Test Suite Design
Annotation Scheme

Lorna Balkan, Siety Meijer, Doug Arnold,
Dominique Estival, Kirsten Falkedal, Sabine Lehmann,
Eva Dauphin, Sylvie Régnier-Prost
Klaus Netter, Stephan Oepen

Report to LRE 62-089 (D-WP2.2)

Test Suites for Natural Language Processing (TSNLP)

University of Essex

September 1994

Summary

Title	Test Suite Design — Annotation Scheme
Project	LRE-62-089
Document ID	D-WP2.2
WP	Test Suite Design
Date	September 1994
Principal Authors	Dominique Estival, Kirsten Falkedal, Sabine Lehmann
Authors	Lorna Balkan, Siety Meijer, Doug Arnold, Dominique Estival, Kirsten Falkedal, Sabine Lehmann Sylvie Régnier-Prost, Eva Dauphin Klaus Netter, Stephan Oepen
Reviewers	the consortium

Contents

Test Suite Design
Annotation Scheme

1 Introduction

This deliverable is the report for Task 2.2 of TSNLP WP2. While the first months of the project were devoted to studying existing test suites and designing a general methodology for test suite writing, this task had the concrete goal of proposing an annotation scheme for the test suite database of TSNLP.

We have drawn extensively on the conclusions of the previous tasks, and especially on the proposals that were suggested to us by the observed inadequacies and shortcomings of most currently available test suites (see Reports D-WP1 [3] and D-WP2.1 [2]).

The work in this task was divided into three steps:

- collect, describe and review existing annotation schemes (in test suites and for morpho-syntactic taggers and other NLP tools or systems);
- design the set of <attribute:value> pairs which constitute the annotations for a TSNLP test suite;
- apply this annotation scheme to the prototype data which had already been created during WP2.1.

The second and third steps were repeated several times to improve the annotation scheme in the light of applying it, and the scheme presented here is the latest at which we have arrived. It is still subject to modifications during the course of WP3 (the WorkPackage in which we write the actual TSNLP test suites for the three languages), but the main ideas are already in place.

The overview of existing annotation schemes is given in section 2. We then turn to a general discussion of the TSNLP annotation scheme and of the motivations underlying some of its specific aspects in section 3.

The bulk of the report consists of a detailed presentation of the proposed annotation scheme, with explanations for each feature and a list of suggested (or only suggestive for some of them) values for each of them in section 4.

We then provide in section 5 the prototype data which has been written according to this scheme. This prototype data consists of a small number of *test sets* (see section 4) for three phenomena: verb valency, NP agreement and passive in English, French and German.[1]

[1]Essex was responsible for the English data, Aerospatiale for the French data and ISSCO for the German data.

2 Overview of existing annotation schemes

To get an overview of existing annotation schemes, the tasks of looking at some specific test suites or application systems likely to use something comparable to an annotation scheme were distributed among the partners in the following way:

- ESSEX: IBM-Spain, CLAWS, SUSANNE;
- DFKI: DITO;
- AEROSPATIALE: GRAAL-EUROCASTLE, EDF;[2]
- ISSCO: GRAMMATIK, HP, CRATER, Neal-Montgomery.

What we asked each partner to do was not an input paper or a detailed description of systems, but a short description of the kind of information they provide. In the case of actual applications or commercial systems, this is the information the system gives the user. In the case of test suites, we wanted to know the kind of information given by the annotations, in particular whether the annotations apply to lists of test items or to each test item, and whether the annotations give information both on the given input and the expected output. Even if the "annotations" turned out to be merely headings, we also collected the lists of these for comparison purposes.

In addition to those listed above, we also asked that each partner check whether there were new sources of test suites (tools or systems) with annotation schemes and report on them if information became available during the course of the task. Indeed, we received information about the **SEMEVAL** project and studied how the proposed annotations on the input data in that project could be compared to what we have in TSNLP (see section 2.8).

We now give the results of the study of these various annotation schemes, first for those used in existing test suites, then for those in morpho-syntactic taggers and last for those found in other NLP systems or applications.

2.1 The Hewlett-Packard test suite

The only kinds of annotations in the HP test suite [4] are 1) the headings under which the test items are divided according to the linguistic phenomena they are meant to test and 2) the star ($*$) indicating that a given test item is ill-formed. The headings for lingusitic phenomena also indicate whether a phenomenon is to be considered part of the "core" of the language or "peripheral". This distinction has been adopted in the TSNLP annotation scheme.

[2] Unfortunately, it was not possible to obtain any information from EDF about the annotations in the test suite designed by Yves Lepage, nor indeed any more information about that test suite.

The list of phenomena derived from the headings in the HP test suite had served as the basis for the syntactic part of the questionnaire on test suites we used in WP1 [3], and D. Estival asked I. Sag about recent developments continuing the work that had been started at Hewlett-Packard. The only thing that can be used for now is an expanded list of linguistic phenomena (both syntactic and pragmatic to a certain extent), which could be used as a better basis for the multilingual list of phenomena to be used in TSNLP (see attribute PHENOMENON in section 4).[3]

2.2 DITO

DITO is a test suite database for German [10]. The DITO test suite is designed as a diagnostic evaluation tool for the recognition and characterisation of problems of syntactic analysis. Test items are both systematically constructed and annotated with information about the syntactic phenomena illustrated.

In fact, one of the most original aspects of the DITO test suite is its detailed annotation scheme. Not only are linguistic phenomena annotated for, but the annotation scheme also contains information about the precise structure of the sentence such as the position of the finite verb and the position of other phrases. The aim was to focus on properties that are relatively easy to test and about which there is little theoretical dispute, but that nevertheless provide a reasonable reflection of a system's accuracy.[4]

The DITO test collection is of major importance for TSNLP, precisely because of its uniquely (compared with other test suites we have looked at) explicit annotation scheme. Therefore, a number of the features found in the DITO annotation scheme have been taken up in TSNLP, in particular:

- position of elements in the input string;
- length of the test item;
- attribute indicating whether the item is well-formed;
- category and function of the elements in the input string;
- attributes specifically meant to accommodate comments (at various levels in TSNLP);

However, DITO is specifically designed for German and is based on a particular grammar, while the TSNLP annotation scheme is meant to be language-independent and more general, and to be used for a number of applications.

[3] We have adopted the typographical conventions of using capital letters for attributes and italics for attribute values.

[4] Also important is the fact that not only the constructions, but also the problems known to be difficult in their analysis were catalogued. This catalogue will be used as one of the bases for building the multilingual list of phenomena in TSNLP.

Thus, a number of other attributes must be introduced (e.g. language, multilingual links, output specifications according to the application, etc.) and certain aspects of the test suite design have evolved. For instance, while DITO chooses to explicitly describe only the features of the test item which are relevant for the phenomenon being tested, TSNLP also provides a general description of the test item (TI-ANALYSIS) in addition to the features specific to the phenomenon (TI-PARAMETER for well-formed test items and TI-ERROR for ill-formed test items). The reason for this extra information is to facilitate extraction of test items based on various properties and matching of test items to already tagged texts or corpora (see section 3.2.1). In addition, viewed as a "pattern", this information may allow the automatic derivation of new test items.

2.3 IBM-Spain

For testing the performance of the system developed in their Machine Translation project, IBM Spain uses a collection of sentences from manuals, dictionaries, grammar books and others sources (including artificially constructed sentences) [1]. The CDBM tool for building and managing these collections allows researchers a ready and selective access to the sentences. For this purpose, sentences are marked with a set of labels showing the linguistic features they contain.

The label taxonomy is a tree providing a classification of linguistic phenomena. No other information is provided in the annotations. The taxonomy of phenomena can be used in drawing the list of multilingual phenomena for TSNLP, especially since it breaks them down into sets and subsets and is very detailed.

2.4 The Neal-Montgomery Method

The Neal Montgomery Method [8] (see [2] for a more complete description) is an evaluation procedure that aims to help an evaluator create, modify, or tailor test sentences for a particular NLP system, and, in addition, score its performance. It consists of a categorisation of linguistic features (lexical, syntactic, semantic or pragmatic) requiring processing capabilities, along with instructions for the formation of suitable domain-relevant sentences with these features and for scoring system output for this test material.

Each identified linguistic feature is defined and explained in order to establish a standard delimitation of that feature. Illustrative language patterns and sample sentences are provided to assist the human evaluator in formulating input which tests the feature on the NLP system within the system's native domain.

The classification scheme assumes a blackbox approach to evaluation, thus a phenomenon such as anaphora is not further divided into lexical, syntactic, semantic, and discourse aspects. However, the classification scheme is tree-structured and includes pointers to other sections of the classification scheme for classes that could be classified under multiple parents. Each test item includes:

- an explanation and a definition of the linguistic phenomenon or capability being tested, along with any special instructions for testing;
- patterns/descriptions that define the structure and features of the test sentences to be composed and input to the NLP system under evaluation;
- example sentences to aid the evaluator in composing test sentences;
- a statement of the evaluation criteria for the individual test item;
- a place for the evaluator's test sentences;
- a place for the evaluator's score.

The Neal Montgomery Method is inspirational to TSNLP in the sense that it attempts to classify a wide range of linguistic phenomena for evaluation purposes, with an emphasis on making these phenomena as explicit as possible. Thus a number of aspects of the Neal Montgomery strategy for test suite construction are useful to TSNLP. In particular, the hierarchical structure of the classification scheme is taken up in our annotation scheme (see PHENOMENON/TYPE/SUBTYPE in section 4).

