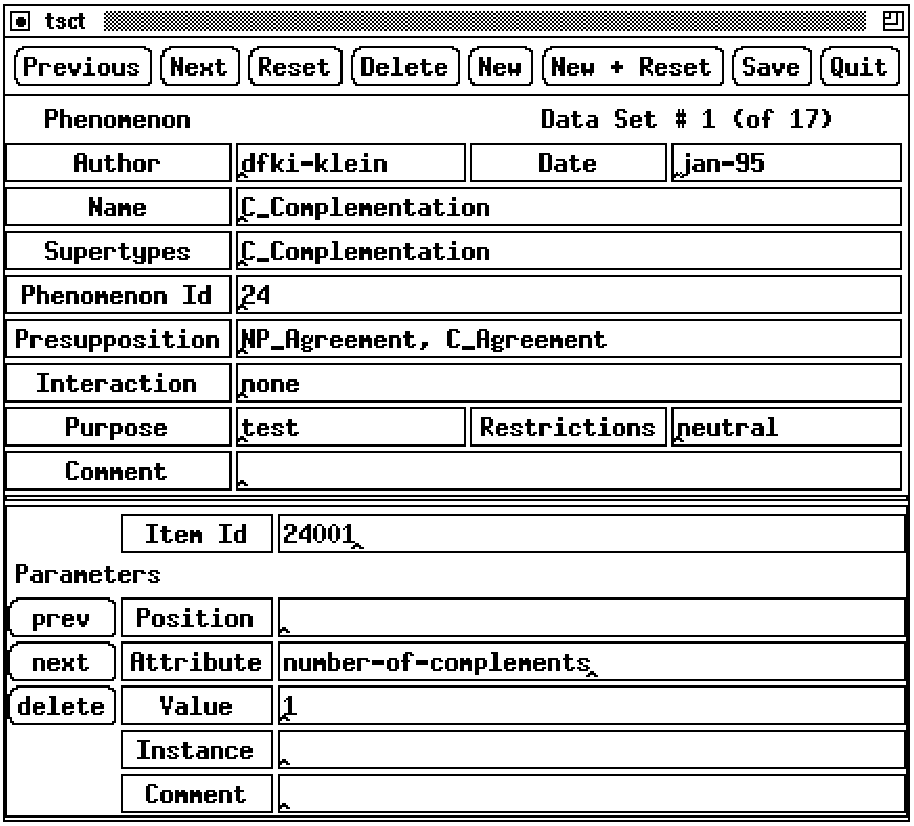

Figure 2: The *Phenomenon* window of tsct.

Based on a hierarchical classification of phenomena (e.g. verb valency being a subtype to general complementation), each phenomenon description is identified by its phenomenon id, supertype(s), interaction with other phenomena, and presupposition of such. Moreover, the set of (mostly syntactic) parameters that is relevant in the analysis of a token phenomenon (i.e. the number and type of complements in the case of verb valency) is determined.[2] Individual test items are then assigned to one or several phenomena and annotated according to the phenomenon-specific parameters.

2.6 Test Set Window

As an optional descriptive device inbetween the test item and phenomenon levels pairs or sets of grammatical and ill-formed items can be grouped into so-called **test sets** (see figure 3). Thus, it can be encoded how for

[2] Although originally a plain relational database representation was envisaged (see below), the direct encoding of the phenomena hierarchy would suggest a notion of parameter inheritance similar to concepts known from taxonomic databases. Given the limited application, however, it remains highly questionable whether it is worthwhile — if at all possible — to incorporate such a concept into the TSNLP data model.

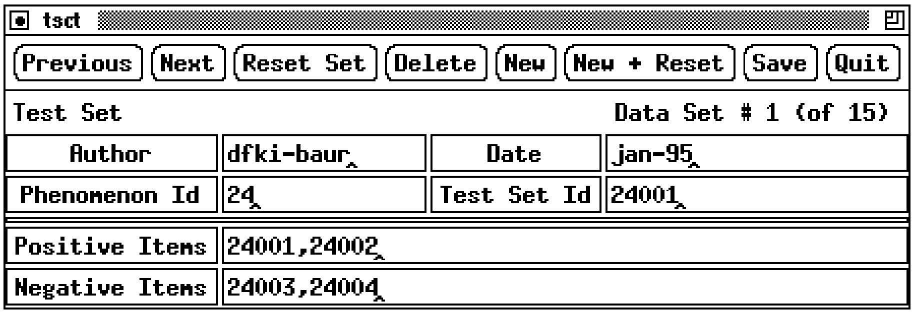

Figure 3: The *Test Set* window of tsct.

example test items that originate from the systematic variation of a single phenomenon parameter relate to each other.

2.7 User & Application Profile Window

Information that will typically correlate with different evaluation and application types in the use of a test suite is factored from the remainder of the data into **user & application profiles** (see figure 4). Currently, we have foreseen (i) a centrality measure on a per phenomenon, test set or even test item basis (e.g. user x may in general consider phenomenon y to be central but test set z to be marginal within y) and (ii) judgements of relevance with respect to a specific application. To match the results obtained from some NLP system (a parser, say) against a test suite, users can add informal or formal output specifications at the profile level. As these are dependent on the purpose of the use of the test suite, there is no template provided.

In general, the user & application profiles are considered to be a hook into the TSNLP database that developers or end users with only a very basic knowledge of relational database technology shall use to extend the database schema and add user- or application-specific data without touching the core inventory of test items and annotations. Accordingly, data in this part of the database will not necessarily be input through tsct — which typically will be too restricted in its static number of fields — but possibly imported from different sources (e.g. output specifications for a specific application might be produced semi- or fully automatically).

2.8 A Typical Test Data Construction Scenario

The following algorithm (sequence of actions) has been suggested for the test data construction using tsct; and even though probably procedures or shortcuts very different to these may turn out to be applicable depending on the individual preferences of some test data author or the specific characteristics of some phenomenon,[3] the algorithm proposed can be read as a

[3]In the German coordination data, for example, the core strings and some of the annotations already result from systematic permutations and deletions with respect to an

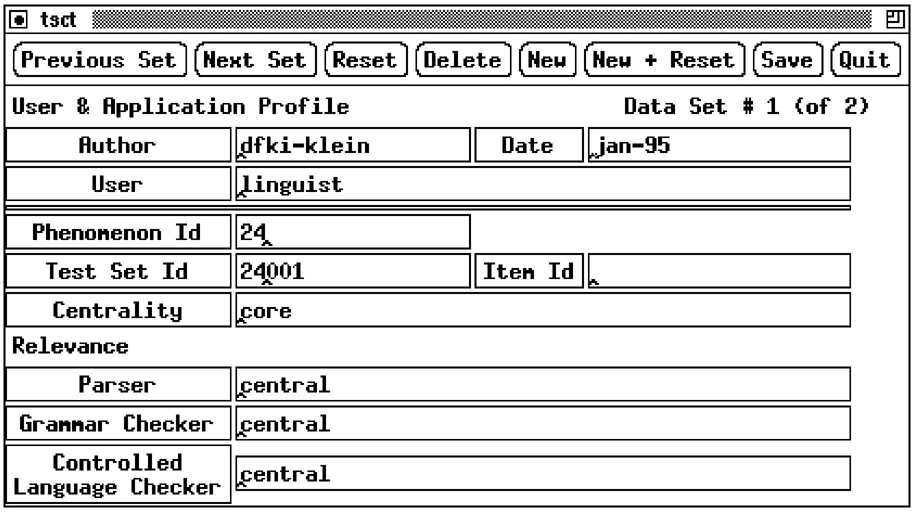

Figure 4: The *User & Application Profile* window of tsct.

reasonable (but naturally informal) procedure to follow (the instructions to save frequently have meanwhile become less important as tsct has matured and become more stable).

(1) Fill in a particular *Phenomenon* description and identify the parameters that play a role in testing and systematic variation;

(2) instantiate the *Test Item* form and save it;

(3) paste the id from step (2) into the *Phenomenon* window; add any relevant *Parameters* information and save it;

(4) iterate through steps (2) and (3);

(5) organize the test items from steps (2) into test sets; fill in the *Test Set* window accordingly;

(6) iterate through steps (2) to (5);

(7) add the classification for the current phenomenon to the *User & Application Profile* window;

(8) continue with step (1).

2.9 The tsct Data File Format

Because the design and implementation of the TSNLP database is scheduled only towards the end of the project, the tsct data file format has to serve several different purposes (viz. data exchange in WP3, test item retrieval in

initial (or base) sentence that were computed off-line; see [Balkan *et al.* 1994] for details.

WP4 and as an interface to the tools built in WP5) throughout the project. To meet the various requirements

- tsct organizes the test data into human-readable ASCII files that can be manipulated using ordinary text processing tools (e.g. standard editors and Un∗x commands or even Microsoft Word macro processing); and
- at the same time the use of tsct enforces the data to be stored in a well-defined machine-readable exchange format.

— For further details on the tsct data file format see appendix B.

3 The Test Suite Database (tsdb)

3.1 Motivation

Since the main objectives for the test data storage, maintenance and retrieval in TSNLP are *usability*, *suitability* and *extensibility* (see section 1), the database technology and tool building are crucial to the successful completion of the project and even more so to the further assessment and dissemination of its results. The objective of work package 6 (WP6) is to design and implement a relational database that has all the necessary properties. Special emphasis in the database design is put on the aspect of flexibility and interfaces (to both people and programs); in a recent working meeting the TSNLP consortium agreed on the importance of being aware of the fact that technological requirements of developers or users from an academic vs. industrial background differ substantially. Accordingly, from a technical perspective *portability* and *simplicity* of the database have to be included into the list of desiderata.