The progression from easy examples to more complex ones, by varying one parameter at a time, is reproduced in the TSNLP database and is expressed in the TSNLP annotation scheme by the attributes PRESUPPOSITION, INTERACTION and also by TS-DIFFICULTY (see section 4).

The expression of patterns out of which the evaluator constructs particular test sentences is underlying the TI-ANALYSIS, TI-PARAMETER and TI-ERROR attributes in TSNLP. However, while the expression of patterns is the basis of the Neal Montgomery method, it is not a requirement for the TSNLP database: in TSNLP, the structural analysis of a test item is given so that lexical replacements can be performed and new test items may be derived, but a test item may be used as such without requiring the test suite users to perform lexical instantiation themselves.

There is no attribute directly expressing the evaluator's score in TSNLP. What we have instead is a number of pre-defined possibilities for expressing what the output of the system should be (see TI-OUTPUT in section 4), which allow test suite users to specify elements that will be part of their evaluation. Moreover, we assume that the values which can be stated in the input annotations for FREQUENCY and WEIGHTING can serve as a basis for calculations of scores.

2.5 CLAWS

The CLAWS tagset is used by a morpho-syntactic tagger [5]. We have only been able to use it as an example for the possible tags at the word level.

2.6 CRATER

This Spanish tagset is described in [13], and is derived from the one used by CLAWS [5]. It makes distinctions between all the major morphosyntactic categories and also some syntactic and semantic distinctions beyond the level of morphosyntax. It is meant to be used by an automatic tagging system and the aim of the CRATER project is to produce a tagged corpus of over one million words. The two goals of reusability of resources and robustness of the system have thus lead on the one hand to extension of the CLAWS tag set to "include all the feature specifications proposed by TEI and/or EAGLES", and on the other hand to its simplification by elimination of certain tags for forms which could not be disambiguated.

The result is a quite complete set of tags, which could serve as a basis, or even as a model, for tagsets we might envisage as one of the attributes in the annotation scheme (see section 4). But it cannot serve as the basis for the annotation scheme itself, as none of the information we require is envisaged.

2.7 SUSANNE

The SUSANNE annotation scheme was developed as a parsing scheme for a corpus (the SUSANNE corpus [12]) and was later overtaken by other projects including, most notably, the Penn Treebank project [7]. The aim was to provide better statistics for probabilistic automatic parsing techniques. The project thus requires a very consistent and explicit analytic scheme, and the analysis given by SUSANNE is always the most neutral possible with respect to alternative linguistic theories.

The annotation scheme is based on the Lancaster-Leeds Treebank and extends it by including methods for representing logical grammar and by refining both its surface and logical aspects. Each word in the input is annotated by six fields:

1. reference
2. status
3. wordtag
4. word
5. lemma
6. parse: each node bears the following labels:
   a. indices
   b. formtags
   c. functiontags

All of this information is very important for TSNLP, and even though the way it is presented in TSNLP differs, it is nevertheless fully covered by our annotation scheme (see section 4).

The *parse* field corresponds to our TI-ANALYSIS attribute, in which *indices* and *functiontags* have direct equivalents.

The *wordtag* information is (optionally) provided in the TI-TAG attribute.

Neither the *lemma* nor the *formtag* fields have direct equivalents in the TSNLP annotation scheme, because lexical information in TSNLP, such as the lemma for a word and its lexical properties (*formtag* encodes "the internal properties of the word or word-sequence dominated by the node bearing it") is assumed to be provided by an independent lexicon accessible through the tools provided with the database. In the absence of such a lexicon, this information concerning the relevant lexical items can still be stated in the attributes ATTRIBUTE of TI-PARAMETER and TI-ERROR.

2.8 SEMEVAL

SEMEVAL is an initiative to explore the appropriateness of existing English-oriented guidelines for languages other than English, and to provide input needed for possible decisions on European annotation efforts. The current specifications have been developed by Ralph Weischedel on the basis of the notation developed by Bob Moore and Jerry Hobbs for predicate-argument structure. Predicate-argument structures generally reflect "deep" or "functional" syntactic relations and this representation can thus been seen as similar to lexical-functional structure (the f-structure of LFG) or a dependency structure with functional role labels. As the name of the project indicates, these representations are mostly semantic, and the set of annotations which has been suggested is very different from those in other systems or projects.

S. Meijer has actually tried to apply the set of annotations provided in the SEMEVAL project during a class she attended at ESSLLI'94, but the conclusion is that the assumptions which serve as the basis for these annotations are so different from those of the TSNLP project (who will use them? for what purpose? what kind of information should they provide? etc.) that at best they can help us think about the kind of semantic information that might be needed on each test item. However, they cannot serve as the basis for any specific attributes in the TSNLP annotation scheme.

However, another (besides the contribution to semantics) interesting aspect of the SEMEVAL annotation scheme is the fact that it is written in SGML. While we have not yet decided what the formal language representation for TSNLP annotations should be, this would obviously be one of the best choices for us to make.

2.9 GRAMMATIK

GRAMMATIK is a commercial grammar checker [6], and we looked at the French version of this tool and studied its accompanying documentation. The types of errors which are reported and the way they are formulated to the user should serve as inspiration for the information we want to provide in the OUTPUT sections of the TSNLP annotation scheme for one of the applications envisaged in TSNLP, i.e. Grammar Checkers (see section 4). They cannot, however, serve as the basis for the other parts of the TSNLP annotation scheme.

2.10 GRAAL-EUROCASTLE

The GRAAL-EUROCASTLE SE (Simplified English) Checker is a controlled language checker developped within the EUREKA "GRAAL" project. It processes AEROSPATIALE's maintenance manuals (technical documentation) in order to produce a report of all possible errors according to SE rules. As an example, a sentence fragment such as *... can be analysed by...* will generate 4 different diagnoses:

(1) type morphology: UK form (instead of US) of the word "analyse",

(2) type lexicon: use of an unapproved word "analyze",

(3) type morpho-syntax: unapproved verbal complex form "can be analyzed",

(4) type syntax: unapproved syntactic construction (passive).

The types of errors which are reported by this tool (and the precise list of errors itself, if it is available) might be useful in designing the set of error reports for one of the applications envisaged in TSNLP, i.e. Controlled Language Checkers. They cannot serve as the basis for the rest of the annotations in the TSNLP database.

2.11 Concluding remarks

This overview of existing annotation schemes from various sources only served to demonstrate what we already knew after the work done during WP1 (the survey of existing test suites), that there were no equivalents to what we have set out to do in this project.

However, from each of these sources, we were able to take some inspiration and we feel confident that no information that a system has considered useful to provide is missing from the TSNLP annotation scheme.

Study of systems such as GRAMMATIK and GRAAL-EUROCASTLE have made it obvious to us that the formats for stating expected outputs (whether in a "formal" or an "informal" way) have to be left unspecified since the optimal notation may differ radically from one application to another, even among applications of the same type. This work will be done in WP3 and

WP4 when we actually write the real test data and test them on specific applications.

Moreover, the contacts we have had with the SECC project have also contributed to a better understanding of the type of annotation a user might want to put in the output section for the specific application type of Controlled Language Checkers.

3 The TSNLP Annotation Scheme

The need for annotations in a test suite was outlined in the Technical Annex and, after the study of existing test suites done during WP1, a number of requirements had already emerged.

In particular, it had become clear that, besides the general information which is needed for identification of the test suite itself (e.g. language, author, institution, date of creation, available documentation, etc., see [3]), and the classification of the linguistic phenomena addressed by the test items, which is the kind of information most test suites limit themselves to when they have any type of annotations, there are two types of annotations which test items in a test suite should carry in order to be useful for testing.

1. Annotations on the input data (describing the test item):

   These annotations must both be based on linguistic descriptions so as to ensure reusability and consistency throughout the database, and contain explanations understandable and usable by non-linguists.

2. Annotations describing the expected output:

   While defining these annotations has been rather difficult, since this aspect of test suite annotation is not present in any existing test-suite, it has proved even more challenging to try and instantiate them in the prototype data.

   The information in these annotations depends on the different types of applications, and we have limited ourselves to the three applications that must be addressed in the rest of the project:

   - **Parser**: the output of a parser is usually a parse tree, and the description of the expected output in the test suite could also be a parse tree or a representation based for instance on the Penn Treebank.
   - **Grammar Checker**: the output is of two sorts:

     - description of the error: this is difficult because we have to design a way of describing errors to the users, but we try to use the work done in the EAGLES and TEMAA groups, as well as the types of response given by existing grammar checkers.[5]

     - advice: for TSNLP, we must at least suggest what a system might reasonably be expected to say.
   - **Controlled Language Checker**: the output is the same as that for grammar checkers, i.e. a description of the error (in relation to the Controlled Language specifications, and advice on how to correct it).

[5] The formal description of the error in the annotations on the input data was of course also difficult to design, but the fact that it does remain at a formal level made the task somewhat easier.

To illustrate the extensibility of the annotation scheme to other types of applications, we have added an example of attributes describing the expected output of an MT system (see section 3.1).

The annotation scheme at which we have arrived also tries to address a number of other issues: the multi-linguality of the test database, the progressive and systematic organization of the test items and the derivation of ill-formed inputs.

We first give the empty annotation scheme with possible values for each attribute in 3.1, then we discuss the set of motivations underlying the annotation scheme in 3.2.

3.1 Empty Annotation Scheme for TSNLP

The TSNLP database is conceptually organized in at least two levels:

- Test Suite level: one test suite for each language.
  A test suite may be divided into groups of test items (or modules), which can be organized along parameters defined by the phenomena under consideration, the phenomena assumed to have already been tested and other common characteristics.