The basic idea of the work package is:[4]

- to design an abstract data model that represents TSNLP test data and annotations in a linguistic database;
- to organize the data into a relational database that, in a sense, can serve as a virtual (or meta-) test suite allowing the retrieval (i.e. instantiation) of actual test suites according to arbitrary linguistic or extra-linguistic constraints over the data;
- to define an import procedure from tsct into tsdb that eliminates redundancy and ensures the basic consistency of the test data; and
- to automate the import procedure as an utility program that in later stages — when the database is in place — will still allow the use of the construction tool for data production.

Because the TSNLP partners firmly believe that the applicability and acceptance of the tools (i.e. the technology) produced by the project will be one of the keys to the wide distribution and deployment of the methodology and test data developed, its awareness for the differences in academic vs. industrial environments manifests in a dual database implementation strategy.

Firstly, to produce a tool that is well tailored to the purpose of storing and retrieving TSNLP test data, highly flexible in its interfaces and portable and free of royalties at the same time, DFKI implements a small and simple relational database engine in plain ANSI C (which we presently call tsdb_1 using the subscript to distinguish between the two parallel implementations of tsdb; see section 3.6). Since the expected size of the database as well as major parts of the database schema are known in advance, for this implementation it is plausible to impose a few restrictions on the full generality

[4]See [Oepen and Netter 1994] for a brief summary of the motivation and the design goals for the TSNLP database.

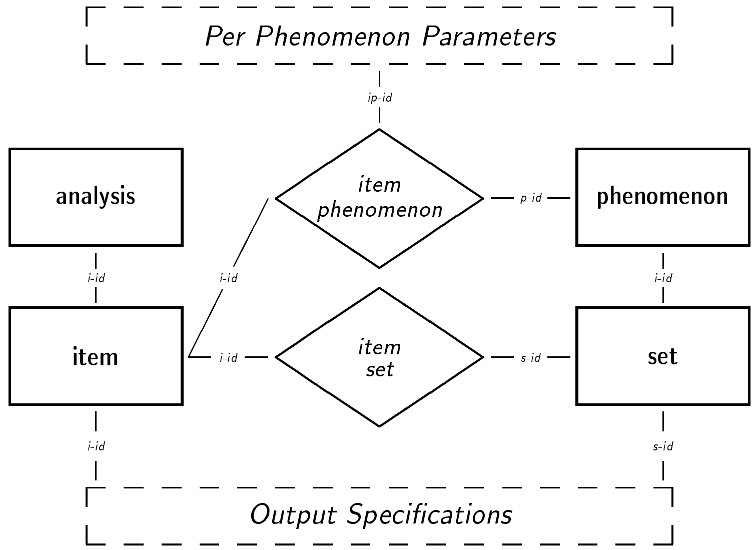

Figure 5: Graph representation of the TSNLP database schema; see figure 6 for details on the individual relations.

of relational database systems that will allow to fine-tune and simplify both the program and especially the query language.

Secondly, however, the consortium acknowledges the recommendation of its peer reviewers as well as the specific requirements of the industrial partner Aerospatiale in the parallel exploitation of a commercial database product (to yield tsdb_2; see section 3.7). Because commercially available database systems have different assets (and typically many more features) than the home-grown implementation tsdb_1, the dual realizations of the TSNLP database carried out in parallel are expected to nicely complement each other and to provide for valuable feedback and comparison.

3.2 The TSNLP Database Schema

While the current tsct data format was primarily motivated by data construction considerations (and possibly the requirements imposed by its use in work packages 3 through 5 before the actual database implementation), the design goals for the abstract TSNLP data model (and analogously for the tsdb database schema) are rather different.

Using the construction tool and its reuse and duplicate facilities, the resulting data files are highly redundant in that static fields (see section 2.3 for the distinction of static vs. reset fields) are stored with each individual data record; looking at the *Phenomenon* window (in figure 2) of tsct where about half of the fields are static, it is obvious that the rough classification into four basic relations serves very well for data input purposes but falls short from a data storage and maintenance perspective.

Accordingly, the four-valued classification corresponding to the tsct win-

Basic Entity Types

item
- i-id *:integer :key*
- i-origin *:string*
- i-register *:string*
- i-format *:string*
- i-difficulty *:integer*
- i-category *:string*
- i-input *:string*
- i-wf *:integer*
- i-length *:integer*
- i-comment *:string*
- i-author *:string*
- i-date *:string*

analysis
- i-id *:integer :key*
- a-position *:position*
- a-instance *:string*
- a-category *:string*
- a-function *:string*
- a-domain *:string*
- a-tag *:string*
- a-comment *:string*

phenomenon
- p-id *:integer :key*
- p-name *:string*
- p-supertypes *:string*
- p-presupposition *:string*
- p-interaction *:string*
- p-purpose *:string*
- p-restrictions *:string*
- p-comment *:string*
- p-author *:string*
- p-date *:string*

parameter
- ip-id *:integer :key*
- position *:string*
- attribute *:string*
- value *:string*
- instance *:string*
- comment *:string*

set
- s-id *:integer :key*
- p-id *:integer :key*
- s-author *:string*
- s-date *:string*

Relationship Types

item-phenomenon
- ip-id *:integer :key*
- i-id *:integer :key*
- p-id *:integer :key*
- ip-author *:string*
- ip-date *:string*

item-set
- i-id *:integer :key*
- s-id *:integer :key*
- polarity *:integer*

Figure 6: Preliminary TSNLP database schema. The **item** and **analysis** relations roughly correspond to the tsct *Test Item* window; the **phenomenon** and **item-phenomenon** relations to the *Phenomenon* and the **set** and **item-set** relations to the *Test Set* windows respectively. Relationship types are postulated for $m \times n$ relations only; $n \times 1$ relationships rather are represented as additional index attributes of the entity types (e.g. the *info-id* attribute of the **item**, **set** and **item-phenomenon** relations).

dows has to be refined and cast into more specific relations to allow for the elimination of redundancy. In addition to eliminating redundant information, the database schema has to more clearly

(i) factor basic entity (or object) types from relationship types (for a semi-formal introduction to the entity-relationship conception of databases see [Elmasri and Navathe 1989]); and

(ii) make the distinction explicit in its postulation of relations and index attributes used to relate entity types to each other in order to allow for variable logical views on the data.

Another important issue in the refinement of the data model is the representation of list- or multi-valued attributes which usually are not allowed in plain relational databases. Hence, the tsct representation of test sets — typically relating a comma-separated group of positive test items (or rather their identifiers) to a set of negative items — has to be transformed into multiple data records that relate the individual items to one and the same test set (viz. its identifier again); but since the lists of positive and negative items may not match up pairwise and in order to avoid empty fields, the polarity (positive or negative) for each of the items is cast as an attribute of its own.[5] As an example for this transformation consider the following records from the **set** and **item-set** relations respectively which are the tsdb equivalent of the test set in figure 3:

s-id	*p-id*	*info-id*
24001	24	4711

i-id	*s-id*	*polarity*
24001	24001	1
24002	24001	1
24003	24001	0
24004	24001	0

Figures 5 and 6 present the core of the database schema as it has been presented at the May 1995 working meeting of the TSNLP consortium; currently missing from the database schema is an account of how phenomenon-specific parameters will be represented and how user & application profiles will be linked to the core data (section 3.4 gives the reasons for these omissions).

3.3 Importing from tsct into tsdb

The import procedure from tsct into tsdb is implemented in the `awk(1)` programming language; it reads in tsct data files sequentially, distributes

[5] The reason to introduce a separate *polarity* attribute in addition to the already existing *i-wf* judgement on grammaticality of the test items themselves, is to still allow for test sets for which well-formed items can serve as negative examples or vice versa. In testing a controlled language checker, for example, it might be desirable to construct test sets containing grammatical items that still are assigned negative polarity simply because they are not part of the controlled target language (e.g. based on a distinction between active vs. passive voice).

the individual data records and fields over the appropriate tsdb relations, provides for the necessary transformations and links between relations and writes the resulting data into files again. Besides, the import procedure is the appropriate time and place to normalize and test for the basic logical consistency of the data regarding properties that are local to, say, one data record or input file at a time. Normalization and consistency checking can include (but need not be limited to):

- capitalization and inter-word and inter-punctuation spacing normalization;
- elimination of empty fields;
- checking of data types (for numerical fields and attributes with a limited set of admissible values);
- removal of duplicates (possibly originating from spelling variations in attribute values);
- inspection of *i-input* vs. *i-length* and *a-position* vs. *a-instance* mismatches; and
- generation of new unique identifiers where necessary (e.g. for the *ip-id* attribute).

— Essex has agreed to reallocate labour that was originally scheduled for the database import to normalization and checking of consistency.

The output file format of the import procedure is immediately suitable for the home-grown tsdb_1 implementation (see section 3.6): one ASCII file per relation with one record per line and variable-width fields delimited by a special field separator character. At the same time, the version of tsdb based on Microsoft FoxPro can directly read these files and convert them into its internal binary format. For further details on the tsct to tsdb import procedure see appendix C.

3.4 A Few Open Questions

The tsdb database schema — unlike the tsct data model which basically followed very pragmatic data construction considerations — is determined by formal and data organization criteria; therefore, there cannot be a one-to-one mapping between the two. Accordingly, the import procedure into tsdb has to incorporate a few non-trivial transformations and poses some open questions which mostly deal with the formal nature of the representation.

- In the *Analysis* part of the *Test Item* window the pair of attributes *function* and *domain* is overloaded in the description of functor – argument dependencies: whereas functor elements use *domain* to indicate the substring in which they act as the top-level functor, arguments, in turn, have the position of their functor as the value of *domain*. As the two interpretations of the attribute *a-domain* are rather different, it might become desirable to recast the dependency structure in a separate relation. Yet, there is no loss of information in the current encoding as long as there is agreement on its intended interpretation.

- In order to be maximally general in the description of phenomenon-specific parameters, the tsct *Phenomenon* window has the fields *attribute* and *value* which, strictly speaking, serve as a meta-level description of per phenomenon relations. Again, it is possible but possibly undesirable to simply carry over the current encoding into the database; however, a major disadvantage of this method would be that it imposes *absolutely no* restrictions on the number and type of parameters associated with some token phenomenon.