- Test Set level:
  A Test Set contains one positive input and one or more negative input. The relation between positive and negative test items is one of the most difficult questions addressed by the whole project, and is taken up several times in different places in this report.

In the following presentation, we have adopted the typographical convention that, when they are given, the values for the attributes are written in italics.

TEST SUITE
language *de/fr/en*

TEST SUITE MODULE
tsm-id
phenomenon
type
subtype
phenomenon-code
presupposition
interaction *none/list of phenomena*
tsm-comment

TEST SET
ts-id
ts-info
    author
    date
    purpose *test/evaluation/ ...*
    restrictions *neutral/GC/CCL/P/...*
    centrality *core/peripheral*
    relevance
        parser *central/marginal/...*
        grammar checker *central/marginal/...*
        controlled language checker *central/marginal/...*

user specifics
  system 1
    text type *spec of text type*
    frequency *scale*
    weighting *scale*
  system 2
  etc..
origin *invented/corpus/derived from corpus*
register *formal/informal/scientific/...*

ts-category *S/NP/...*
ts-format *none/header/table/graph/list/...*
ts-difficulty *1/2*
ts-comment -

TEST ITEM
ti-id
ti-input
ti-wellformedness *1/2*
ti-length
ti-analysis
 pos cat li function fd
ti-tag
 LI tag (X)
ti-parameter
 pos attribute value LI

ti-output
  Parser
    informal
    formal
      system 1
      system 2
      ...
  Grammar Checker
    informal

  Controlled Language Checker
    informal

  Machine Translation system
    target language 1
      informal
      formal
        system 1
          pref 1

          - pref 2
          - ...
        - system 2
          - pref 1
          - pref 2
          - ...
        - ...
    - target language 2
    - ...
  - output-comment
- ti-comment-

TEST ITEM

ti-id

ti-input

ti-wellformedness *0/2*

ti-length

ti-error

pos	attribute	value	LI

- ti-output
  - Parser
    - informal
    - formal
      - system 1
      - system 2
      - ...
  - Grammar Checker
    - informal
    - formal
      - system 1
      - system 2
      - ...
  - Controlled Language Checker
    - informal
    - formal
      - system 1
      - system 2
      - ...
  - Machine Translation system
    - target language 1
      - informal
      - formal
        - system 1
          - pref 1

pref 2
...
system 2
pref 1
pref 2
...
...
target language 2
...
output-comment
ti-comment-

3.2 Considerations underlying the TSNLP annotation scheme

One of the most important innovations in the TSNLP approach to test suite construction is the ambition to take the step from "general purpose" test suites to "multi-purpose" (and multi-lingual) test suites by providing a database format for test data which allows for storing and retrieval of both general and application specific test data in several languages.

The TSNLP test suite is thus not only designed to accommodate a set of test data assumed to be relevant to a range of applications, but also to accomodate test data which are necessary to make a test suite interesting for specific applications or even to fit it to specific user requirements.

In order to achieve the desired flexibility, each individual test item must be associated with a set of annotations which provide detailed information on various aspects of its contents, status and appropriate use.

The annotation scheme we have arrived at during this work package is discussed in detail in section 4. It should first be noted though that the presentation of this annotation scheme as given in section 3.1 above is paper-oriented: it uses headings, lay-out structure and typesetting fonts as a way to specify the hierarchies, the relations and the values which will later be translated into a relational database format. For the construction of the test suite fragments in WP3, we envisage to provide a template which will minimize the amount of work needed to import the data into the database once it has been constructed.

The order of the annotations in the annotation scheme in its present shape reflects to a certain extent the most natural order for filling in the scheme. What does not appear explicitly from looking at the scheme is the number and types of considerations which have led to the inclusion of individual annotations. When discussing these annotations in detail in section 4, we also indicate briefly the purpose(s) for which each annotation has been included. In the remainder of this section, we give a more systematic overview of the types of underlying considerations which have led to the present selection and structure of the annotations.

These considerations fall into two broad classes:

- considerations concerning the future uses of the TSNLP test suite database and its accompanying tools;
- considerations concerning the creation of test data, the structure of the test suite and the control of its contents.

We briefly discuss these sets of considerations below. It should go without saying that most of the individual annotations are motivated by a combination of considerations. We will not, however, discuss all the annotations motivated by one consideration but rather concentrate on the most representative examples for each of these considerations.

3.2.1 Considerations concerning future uses of the test suite

As reusability is one of the major concerns of the TSNLP project, this class of considerations is the most obvious. It can be further subdivided into considerations concerning:

- retrieval of test suites along various attributes or combinations of attributes;
- description of the individual test items, as information to the user of an extracted test suite;
- provision of annotation types where users can indicate specific values.

Retrieval

It is a consequence of the multi-purpose approach to test suite design that reuse of the test data implies extraction of (partial) test suites based on some user defined set of annotations. In principle, any annotation or set of annotations can be used for extraction of test data for some specific purpose, and one of our concerns has therefore been to cover as wide a range of imaginable purposes as possible.

Some of these purposes for which users might want to extract test suites have only been imagined rather vaguely whereas others are central to the project, e.g. extraction of test suites to test the three applications envisaged in TSNLP: parser, grammar checker and controlled language checker. Each test item must therefore, directly or via links to other test items, be annotated for whether it is considered generally relevant to all applications or only relevant to one or more specified application type. This information is given by the attribute called RESTRICTION in the annotation scheme.

Furthermore, it must be envisaged that test suite extraction based on application type might need to be further restricted either with respect to the coverage of linguistic phenomena or with respect to the importance or centrality of the test data relative to some user specific requirements. For this latter purpose, we have introduced two complex attributes, RELEVANCE and USER SPECIFICS.

RELEVANCE makes it possible to indicate the general relevance of a test item for each application type. Indeed, the same test item may be central to the testing of a grammar checker or a controlled language checker but marginal to the testing of a parser, and vice versa.

Another type of restrictions can be expressed by the attribute which, for lack of a better name, is called USER SPECIFICS. This attribute has been introduced to allow specific users who reuse a test suite to extract partial test suites based on indications of measures of frequency or weighting of the test items (e.g. test suites concentrating on highly frequent and/or highly weighted phenomena in some corpus or specific text type).

It should be noted in passing that the option to annotate for relevance and user specific frequency and weighting information is also motivated by a concern to make the results of applying a test suite describable and interpretable if the test suite is used for evaluation purposes. Although interpretation of results is not an explicit part of the TSNLP project, the very notion of a test suite implies that it should be an efficient tool for testing and comparing systems for their treatment of phenomena of special importance to a user.

Two further types of retrieval which we foresee deserve special mention here, namely those provided for by the attributes TI-TAG and TI-ANALYSIS.

TI-TAG has been introduced as a consequence of the concern to enable users to extract partial test suites which represent frequently occurring or highly weighted phenomena or structures in specific corpora or text types. As tools for tagging and for statistical calculations of occurrences of tag patterns will become more readily available and standardized, we found it desirable to be able to retrieve test items and test suites by searching on their tag patterns. This is, however, an option for the future and no tagset will be provided by the TSNLP project.

The attribute TI-ANALYSIS is mainly meant as a linguistic specification of the composition of a test item and it allows the retrieval of test items on the basis of specific constituent types, positions or lexical instantiations. It can, however, also be considered as an object of retrieval in its own right. Provided that a test suite has a sufficiently interesting coverage of some phenomenon, of a set of phenomena or of a language, the sentence patterns represented in the TI-ANALYSIS attribute of its test items can be seen as a kind of linguistic specification for the structures which some, or all, NLP systems have to master.

The relatively shallow level of representation proposed for TI-ANALYSIS could thus serve as a basis for (semi-)automatic acquisition of grammars for shallow parsers. A deeper analysis would, of course, widen the scope of exploitation to more ambitious NLP systems.

Exploiting this side effect of the annotation scheme falls beyond the goals of the TSNLP project but as we find this perspective important, ensuring that the information provided in TI-ANALYSIS is organized so as to support this type of retrieval, has been a major concern in designing the detailed structure of this attribute.[6]

Description and information

The TSNLP test suite is intended for different kinds of users with varying degrees of linguistic training and familiarity with grammatical and linguistic terminology. Moreover, they will be native speakers of different languages. A specific set of considerations must therefore go into ensuring that all types of users can make optimal use of the database.

[6]Note that this information could also have been provided in a separate attribute, e.g SENTENCE PATTERN, but this solution is at the moment considered too redundant.

One part of these considerations is not visible in the annotation scheme as presented in this report, since it is written in English and the names for the linguistic phenomena as well as for all the attributes we are describing are also given in English. We do, however, envisage to provide names for all attributes and values used in the scheme in each of the languages covered by the test suite, and to establish equivalences for these names (see section 4.2).

The amount of description given in the annotation scheme for each test item should in itself enable any user to understand rather easily what a test item contains and what it is meant to test. It may, however, be desirable to give further information on specific phenomena, reference to standard grammars, to give advice on how to customize or extend parts of a test suite, to make user specific comments etc. To this purpose, attributes for COMMENT are included at each level of the annotation scheme. Ideally, the test suite should be implemented as a hypertext, where the user could click on any attribute, value or lexical item displayed for a test item and get supplementary information.

User specific annotations

The attributes USER SPECIFICS and TI-TAG have already been mentioned as attributes where user specific values can be provided, and which would serve, among other things, for retrieval purposes.

However, an often encountered experience is that, at the other end of the testing process, the analysis of the results of applying a test suite is a time consuming and cumbersome task. Users and evaluators therefore often express the expectation (or even the requirement) that the test items in diagnostic evaluation tools such as test suites should be associated with (partial of full) specifications of expected output, preferably in a format which allows automatic comparison with the actual output from the system under test. The analysis task for the user would then be reduced to analyzing the test items where expected and actual results differ.