- The *User & Application Profile* of tsct basically can only serve as an initial approximation of the information that shall go into this part of the database (e.g. it has no fields to allow the input of application-specific output specifications since these typically will have variable formats).

Since in the data construction efforts so far there has been no emphasis on the annotations affected by the open questions, we expect to arrive at suitable solutions during the continued data construction work and the parallel instantiation of the database. In general, the harmonization of the data and a better understanding of the purpose of the annotations (e.g. for the phenomenon-specific parameters) seems to be a prerequisite to giving convincing answers to the above questions. Moreover, the compilation of the current inventory of parameters and values used — as it is currently worked out by the consortium — will provide for valuable input.

3.5 Query and Retrieval: Some Examples

For the TSNLP test data the basic purpose of the relational database retrieval facilities are:

- to dynamically construct test suite instances according to an arbitrary selection from the data; and

- to employ the query engine to compute appropriate virtual (join) relations that contain everything relevant for a particular task but at the same time hide away all internal, bookkeeping or irrelevant data.

— Following are a few retrieval examples formulated in the simplified SQL-like query language that is interpreted by the home-grown implementation tsdb_1 (see section 3.6):

- find all grammatical verb-second sentences relevant to the phenomenon of clausal agreement:

```
retrieve i-input
  where i-wf = 1 && i-category = "S_v2" &&
        p-name = "C_agreement"
```

- collect all grammatical sentences with pronominal subjects together with their identifiers and categories:

```
retrieve i-id i-input i-category
  where i-wf = 1 &&
        a-function = "subj" && a-category ~ "PRON"
```

- extract a list of noun phrases (i.e. substrings of the *i-input* field according to the *a-position* value) together with the identifiers of the test items that they are embedded in:

```
retrieve i-id i-input[a-position]
  where a-category ~ "NP"
```

3.6 Implementing tsdb_1: Building It Ourselves

The motivation for the dual database implementation has already been given in section 3.1. And even though it is not planned to compete with commercial database products in terms of efficiency and generality, there are good reasons speaking in favour of the proprietary implementation of tsdb_1 within TSNLP:

+ the database software and query language can be fine-tuned to the test suite extraction task; because the focus of natural language test data retrieval is on string manipulation, the tsdb_1 implementation will be enriched with regular expression matching and possibly some substring position arithmetics (two concepts that are typically unavailable in standard relational database products)

+ both data *and* software can be made available to interested parties royalty-free;

+ the database can be given a flexible and well-documented interface allowing to connect it to arbitrary (client) application within or outside of TSNLP; the applicability of the interface will be exemplified in at least two cases during work packages 4 and 6; and

+ the data format can be designed to be transparent (similar to the tsct ASCII files) and is thus accessible to standard Un∗x text processing utilities.

Figure 7 gives an overview of the tsdb_1 architecture: the database kernel is constituted by the inference engine and a separate layer of interface functions that are organized into a library of C functions.

Thus, any application that is capable of interfacing to a library of external functions (e.g. programs that can be statically linked with the library or

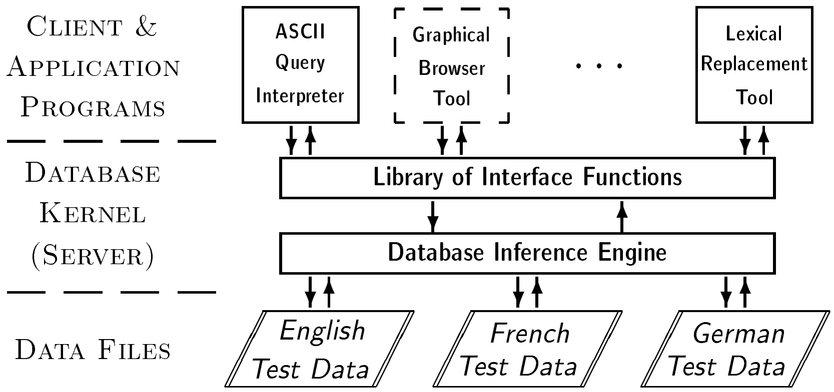

Figure 7: Rough sketch of the modular tsdb_1 design.

Common-Lisp or Prolog applications by use of the foreign function interface) can be given full bidirectional access to the database. Within work package 6 only one of the client applications will be developed, viz. an ASCII-based query interpreter that allows to retrieve data in a subset of the SQL query language (figure 8 shows a screen dump from a retrieval session using tsdb_1). The tsdb_1 query language will provide for the standard numeric and string comparison operations plus regular expression matching and the boolean connectives (viz. conjunction, disjunction and negation of conditions).

Other applications to connect to the TSNLP database could include the lexical replacement tool built in work package 5 (which currently reads and writes data files in the tsct format) and possibly a graphical browser for test data derived from tsct itself.[6] Additionally, it would be straightforward to set up tsdb_1 as a network database server that accepts connections from remote machines in a TCP/IP-based networked environment.

In addition to the work done in work package 6, DFKI will deploy the functional tsdb_1 interface in connecting the database to the parser for German that in the course of work package 4 will be used to test for the adequacy of TSNLP data and annotations. For this application the bidirectional interface will allow for automated retrieve, process and compare cycles in grammar development and diagnosis.

The implementation of tsdb_1 is in ANSI C and supports (among others) Un$*$x workstations as well as Macintosh and Intel x86-based personal computers. The interface specification is as a C header file to be included by client programs.

[6] For the time being, however, it is rather envisaged, that the implementation of tsdb based on the commercial database software Microsoft FoxPro (see section 3.7 for details) will allow for the convenient graphical browsing of the test data already since FoxPro comes with very high-level support for building graphical interfaces.

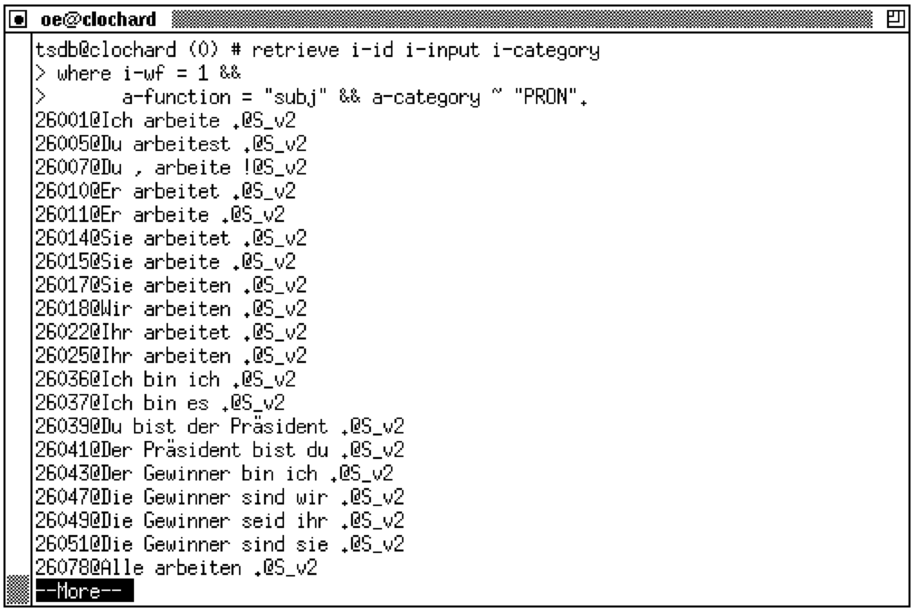

Figure 8: Screen dump of a tsdb_1 session: query results are displayed through a pager (`more(1)` in this case) and written into ASCII files. The tsdb_1 ASCII query interpreter (topmost three lines) has command line editing and relation and attribute name completion facilities.

3.7 Implementing tsdb_2: Using a Commercial Database

Even though the ANSI C implementation of tsdb_1 allows to deploy the tool in a standard personal computer environment (which is more typical in a non-academic context than are Un∗x-based workstation installations) easily, an alternate implementation of tsdb based on a commercial database product is carried out in parallel in order to produce a tool that has a *look + feel* familiar to personal computer users from standard office software.

Based on the evaluation of several low- and medium-priced commercially available database systems for Macintosh and Intel x86-based personal computers running under Microsoft Windows, the product of choice was Microsoft FoxPro (in its current version 2.6) because it has the following desirable properties:

- it is available for the two industry standard platforms (Apple Macintosh and MS Windows);
- it has sufficient descriptive power and efficiency;
- it comes with high-level interface building support for implementing a graphical frontend to the TSNLP database;
- its data format is compatible to the entire family of *xbase* database products;

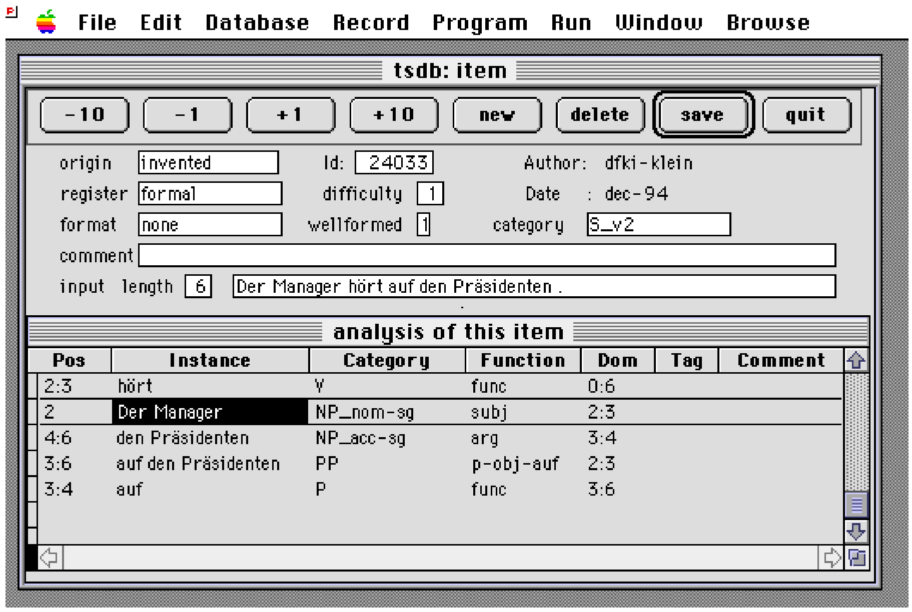

Figure 9: Screen dump of the development version of tsdb_2: the FoxPro database software has high-level support for graphical browsing of the data.