In the general case, i.e. without having a specific system in mind, precise specification of expected output is difficult or even impossible to provide. First, NLP systems for different applications differ as to the type of output they produce:

- parsers produce some kind of representation (usually a tree or graph, or both) for well-formed test items and a failure message or a partial parse for ill-formed items;
- grammar checkers and controlled language checkers produce nothing for well-formed test items and error messages and advice for ill-formed items;
- widening the scope of application types envisaged here, we know that

MT systems produce translations for well-formed items and, if they are robust, for ill-formed items as well.

The annotation for output specifications must therefore be divided at least into separate attributes for different application types.

Secondly, as was described in section 2 (the study of existing annotation schemes), even systems belonging to the same application type differ in the way they present their results: representations produced by different parsers are displayed and labelled differently, grammar checkers and controlled language checkers formulate their error messages and advice in different ways, etc. Thus, even in cases where automatic comparison is not doable, it would be desirable to be able to specify expected output as closely as possible to the way actual output is presented by specific systems. This would at least ease the task of the user who performs the comparison.

In certain scenarios for use of test suites it is, however, possible to give precise specifications for the expected output. The most typical example is that of a developer who uses test suites to measure the progress and/or deterioration after having made changes or extensions to system grammars or lexica. In this case, the output produced by the system, be it correct, corrected or incorrect, can be stored as the output expectation and later used for automatic comparison whenever a test item is reused.

The concern that the output specifications be as usable as possible has led to the introduction of a distinction between INFORMAL and FORMAL output annotations, where INFORMAL specifications are reserved for human use, whereas FORMAL is reserved for (partial or full) specifications for automatic comparison. In the annotation scheme presented above we have not subdivided the INFORMAL output specification as to specific systems, as we have done explicitly for the FORMAL specifications, but such an extension can easily be introduced.

3.2.2 Considerations concerning the creation, structure and control of the TSNLP test suite

Two main principles have been adopted to guide the development of the TSNLP test suites. First, they must be progressive, that is, they must be structured in such a way that each test item contains only one new (sub-)phenomenon. Second, they must test systematically for negative (ill-formed) input. In addition, they must be multi-lingual which means that, as a minimum, corresponding phenomena in the languages involved must be linked in a systematic way.

In order to ensure that these requirements are also met by the partial test suites which can be extracted from the TSNLP database, and that they are also respected by later extensions and modifications to the database or to test suites based on it, two types of considerations must be reflected in the annotation scheme, namely considerations concerning:

- means to state and control the *coverage* of the test suite;
- means to state and control the *progressive structure* of the test suite.

Means to state and control coverage

It became clear to us during our studies of existing test suites in WP1 that stating the coverage of a test suite with respect to its breadth and depth is far from straightforward, especially if the test suite merely consists of sets of sentences collected under some general heading and with a $*$ to indicate that a test item is ill-formed.

Our annotation scheme has therefore been developed in such a way that it allows the test suite writer - and later the test suite user - to state in detail which phenomena and combinations of phenomena the database or any partial test suite covers. For partial test suites, it is possible to state which phenomena are not covered, by doing a simple comparison with the full list of phenomena covered by the database. What cannot be easily stated is which phenomena are not covered by the database, unless we explicitly provide a list of the lacunae of which we are aware.

The phenomenon which is explicitly being tested by a specific test item is indicated by the attributes PHENOMENON, TYPE, and SUBTYPE. However, as soon as two words are put together, several phenomena may already be at play.

Stating and controlling the coverage of a test suite therefore also implies the necessity to ensure that no test item contains phenomena which are not treated explicitly in the test suite (and, in the progressive order of test items, before the phenomenon in focus).

This control of the coverage of the test suite, which is at the same time a control of the structure of the test suite, is performed by the attribute PRESUPPOSITION, for which all the phenomena contained in a test item other that the one in focus (or those for which interaction may be tested) must be listed. The exercise of analyzing candidate test items this way has already proved a useful way of extending and deepening our provisional lists of phenomena and sub-phenomena.

The breadth of a test suite annotated this way can thus be stated in terms of its coverage at the level of the PHENOMENON attribute. The depth of a test suite can, however, not be fully described by listing the (number of) TYPEs and SUBTYPEs for which test items have been provided since the notion of depth of coverage also includes the type and amount of ill-formed or negative test items.

As stated above, it is one of our guiding principles to provide test items for systematic testing of a system's behaviour when it is confronted with ill-formed input. In order to ensure that ill-formed test items are constructed systematically whenever possible, the annotation scheme operates with the notion of "test set" where a test set is defined as a group of one well-formed test item plus one or several ill-formed variants. Test sets are discussed in detail in section 4 below, so we concentrate here on the underlying motivation for them.

As the annotation scheme is presented in this report, it might appear that test sets are the only acceptable units of a test suite. Test sets are, however, no more than a pragmatic format for annotating test items which share a number of attributes and are related to each other as positive and negative tests items for the same (sub-)phenomenon. Thus they are test items which must be linked to each other when they are stored in the relational database so they can be easily retrievable together as a set.

The test set structure is not meant to preclude the inclusion of unpaired well-formed or ill-formed items in the test suite, we simply assume that in that case, a subset of the test set annotations will suffice for such items. We actually need to keep a clear distinction between considerations concerening the development of the test suites and the way they ultimately get stored in a relational database.

In putting an emphasis on test sets during the development of test suites, our goal is not only to control the systematic provision of ill-formed test items for each phenomenon covered by the test suite but also, and more importantly, to check that the ill-formed test items we provide do actually test for the same phenomenon as the one focussed on by the well-formed item in the set, and nothing else.[7]

Although limited, our experience from constructing prototype test data has clearly shown that the operations standardly used to derive ill-formed test items from well-formed ones (modification of an element, addition or deletion of an element, and scrambling of elements in the item) all too easily lead to a confusion of the phenomenon being tested and the test item used to test it. The result of this confusion may thus be the derivation of ill-formed test items which cannot be said to test for the same phenomenon as the well-formed item from which they were derived.

If not strictly controlled by being bound to phenomena via test sets (as opposed to being bound to test items which may contain several other phenomena), the number of ill-formed test items which can be derived from the well-formed test items in a test suite is potentially unlimited. Many of these ill-formed items will however turn out to test for the same phenomena and thus only create undesirable redundancy and extra work for the user analyzing the results of applying a test suite. Others will be undescribable as to what they test for and thus of dubious value in a test suite except if classified as tests for the phenomenon *nonsense*.

Pushing the test set approach to its limits, at least during the initial phase of development, seems to be the best way to control redundancy and to identify the types of (and reduce the number of) ill-formed test items which are interesting in their own right and so deserve inclusion in a test suite.

[7] A phenomenon may of course also be some specific combination of phenomena.

Means to control the progressive structure of the test suite

The annotations which allow to state and control a progressive ordering of the test items with respect to the number and type of phenomena contained in them is discussed in detail in section 4 and is not further elaborated on here. What remains to be investigated is whether the place of each test item in this progression has to be stated explicitly by an extra attribute (e.g. in terms of a number indicating its level in complexity) or whether this can be calculated automatically on the basis of the list of phenomena given in the attribute PRESUPPOSITION. For now, it seems that this question can only be answered in connection with the specification of the database.

3.2.3 Concluding remarks

The different types of considerations are all reflected in the annotation scheme as it stands. It is, however, a consequence of the sequencing of the tasks in the project that the scheme during its elaboration has only been tested on a limited amount of rather simple test data. We thus foresee that it might undergo changes in the light of empirical experience of using it and in the light of database construction and management considerations. In both cases, the underlying considerations discussed in this section (and potential additional considerations not yet encountered) will have to serve as a framework for making decisions.

4 Description of each attribute

4.1 Attributes at the TEST SUITE level

In the annotation scheme, TEST SUITE is only a heading. It is not an attribute.

- LANGUAGE:
  Since TSNLP is meant to be multilingual, we need an attribute to indicate to which language test items belong. The possible values so far are *en, fr, de*. This attribute is at the topmost level to ensure that all test items belonging to one language are grouped together.

4.2 Attributes at the TEST SUITE MODULE level

A test suite module is a group of test items which test the same (sub)phenomenon (PHENOMENON - TYPE - SUBTYPE) and which share the same presuppositions (PRESUPPOSITION). Test suite modules may be formed either by test items which differ at a level of detail not found sufficient to constitute a subphenomenon, or by test items resulting from lexical customizations performed in order to match the vocabulary of some specific domain.

The test items may be grouped together because of distinctions not made at the (sub)phenomenon level, or because they are derived from each other by lexical instantiation for customization of the test suite.

In the annotation scheme, the string TEST SUITE MODULE is only a heading. The attributes collected under this heading are:

- TSM-ID
  TSM-ID is the identification number of the test suite module. This attribute exclusively serves database management purposes.

- PHENOMENON - TYPE - SUBTYPE
  The PHENOMENON attribute specifies which phenomenon is being addressed by the group of test items. If users need fine-grained divisions for a particular phenomenon, they can give further specifications by using TYPE and SUBTYPE, SUBSUBTYPE, etc. The number of test items belonging to the test suite module is clearly related to the description of the phenomenon: the more detailed the description is, the smaller the number of test items is.

  The values for these attributes must come from the list of phenomena which has to be built for every language and related to phenomena in other languages. (This task is addressed in WP3.)