- it is very competetively priced; and
- because DFKI licensed a so-called FoxPro distribution kit, it will be possible to compile binary standalone versions of tsdb_2 and make them availabe to outside parties free of royalties.

The parallel implementation of two tsdb versions is possible because both tsdb_1 and tsdb_2 use exactly the same database schema (see section 3.2); thus the two resulting databases come out as one-to-one replicas of each other such that data can be freely exchanged between them at any time.

While the home-grown implementation of tsdb_1 scores high for its flexible programmatic interface (to client software), the FoxPro version tsdb_2 will be superior in its user interface (see figure 9 for a screen dump of one of the windows in the current development version). Since FoxPro encourages the implementation of graphical, mouse- and menu-based forms that hide away technical details of the underlying database, tsdb_2 will come with a sophisticated graphical browser and editor for the TSNLP test data.

Similar to the tsdb_1 implementation, FoxPro has a builtin retrieval engine that processes queries in a subset of SQL. To feed test data into applications to be tested in diagnosis or evalution, the FoxPro export facilities allow to write ASCII files in various formats.

A Contents of the Current tsct Distribution

Following is a listing of the contents of the current distribution of tsct version 1.0 (as of June 1995):

```
oe@clarinet (~) 52 $ tar ztf tsct.tar.z
tsct-1.0/
tsct-1.0/tsct
tsct-1.0/item.ari
tsct-1.0/phenomenon.ari
tsct-1.0/profile.ari
tsct-1.0/set.ari
tsct-1.0/README
```

The distribution is in `gzip(1)`ed `tar(1)` format and available from the TSNLP `ftp(1)` repository at `tsnlp.dfki.uni-sb.de`.