- PHENOMENON-CODE
  This attribute is a link between the test suites for the different languages and is needed for the multilingual aspect of the database. It is

helpful to have the names of the (sub)phenomena available in all three languages. PHENOMENON-CODE relates these names to each other by a code which is language independent. For example:

ENG	DE	FR	CODE
passive	passiv vorgangspassiv	passif	PAS VOR:PAS

So the annotations would be:

phenomenon *passiv*
type *vorgangspassiv*
subtype *A*
phenomenon-code *A:VOR:PAS*

The existence of this attribute has consequences for drawing the list of phenomena for the three languages, since we should keep these lists as similar as possible. Of course, there are subphenomena (or even phenomena) which exist only in one language, such as for example the type "A" of the "vorgangspassiv". The code for a subphenomenon should therefore express the hierarchy, i.e. it must express which phenomenon(-a) it belongs to. The list of phenomena is attacked in WP3, thus the possible values of PHENOMENON-CODE is still an open issue.

Thus, the PHENOMENON-CODE of the example above (A:VOR:PAS) points out that the subtype "A" belongs to VOR which itself belongs to PAS. This way of expressing the code has two advantages: first, we avoid ambiguities which might arise if two subphenomena have the same name. Furthermore, it allows going up in the hierarchy of the subphenomenon.

This latter aspect will be necessary for making connections with test suite modules for another language: for instance, we might want to access the English test suite modules which test for the same subphenomenon (still A:VOR:PAS), but since not all languages have this subphenomenon, we can go one level up (VOR:PAS) to check whether there is a corresponding (English) test suite module. If this is not the case, we can go to the topmost level in order to check again whether there is an English test suite module which has a corresponding code. In our example, there is actually an English test suite module with code PAS.

One could argue that the values of PHENOMENON, TYPE and SUBTYPE are redundant since the value of PHENOMENON-CODE contains the same information, and that we do not need these three attributes. However we keep them here, because they are much more user-friendly since the codename consists only of an abbreviation.

However, we still might need another attribute to provide more specific multi-lingual links, if these codes for phenomena turn out not to be enough for other kinds of linking between the test suites for the different languages that test suite users might want to perform.

- PRESUPPOSITION
  This attribute lists the (sub)phenomenon(-a) other than the phenomenon under consideration which should have been tested or can be considered as already tested when using the test item, because these phenomena are present in the test input, but are not the subject of testing in this test item.

  This attribute allows a greater control over the actual coverage and over the structure of the test suite.

  When creating test items, the fewer presupposed phenomena there are, the more precise the testing for the phenomenon under consideration can be. Thus, when creating test items, it is essential to keep the number of presuppositions as low as possible.

- INTERACTION
  This attribute indicates whether the test items expressly treat interaction with other phenomena than the one under consideration. Note that this attribute depends on the *intention* of the test suite builder, since all sentences have some amount of interaction. The point is whether this interaction is being tested on purpose or not.

  The value is either *'none'* or a list of the additional phenomena. In that case, the phenomena mentioned in INTERACTION must be different from the phenomena listed in PRESUPPOSITION.

  The (sub)phenomena given by the attributes PHENOMENON, PRESUPPOSITION and INTERACTION (if any) must exhaust all the (sub)phenomena observable in the test input.

- TSM-COMMENT
  This attribute allows the test suite writer to make comments in an informal way.

4.3 Attributes at the TEST SET level

TEST SET is only a heading. It is not an attribute, it is only a way of stating relations between test items when writing them. In the actual database, the test items may be organized in a flat structure.

Each test set contains several test items, one positive (or well-formed) test item and at least one negative (or ill-formed) test item, derived from the positive item. The number of negative test items correlates with the number of attributes contained in TI-PARAMETER (see below). The more attributes TI-PARAMETER contains, the larger the number of negative test items.

- TS-ID
  identification number of the test set.
- TS-INFO
  This is only a heading which groups together a number of attributes giving information about the creation and applicability of the test set.

  This information is given at the level of the test set and not at the test suite module level, because test suite writers may write new test sets in an already existing test suite module, and thus must be able to specify new information concerning author, date, restrictions, comments etc. at this level.

  - AUTHOR
    Name of the author/institution who created the test set in question.
  - DATE
    Date of creation of the test set.
  - PURPOSE
    This indicates what the test set is used for. Following EAGLES and TEMAA, possible values are: *test, evaluation*, etc. It is debatable whether this information is really needed.
  - RESTRICTIONS
    The sub-divisions in RESTRICTIONS are meant to account for the fact that the same test set (actually the positive test item) may differ in frequency and weighting between one text type (or corpus) and another, as well as between one application and another.
  - CENTRALITY
    This attribute indicates whether the phenomenon addressed by the test set is considered to belong to the core of the language. We can still refine the values for this attribute: *core, peripheral.*
  - RELEVANCE
    This attribute takes as its value a set of attribute:value pairs where the attribute carries the name of an application and the value states the relevance to that application.
    * PARSER
      The values *central, marginal, etc* indicate whether the phenomenon addressed by the test set is central with respect to a parser.
    * GRAMMAR CHECKER
      The values *central, marginal, etc* indicate whether the phenomenon addressed by the test set is central with respect to a grammar checker.
    * CONTROLLED LANGUAGE CHECKER
      The values *central, marginal, etc* indicate whether the phe-

nomenon addressed by the test set is central with respect to a controlled language checker.

- USER SPECIFICS
  The structured set of annotations under USER SPECIFICS has been introduced in order to accommodate information about frequency and weighting. Contrary to the values for CENTRALITY, which express the importance of a phenomenon from a linguistic point of view, and the values for RELEVANCE, which express importance relative to application types, values for FREQUENCY and WEIGHTING will always be relative to some corpus or text type.

  Since the weighting of a phenomenon may differ across application types or across users testing systems belonging to the same type, we have introduced an intermediate attribute SYSTEM where either application type or specific system names can be indicated.

  As already stated in section 3.2.1 above, this set of annotations has been introduced partly for retrieval purposes, partly for assessment purposes. Especially for assessment purposes, values for weighting are frequently used when scoring and comparing systems for their performance.

- ORIGIN
  Describes where the input string (the TI-INPUT of the positive test item) comes from.

  Possible values are: *invented, corpus, derived from corpus.*

- REGISTER
  This attribute is needed at least for grammar checkers. Possible values could be: *formal, informal, scientific*, etc.

- TS-CATEGORY
  Usually, the test item is a complete sentence. Nevertheless, we want to allow NPs or other phrases and text fragments as admissible test items and to be able to find them easily. Thus, the possible values for this attribute will be the names for syntactic categories (*S, NP, PP*, etc.) and text fragment names such as *paragraph.*

- TS-FORMAT
  In case the test item requires formatting information, the attribute TS-FORMAT allows us to specify what it is.

  Other than *neutral*, which is the default value, possible values could be: *headers, tables, graphs, lists*, etc.

- TS-DIFFICULTY
  The purpose of this attribute is to indicate whether the test items in the test set are traps (both positive and negative), or whether they are straightforward (for a definition of traps we refer to [2]). It should be

noted that it depends on the *intention* of the test suite builder whether a test item is considered as a trap or not, not on the intrinsic difficulty of the test items.

Thus, the possible values are *'1'* (straightforward) and *'2'* (trap).

- TS-COMMENT
  This attribute allows the test writer to make comments in a more informal way.

4.4 Attributes at the TEST ITEM level (for both well-formed and ill-formed test items)

TEST ITEM is only a heading. It is not an attribute.

- TI-ID
  TI-ID is composed of the TS-ID (a number) and the "id of the test item" (a letter). For example, "1b" would be the second test item of test set number 1.
- TI-INPUT
  This is the actual input string. TI-INPUT can contain punctuation as well as lexical items.

  A TI-INPUT which is meant to be a complete sentence should end with a punctuation mark (period, question mark, etc.) and start with a capital letter.
- TI-WELLFORMEDNESS
  The possible values are: *'1'=correct, '0'=ill-formed, '2'=questionable.*
- TI-LENGTH
  The number of elements in the input string. Punctuation marks count as elements.

4.5 Attributes at the TEST ITEM level (for well-formed test items only)

- TI-ANALYSIS
  TI-ANALYSIS is an attribute which allows us to group together the five attributes it is composed of, namely *pos, cat, li, function and fd*.

  TI-ANALYSIS provides general (linguistic) information concerning the well-formed test item and mentions at least all the constituents in the input.

  This attribute is needed because it is necessary to have general information about the whole input item (note that TI-PARAMETER only contains (sub)phenomenon-specific information which often concerns only a part of the input string). This information may be seen as a “pattern” or template which will allow the creation of other test items through lexical replacement.

The amount of linguistic information given here is kept very small and only describes the most important aspect of the data (thus it is debatable whether the fourth attribute, FUNCTION, is really needed).

- POS

  The value of this attribute indicates the position of the constituent in the input string. It is expressed by means of 2 numbers separated by a colon. The value may be complex in case of discontinuous constituents:

  Example:

  ”Der Direktor wird um zwoelf Uhr essen.”

  NP 1:2

  VP 3:3,7:7

- CAT

  This attribute gives the syntactic category of the constituents. So far, its values are the following:

  SBAR, NP, N, PRON, VP, V, AdvP, Adv, AdjP, Adj, Conj, Neg, ...

  The level of analysis depends on the phenomenon which is treated in the test set. In general, we stay at the constituent level, i.e. if the phenomenon doesn’t concern elements inside a constituent, we do not describe these. However, if the phenomenon concerns elements inside a constituent, we must give the elements for this constituent.