B README File of the Current tsct Distribution

Following is the contents of the README file included with the current distribution of tsct version 1.0 (as of June 1995):

```
Some notes on the use of the DFKI Test Suite Construction Tool (tsct):

(version 0.2)

- according to the proposal from 24-oct there are four windows, all of
  them independent of each other; hence, any of the commands ‘save’,
  ‘reset’ and ‘quit’ is only relevant to the current window;

- in the directory where you have unpacked the distribution start up
  the four input windows with the commands

    tsct -rf item.ari -file <file_1> &
    tsct -rf phenomenon.ari -file <file_2> &
    tsct -rf set.ari -file <file_3> &
    tsct -rf profile.ari -file <file_4> &

  where <file_n> is the data file to write; _caution_: do _not_ attempt
  to mix the data files for the four different windows; each of them
  has its own format;

- the tsct executable will accept any of the standard X11 options, e.g.
  in particular the ‘-title’ or ‘-geometry’ directives;

- each of the windows  has its own data file associated with it; the
  data file is determined through the ‘-file’ option: if the file
  exists at startup a backup version (file name with ‘~’) is created
  and new data is appended at the end of the file;

- <tab>, <return> or <down> advance the input focus (cursor) into the
  next field; <shift><tab>, <shift><return> and <up> navigate in the
  reverse direction;

- for the ‘analysis’ and ‘parameters’ tables there are ‘prev’ (ious),
  ‘next’ and ‘delete’ buttons to scroll through the list of entries;
  additionially, <tab>, <return> and <down> in the last field of the
  table (i.e. in the bottom-most ‘comment’ field) act synonymously to
  the ‘next’ button;

- each of the windows has four buttons at the top: ‘save’, ‘save &
  reset’, ‘reset’ and ‘quit’; ‘save’ writes out the current data in the
  window; ‘reset’ resets a subset of the fields (see below) to an empty
  state; ‘save & reset’ combines the two and ‘quit’ terminates the
  window _without_ saving the current set of data;

- for all of the windows there is a solid horizontal bar delimiting the
  attributes that are considered static through one session (e.g. for
  the ‘phenomenon’ window everything from ‘author’ to the first
  ‘comment’) and thus not affected by the ‘reset’ button;

- the data files are written in plain ascii format with a ‘@’ sign
  separating the individual fields; for some basic extraction and
  consitency checking the unix commands sort(1) and awk(1) can be used
  to interprete this format;
```

```
- i recommend to collect the data files in small pieces (and eventually
  con-cat(1)enate them into a master file); e.g. one might choose to
  use a temporary file for the data input and to collect the temporary
  files into a backup version daily;

- for the next version we plan to add `previous' and `next' buttons at
  the top level so that you can use tsct to edit data that has been
  constructed earlier (this, for the time being, requires editing the
  data files in some text editor).

- please send all bug reports and comments to: `tsct@dfki.uni-sb.de'.

                                                    28-oct-94  -  oe

(version 0.3)

- streamlined attribute layout.

- new functionality: in the test item window `instance' is computed
  automagically from the `position' description; valid formats are
  `n:m' and `n:m,x:y'.  _note_ that one or more blanks are required to
  separate words _and_ punctuation from each other, e.g. ``er kommt .''
  rather than ``er kommt.''

- new option `-intelligent' to enable the automagic compution of the
  `instance' field.

                                                    14-nov-94  -  oe

(version 0.4)

- numerous bug fixes.

                                                    25-nov-94  -  oe
(version 0.5)

- numerous bug fixes.

- browsing and editing of data files is now implemented:

  + `next' and `previous' buttons navigate through the data;
  + _at_the_end_ of the list of data sets `new' appends a copy of the
    current set to the list; `new + reset' additionally resets the
    reset part of the new data set (analogously to what `save' and
    `save + reset' did in previous versions);
  + `delete' removes the current data set from the list;
  + `save' writes the current list of data sets to disk;
  + `quit' saves the current list and quits.

- in the `test item' window when using the `-intelligent' option the
  `length' value is computed automagically.

- the file `vg.items' include in this release contains a snapshot of
  verbal government data for german (newer versions of DFKI data are
  often distributed separately through the ftp(1) repository).
```

```
                                                  9-jan-95  -  oe

(version 0.5.1)

- three bug fixes (in the ‘test set’ and ‘profile’ windows).

- ‘Previous’, ‘Next’ and ‘New’ buttons now beep when used in an invalid
  context (e.g. ‘Previous’ when looking at the first data set).

- before adding more functionality this version is defined to be stable
  and frozen to guarantee a functional tsct(1) state.

                                                  13-jan-95  -  oe

(version 0.6)

- no bug fixes.

- new functionality: clicking mouse middle on ‘Next’ or ‘Previous’
  moves by five, mouse right by ten data sets forward or backwards
  respectively.

- new navigation display: each window displays the index of the current
  data set and the total number of sets in the upper right corner.

- new option: ‘-offset n’ will make the n-th data set current on startup.

                                                  13-jan-95  -  oe

(version 1.0)

- several bug fixes: (occasional core(5) with ‘new’ and ‘new + reset’
  buttons; file format bug reported from essex; missing data set
  counter in ‘sets’ window);

- new feature: ‘quit’ no longer saves the current data by default but
  offers the choice of saving or not in a popup window; this, of
  course, is only necessary when data has been modified since the last
  ‘save’.

                                                  24-apr-95  -  oe
```

C The Import Procedure

Following is a listing of the contents of the current distribution of the `tsct2tsdb` import procedure:

```
0 oe@clarinet (~) 137 $ tar ztf tsct2tsdb.tar.z
tsct2tsdb-0.1/
tsct2tsdb-0.1/tsct2tsdb.bash
tsct2tsdb-0.1/phenos-to-tsdb.gawk
tsct2tsdb-0.1/README
tsct2tsdb-0.1/items-to-tsdb.gawk
tsct2tsdb-0.1/sets-to-tsdb.gawk
tsct2tsdb-0.1/extract-embedded.gawk
tsct2tsdb-0.1/common.gawk
```

The distribution is in `gzip(1)`ed `tar(1)` format and available from the TSNLP `ftp(1)` repository at `tsnlp.dfki.uni-sb.de`.

References

[Balkan *et al.* 1994] Lorna Balkan, Doug Arnold, Frederik Fouvry, Dominique Estival, Kirsten Falkedal, Sabine Lehmann (last three: principal authors), Eva Dauphin, Sylvie Regnier-Prost, Véronika Lux, Klaus Netter, Stephan Oepen, and Judith Klein. The Construction of Test Material. Report to LRE 62-089 D-WP3, University of Essex, UK, 1994.

[Elmasri and Navathe 1989] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Redwood City, California (The Benjamin/Cummings Publishing Company), 1989.

[Estival *et al.* 1994] Dominique Estival, Kirsten Falkedal, Sabine Lehmann (principal authors), Lorna Balkan, Siety Meijer, Doug Arnold, Sylvie Regnier-Prost, Eva Dauphin, Klaus Netter, and Stephan Oepen. Test Suite Design — Annotation Scheme. Report to LRE 62-089 D-WP2.2, University of Essex, UK, 1994.

[Nerbonne *et al.* 1993] John Nerbonne, Klaus Netter, Kader Diagne, Ludwig Dickmann, and Judith Klein. A Diagnostic Tool for German Syntax. *Machine Translation*, 8 : 85–107, 1993.

[Oepen and Netter 1994] Stephan Oepen and Klaus Netter. TSNLP — Test Suites for Natural Language Processing. In: *Proceedings of the Conference on Linguistic Databases*, Groningen, The Netherlands, 1994. forthcoming.