  Examples:

  1. The phenomenon tests the valency of verbs with two arguments, so we have for example an arg2, which is an SBAR:

  ”Der Student beklagt, dass er die Pruefung nicht bestanden hat.”

  beklagen: subcat (arg1(nom), arg2(sbar(dass)))

  The phenomenon to be tested does not affect the elements inside the Sbar, thus we describe it as:

  ti-analysis

cat	pos	li	function	fd
NP	1:2	[”der Student”]	arg1	3:3
V	3:3	[”beklagt”]	functor	1:10
SBAR(dass)	4:10	[”dass er die Pruefung nicht bestanden hat”]	arg2	3:3

  2. If we now want to test the valency inside the subordinate clause, we need the information about the elements the SBAR is composed of. To show the dependency between the elements and the constituent which dominates them, we propose to indicate it explicitly by means of a colon. This notation is only meant to increase the user-friendliness, since this dependency is also

expressed implicitly by the value of the attributes POS and LI, and explicitly by the value of the attribute FD.

ti-analysis

cat	pos	li	function	fd
NP	1:2	["der Student"]	arg1	3:3
V	3:3	["beklagt"]	functor	1:10
SBAR(dass)	4:10	["dass er die Pruefung nicht bestanden hat"]	arg2	3:3
CONJ:SBAR	4:4	["dass"]	-	-
PRON:SBAR	5:5	["er"]	arg1	9:10
NP:SBAR	6:7	["die Pruefung"]	arg2	9:10
NEG:SBAR	8:8	["nicht"]	neg	-
VP:SBAR	9:10	["bestanden hat"]	functor	4:10

Indices indicating (co-)reference between constituents can be inserted in the CAT attribute, if such information is relevant for the test item.

Example:

ti-analysis

cat	pos	li	function	fd
$NP_{[i]}$	1:1	["Mary"]	arg1	2:2
V	2:2	["saw"]	functor	1:3
$NP_{[i]}$	3:3	["herself"]	arg2	2:2

– LI

lexical instantiation: the actual lexical items. In a hyper-text implementation, we foresee that it should be possible to click on this attribute and get the corresponding lexical entry, which would provide additional information not present in the annotations in the test item.

Lexical instantations are expressed by means of square brackets and quotation marks.

– FUNCTION

Possible values for this attribute are: *functor, arg1, arg2, arg3, adjunct (X), modifier*. (X can be AGENT, TIME , MANNER, etc.)

We use the standard linguistic definitions for *functor* and *argument*, i.e. the arguments are the phrases which are either lexically subcategorized (e.g. subject, object, etc. for verbs) or structurally analyzed as complements (e.g. complements of a noun or of an adjective: *complement of a noun*, *proud of his children*). Functors are thus the lexical items which lexically require or syntactically allow these arguments. We also follow standard definitions of *adjuncts* as phrases which are not required by the head of a constituent and whose addition does not change the category of that constituent.

The distinction between *adjunct* and *modifier* is more subtle and maybe is not necessary for most purposes: an *adjunct* is always a whole phrase (which in other contexts could have been an argument), while a modifier is not necessarily a whole phrase, it can be a lexical item such as an adjective (modifying a noun) or an adverb (modifying an adjective, a sentence or a prepositional phrase), and is not a phrase which could serve as an argument.

- FD (functional description)
  While the attribute FUNCTION gives the function that a particular element bears in a sentence or a phrase, the attribute FD links each constituent to either its functor if it is an argument, a modifier or an adjunct, or to its arguments if it is a functor.

  This attribute is needed when we look at elements which are embedded into another constituent as in the above example repeated here:

cat	pos	li	function	fd
NP	1:2	["der Student"]	arg1	3:3
V	3:3	["beklagt"]	functor	1:10
SBAR(dass)	4:10	["dass er die Pruefung nicht bestanden hat"]	arg2	3:3
CONJ:SBAR	4:4	["dass"]	-	-
PRON:SBAR	5:5	["er"]	arg1	9:10
NP:SBAR	6:7	["die Pruefung"]	arg2	9:10
NEG:SBAR	8:8	["nicht"]	neg	-
VP:SBAR	9:10	["bestanden hat"]	functor	4:10

  In this case, FD indicates the dependencies between the various arguments and functors (it becomes even more important when there is free word order). Similarly, if we want to allow argument-sharing between several functors, the crucial information for these dependencies is not provided by the FUNCTOR attribute alone, but FD allows us to state them correctly.

  The value of the FD attribute is a string position. For each element, it depends on the value of its FUNCTOR attribute: if the element is an argument, an adjunct or a modifier, we indicate the string position of its functor, while functors use this attribute to spell out their "domain". In both cases, the value may be complex, i.e. it may indicate discontinuous constituents or separate phrases.

- TI-TAG
  TI-TAG contains the information which a morpho-syntactic tagger might give. This attribute may be empty, for instance if no suitable tagger is available.

  The contents of TI-TAG always refers to a specific tagset. A tagger usually assigns a label to every single word (CRATER, XEROX, etc.), but some also do it at the constituent level (e.g. SUSANNE). Thus, the TI-TAG attribute is itself composed of two attributes: LI and tag(X), where the variable X is the name of the tagset which is used. LI can be either a single word or several words which form a constituent.

  The difference between TI-TAG and TI-ANALYSIS is that the latter must be present in all TSNLP test sets, while TI-TAG is optional. Moreover, the information given by TI-ANALYSIS is at the constituent level, even if it may go down to the lexical level when needed, while that of TI-TAG will most probably always remain at the lexical level. Nevertheless, these two attributes are complementary and they could be combined in the display when both are available.

- TI-PARAMETER
  TI-PARAMETER contains the attributes that specifically describe the phenomenon under consideration in the test set. In other words it contains all the "parameters" that characterize the phenomenon in question and that are relevant for the particular test set. Thus TI-parameter is related to the information given at the top level of the test suite module, namely the attributes PHENOMENON, TYPE and SUBTYPE. These attributes and TI-PARAMETER both describe the same thing, but the latter does it by means of a list of attributes and their corresponding values, whereas the former does it in a more general way.

  One function of the TI-PARAMETER attribute is to enable the test suite writer to stay in control of the coverage of the test suite and to build it systematically. Another function is to delimit the types of ill-formed test items that can meaningfully be said to violate the constitutive feature(s) of the phenomenon in question.

  The attributes in TI-PARAMETER are the features which distinguish this test set from other test sets in the test suite module. They are therefore the only features– or parameters– it makes sense to change when deriving ill-formed test items for the (sub)phenomenon. The TI-PARAMETER attribute in the positive test item is directly linked to the TI-ERROR attribute in the negative test items, since TI-ERROR in a negative test item should only contain attributes which appeared in the TI-PARAMETER attribute of the positive test item in that test set.

  TI-PARAMETER is composed of 4 attributes: *pos, attribute, value, LI*.

  - POS
    The value of this attribute indicates the position in the input string. It is again expressed by means of 2 numbers separated by a colon.
  - ATTRIBUTE
    The attributes of TI-PARAMETER are the features which are specific for every test set. ATTRIBUTE is actually the feature which describes the (sub)phenomenon to be tested and which lists any relevant additional constraints local to the positive test item.
  - VALUE
    This feature represents the value of the attribute. We have to be careful in distinguishing VALUE and LI of TI-PARAMETERS. Although VALUE and LI may look similar in some examples (e.g. the example given below), they have different functions with

respect to derivation. VALUE is not to be filled with a lexical instantiation, but with a linguistic descriptor for the value of ATTRIBUTE.

This will become clearer with the examples given below.

– LI

Again, lexical instantiations are expressed by means of square brackets and quotation marks. LI may stay empty (see first example below). Different attributes may have the same LI (see second example below).

The following example shows what TI-PARAMETER looks like:
ti-parameter

pos	attribute	value	LI
1:2	case:arg1	nom	["der Computer"]
4:5	adjunct(AGENT)	(prep + Parg)	
4:4	prep	von	["von"]

pos	attribute	value	LI
1:1	gender:Det	masc	["der"]
1:1	number:Det	sing	["der"]

The section on the attribute TI-ERROR for ill-formed test items gives further explanation concerning these features.

4.6 Attributes at the TEST ITEM level (for ill-formed test items only)

- TI-ERROR

Instead of the TI-ANALYSIS and TI-PARAMETER attributes, a negative test item only contains the attribute TI-ERROR, which describes where the error is in the input string.

All the other attributes given in the TI-ANALYSIS and TI-PARAMETER attributes of the positive test item are inherited by the negative test item: as far as the *attributes and values* of a test set are concerned, TI-ERROR subsumes TI-PARAMETER. Thus, when looking at an ill-formed test item, one must either have access to the TI-ANALYSIS and TI-PARAMETER of the well-formed test item in the same set, or these attributes must be explicitly present through an inheritance mechanism. The choice between these two options will be determined by database management considerations, and we will not say any more about it in this WorkPackage.

TI-ERROR contains the same attributes as TI-PARAMETER, namely POS, ATTRIBUTE, VALUE and LI (see above).

In our examples, the formal description of TI-ERROR is identical to the TI-PARAMETER, except that there is a * in front of each lexical instantiation violating an attribute or the expected value for an attribute (as usual, a more informal description of the errors can be provided in TI-comment).

TI-PARAMETER specifies not only the attribute(s) and value(s) ¿from which ill-formed input can be derived, but also the lexical instantiations and their position. This enables us to give a linguistically motivated but application independent formal description of where the error lies and of which constraint(s) it violates, by simply stating as the TI-ERROR that the lexical instantiation (whether mutilated, deleted or added) doesn't correspond to the expected attribute or value.

The relationship between TI-PARAMETER and TI-ERROR will be described in a more formal way. We propose at the same time a mechanism for deriving ill-formed test items. The first step in this derivation is the TI-PARAMETER attribute of the well-formed test item. Thus, while the TI-PARAMETER requires careful specification, the derivation itself is then relatively easy to carry out.

There are three possibilities to derive an ill-formed item: *modification*, *deletion* and *addition* of elements.

- if we modify an element:

ti-parameter

pos	attribute	value	LI
pos_a:pos_b	attribute	value	["X"]

ti-error

pos	attribute	value	LI
pos_a:pos_b	attribute	value	*["Y"]

"Y" replaces "X". This is possible only if "Y" is a lexical instantiation of the attribute in question. Thus, to create "Y", we must think of another possible value for the attribute and take its lexical instantiation. In our example, the possible values for "case" could be "nom", "dat" or "gen". To derive the ill-formed item, we take the lexical instantiation of one value, e.g. "dat", and have:

pos	attribute	value	LI
4:5	case:arg2	acc	*["dem Computer"]

One could argue that an even more precise description of the error would contain the "new" value (in our example "dat") with the * in front of it.

pos	attribute	value	LI
4:5	case:arg2	*dat	["dem Computer"]

However, a strong argument against this possibility is that the way to derive ill-formed input should consist in changing the LI without any attempt to indicate a new value for this wrong LI. Indeed, a value has only one lexical instantiation whereas a lexical instantiation might be ambiguous insofar as there can be several interpretations for it, thus several possible values.

For example, "Die Computer" can be "*masculine plural*" as well as "*accusative plural*". This is of course a general problem concerning errors.

The way to avoid this problem is simply to state that the lexical instantiation does not correspond to the expected attribute or value. Furthermore, this approach gives a systematic link between ill-formed and well-formed inputs: this ill-formed input then exists in the database only because it is linked to a well-formed input and only because the reason why it is considered ill-formed has been spelled out.

- if we delete an element:
  ti-parameter

pos	attribute	value	LI
pos_a:pos_b	attribute	value	["X"]

ti-error

pos	attribute	value	LI
pos_a:pos_b	attribute	value	*[]

"Y" is empty, i.e. we delete the element in question in the corresponding TI-INPUT. In this case POS will be empty as well.

pos	attribute	value	LI
4:5	case:arg2	acc	*[ ]

In an informal way, this means that the corresponding TI-INPUT is wrong since it should contain a 2nd argument in the accusative.

As was pointed out in the section on TI-PARAMETER, some attributes may have an empty LI. In the mechanism for deriving ill-formed test items, we propose that attributes with empty LI may not serve as the basis for the derivation of ill-formed test items.

- if we add an element:
  ti-parameter

pos	attribute	value	LI
pos_a:pos_b	attribute	value	["X"]

ti-error

pos	attribute	value	LI
pos_a:pos_b	attribute	value	["X"]
pos_c:pos_d	*attribute	value	["Y"]

As this description shows, we only change the value of LI when deriving an ill-formed test item in the first two cases (when adding an element, all the attributes POS, ATTRIBUTE, VALUE and LI have to be added).[8] The addition of an element takes thus a special position since it concerns all attributes. It is also harder to deal with this third kind of derivation. So far, we are not able to propose any restrictions on the type of addition. Thus, we have to be careful in adding elements especially with respect to the (sub)phenomenon. We have to make sure the added element tests the phenomenon we want to be tested.

Another difference concerns the position of the star. This time it will not be put in front of an LI, but in front of the attribute since we add a new attribute which is meant to be wrong. To make the following example clearer, we add the values of the respective TI-INPUT and the TI-WELLFORMEDNESS. The phenomenon to be tested is "valency (1 argument)".

ti-input *Der Mann kommt.*
ti-wellformedness *1*
ti-parameter

pos	attribute	value	LI
1:2	case:arg1	nom	["der Mann"]

ti-input *Der Mann kommt den Studenten.*
ti-wellformedness *0*
ti-error

pos	attribute	value	LI
1:2	case:arg1	nom	["der Mann"]
4:5	*case:arg2	acc	["den Studenten"]

Derivation is still an open issue and we have to practice in order to verify whether these more theoretical indications need a revision.

4.7 Additional attributes at the TEST ITEM level (for both well-formed and ill-formed test items)

- TI-OUTPUT
  The subdivision of output specifications is an attempt to accommodate a number of possible options:
  - The distinction between FORMAL and INFORMAL is introduced with the intention that FORMAL be reserved for output specifications that can be compared automatically with actual output from a system, whereas INFORMAL could be reserved

[8] It might be worth considering the possibility of using negative statements in the TI-PARAMETER of the well-formed test item when deriving ill-formed test items by the addition of forbidden elements.

for cases where only general indications can be given. Both parts may of course be empty.

- Different application types are likely to require different FORMAL and INFORMAL output specifications.
- Different systems are likely to require different FORMAL output specifications (whereas the INFORMAL part of the OUTPUT specifications can probably be the same for all systems of the same application type
- For the sake of illustration, MT has been included as an application type since it requires different output specifications for each different target language; moreover, we can imagine that at least the FORMAL output specifications might need to provide e.g. ranked lists of possible translations. Whether the INFORMAL part should also foresee this possibility is an open question.

At the moment, we don't know in how much detail it is possible and desirable to specify expected output for ill-formed test-items, but we need this option at least for grammar checkers and controlled language checkers.

- TI-COMMENT

5 Prototype Data

5.1 German Prototype Data

5.1.1 Passive

TEST SUITE
language *de*

TEST SUITE MODULE
tsm-id *1*
phenomenon *passiv*
type *vorgangspassiv*
subtype *A*
phenomenon-code *A:VOR:PAS*
presupposition *Subj/V-Agreement; Valency in active sentences;*
interaction *none*
tsm-comment -

TEST SET
ts-id *1*
ts-info
  author *ISSCO*
  date *September 1994*
  purpose *example*
  restrictions *neutral*
  centrality *core*
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type
      frequency
      weighting
  origin *invented*
  register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *Der Computer wird von ihm gekauft.*
ti-wellformedness *1*

ti-length *7*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["der Computer"]	arg1	3:3,6:6
3:3,6:6	V	["wird gekauft"]	functor	1:6
4:5	PP	["von ihm"]	adjunct(AGENT)	3:3,6:6
7:7	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
6:6	surface-subcat	subcat(arg1)	
1:2	case:arg1	nom	["der Computer"]
4:5	adjunct(AGENT)	(prep + Parg)	["von ihm"]
4:4	prep	von	["von"]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Den Computer wird von ihm gekauft.*
ti-wellformedness *0*
ti-length *7*
ti-error

pos	attribute	value	LI
1:2	case:arg1	nom	*["den Computer"]

ti-output
ti-comment *The subject (arg1) should be nominative.*

TEST ITEM
ti-id *1c*
ti-input *Der Computer wird von gekauft.*
ti-wellformedness *0*
ti-length *6*
ti-error

pos	attribute	value	LI
4:4	adjunct(AGENT)	(prep + Parg)	*["von"]

ti-output
ti-comment *The preposition "von" needs an argument.*

TEST SUITE
language *de*

TEST SUITE MODULE

tsm-id *2*
phenomenon *passiv*
type *vorgangspassiv*
subtype *B*
phenomenon-code *B:VOR:PAS*
presupposition *Subj/V-Agreement; NP-Agreement; Word-order;Verbal Morphology*
interaction *none*
tsm-comment -

TEST SET
ts-id *1*
ts-info
    author *ISSCO*
    date *September 1994*
    purpose *example*
    restrictions *neutral*
    centrality *core*
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type
            frequency
            weighting
    origin *invented*
    register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *Dem Studenten wird geholfen.*
ti-wellformedness *1*
ti-length *5*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["dem Studenten"]	arg1	3:4
3:4	V	["wird geholfen"]	functor	1:4
5:5	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
4:4	surface-subcat	subcat(arg1)	
1:2	case:arg1	dat	["dem Studenten"]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Der Student wird geholfen.*
ti-wellformedness *0*
ti-length *5*
ti-error

pos	attribute	value	LI
1:2	case:arg1	dat	*["der Student"]

ti-output
ti-comment *The subject (arg1) should be dative.*

TEST ITEM
ti-id *1c*
ti-input *Des Studenten wird geholfen.*
ti-wellformedness *0*
ti-length *5*
ti-error

pos	attribute	value	LI
1:2	case:arg1	dat	*["des Studenten"]

ti-output
ti-comment *The subject (arg1) should be dative.*

TEST SUITE
language *de*

TEST SUITE MODULE
tsm-id *3*
phenomenon *passiv*
type *vorgangspassiv*
subtype *C*
phenomenon-code *C:VOR:PAS*
presupposition *Subj/V-Agreement; NP-Agreement; Word-order;Verbal Morphology*
interaction *none*
tsm-comment -

TEST SET
ts-id *1*
ts-info
author *ISSCO*

date *September 1994*
purpose *example*
restrictions *neutral*
centrality *core*
relevance
parser *central*
grammar checker *central*
controlled language checker *central*
user specifics
system 1
text type
frequency
weighting
origin *invented*
register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *Es wird gearbeitet.*
ti-wellformedness *1*
ti-length *4*
ti-analysis

pos	cat	li	function	fd
1:1	Pron	["es"]	arg1	2:3
2:3	V	["wird gearbeitet"]	functor	1:3
4:4	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
3:3	surface-subcat	subcat(arg1)	
1:1	case:arg1	dummy	["es"]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Er wird gearbeitet.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	case:arg1	dummy	*["er"]

ti-output
ti-comment *"Arbeiten" requires a dummy subject in passive.*

TEST ITEM
ti-id *1c*
ti-input *Sie wird gearbeitet.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	case:arg1	dummy	*["sie"]

ti-output
ti-comment *"Arbeiten" requires a dummy subject in passive.*

5.1.2 Verb Valency

TEST SUITE
language *de*

TEST SUITE MODULE
tsm-id *4*
phenomenon *Valency*
type *1 argument*
phenomenon-code *1ARG:VAL*
presupposition *Subj/V-Agreement; NP-Agreement; Word-order*
interaction *none*
tsm-comment -

TEST SET
ts-id *1*
ts-info
  author *ISSCO*
  date *September 1994*
  purpose *example*
  restrictions *neutral*
  centrality *core*
  relevance
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type
      frequency
      weighting
  origin *invented*
  register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *Der Student kommt.*
ti-wellformedness *1*
ti-length *4*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["der Student"]	arg1	3:3
3:3	V	["kommt"]	functor	1:3
4:4	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
3:3	subcat	subcat(arg1)	
1:2	case:arg1	nom	["der Student"]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Den Studenten kommt.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:2	case:arg1	nom	*["den Studenten"]

ti-output
ti-comment *The subject (arg1) should be nominative.*

TEST ITEM
ti-id *1c*
ti-input *Der Student kommt die Pruefung.*
ti-wellformedness *0*
ti-length *6*
ti-error

pos	attribute	value	LI
4:5	*arg2	acc	["die Pruefung"]

ti-output
ti-comment *"Kommen" is an intransitive verb.*

TEST SUITE
language *de*

TEST SUITE MODULE
tsm-id *5*
phenomenon *Valency*
type *2 arguments*
phenomenon-code *2ARG:VAL*
presupposition *Subj/V-Agreement; NP-Agreement; Word-order;*
interaction *none*

tsm-comment -

TEST SET
ts-id *1*
ts-info
    author *ISSCO*
    date *September 1994*
    purpose *example*
    restrictions *neutral*
    centrality *core*
    relevance
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type
            frequency
            weighting
    origin *invented*
    register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1*
ti-input *Der Student antwortet, dass er die Pruefung bestanden hat.*
ti-wellformedness *1*
ti-length *11*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["der Student"]	arg1	3:3
3:3	V	["antwortet"]	functor	1:11
4:14	PUNCT	[","]		
5:10	SBAR(dass)	["dass er die Pruefung bestanden hat"]	arg2	3:3
11:11	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
3:3	subcat	subcat(arg1,arg2)	
1:2	case:arg1	nom	["der Student"]
5:10	arg2	sbar(conj + S)	["dass er die Pruefung bestanden hat"]
5:5	conj	dass	["dass"]

ti-output

ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Der Student antwortet, er die Pruefung bestanden hat.*
ti-wellformedness *0*
ti-length *10*
ti-error

pos	attribute	value	LI
5:9	arg2	sbar(conj + S)	*["er die Pruefung bestanden hat"]
	conj	dass	*[ ]

ti-output
ti-comment *The subordinating conjunction is missing.*

TEST ITEM
ti-id *1c*
ti-input *Der Student antwortet, wenn er die Pruefung bestanden hat.*
ti-wellformedness *0*
ti-length *11*
ti-error

pos	attribute	value	LI
5:9	arg2	sbar(conj + S)	*["wenn er die Pruefung bestanden hat"]
5:5	conj	dass	* ["wenn"]

ti-output
ti-comment *"Wenn" is the wrong subordinating conjunction.*

TEST SET
ts-id *2*
ts-info
  author *ISSCO*
  date *September 1994*
  purpose *example*
  restrictions *neutral*
  centrality *core*
  relevance
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type
      frequency
      weighting
  origin *invented*
  register *formal*

ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *2a*
ti-input *Der Student braucht den Computer.*
ti-wellformedness *1*
ti-length *6*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["der Student"]	arg1	3:3
3:3	V	["braucht"]	functor	1:5
4:5	NP	["den Computer"]	arg2	3:3
6:6	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
3:3	subcat	subcat(arg1,arg2)	
1:2	case:arg1	nom	["der Student"]
4:5	case:arg2	acc	["den Computer"]

ti-output
ti-comment-

TEST ITEM
ti-id *2b*
ti-input *Der Student braucht.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
	case:arg2	acc	*[ ]

ti-output
ti-comment *"Brauchen" is transitive; it needs a direct object.*

TEST ITEM
ti-id *2c*
ti-input *Den Studenten braucht den Computer.*
ti-wellformedness *0*
ti-length *6*
ti-error

pos	attribute	value	LI
3:3	subcat	subcat(arg1,arg2)	
1:2	case:arg1	nom	*["den Studenten"]
4:5	case:arg1	nom	*["den Computer"]

ti-output
ti-comment *The subject (arg1) should be nominative.*

5.1.3 NP Agreement

TEST SUITE
language *de*

TEST SUITE MODULE
tsm-id *6*
phenomenon *Agreement*
type *NP*
subtype *Det-N*
phenomenon-code *DET-N:NP:AGR*
presupposition *Subj/V-Agreement*
interaction *none*
tsm-comment -

TEST SET
ts-id *1*
ts-info
  author *ISSCO*
  date *September 1994*
  purpose *example*
  restrictions *neutral*
  centrality *core*
  relevance
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type
      frequency
      weighting
  origin *invented*
  register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*

ti-input *Das Flugzeug landet.*
ti-wellformedness *1*
ti-length *4*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["das Flugzeug"]	arg1	3:3
1:1	Det:NP	["das"]		
2:2	N:NP	["Flugzeug"]		
3:3	V	["landet"]	functor	1:3
4:4	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
1:1	gender:Det	neut	["das"]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *Der Flugzeug landet.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	gender:Det	neut	*["der"]

ti-output
ti-comment *Gender is wrong. "Flugzeug" is neuter.*

TEST ITEM
ti-id *1c*
ti-input *Die Flugzeug landet.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	gender:Det	neut	*["die"]

ti-output
ti-comment *Gender is wrong. "Flugzeug" is neuter.*

TEST SET
ts-id *2*
ts-info
author *ISSCO*
date *September 1994*
purpose *example*

restrictions *neutral*
centrality *core*
relevance
parser *central*
grammar checker *central*
controlled language checker *central*
user specifics
system 1
text type
frequency
weighting
origin *invented*
register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *2a*
ti-input *Der Student kommt.*
ti-wellformedness *1*
ti-length *4*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["der Student"]	arg1	3:3
1:1	Det:NP	["der"]		
2:2	N:NP	["Student"]		
3:3	V	["kommt"]	functor	1:3
4:4	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
1:1	gender:Det	masc	["der"]

ti-output
ti-comment

TEST ITEM
ti-id *2b*
ti-input *Die Student kommt.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	gender:Det	masc	*["die"]

ti-output

ti-comment *Gender is wrong. "Student" is masculine.*

TEST ITEM
ti-id *2c*
ti-input *Das Student kommt.*
ti-wellformedness *0*
ti-length *4*
ti-error

pos	attribute	value	LI
1:1	gender:Det	masc	*["das"]

ti-output
ti-comment *Gender is wrong. "Student" is masculine.*

TEST SET
ts-id *3*
ts-info
    author *ISSCO*
    date *September 1994*
    purpose *example*
    restrictions *neutral*
    centrality *core*
    relevance
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type
            frequency
            weighting
    origin *invented*
    register *formal*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *3a*
ti-input *Die Studenten kommen.*
ti-wellformedness *1*
ti-length *4*
ti-analysis

pos	cat	li	function	fd
1:2	NP	["die Studenten"]	arg1	3:3
1:1	Det:NP	["die"]		
2:2	N:NP	["Studenten"]		
2:2	V	["kommen"]	functor	1:3
4:4	PUNCT	["."]		

ti-tag
ti-parameter

pos	attribute	value	LI
1:1	gender:Det	masc	["die"]

ti-output
ti-comment-

TEST ITEM
ti-id *3b*
ti-input *This is a case where it is not obvious to derive an ill-formed test item which would test only Det - N agreement. Further test items for determiner + Student are of course possible but not inside this specific test set which requires a plural NP: any combination of determiner + Studenten would yield a possible NP in some case in either singular or plural and thus turn the test item into a test for subject/verb agreement.*

5.2 English Prototype Data

5.2.1 NP Agreement

TEST SUITE
language *en*

TEST SUITE MODULE
tsm-id *1*
phenomenon *NP agreement*
type *indefinite + noun*
subtype
phenomenon-code
presupposition
interaction *none*
tsm-comment *the indefinite can only cooccur with a singular count noun*

TEST SET
ts-id 1
ts-info
    author *SX*
    date *Sept 94*
    purpose *test/evaluation/ ...*
    restrictions *neutral*
    centrality *core*
    relevance
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type *spec of text type*
            frequency *scale*
            weighting *scale*
        system 2
        etc..
    origin *invented*
    register *neutral*
ts-category *NP*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *a computer*

ti-wellformedness *1*
ti-length *2*
ti-analysis

pos	cat	li	function	fd
1:1	Det	[“a”]	functor	2:2
2:2	N	[“a computer”]	arg1	1:1

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
1:2	number:NP	sing	[“a computer”]
1:1	number:functor	sing	[“a”]
2:2	number:arg1	sing	[”computer”]
2:2	sem_feature:	count	[“computer”]

ti-output
ti-comment-

TEST ITEM
ti-id *1b*
ti-input *a computers*
ti-wellformedness *0*
ti-length *2*
ti-error

pos	attribute	value	LI
2:2	number:arg1	sing	*[”computers”]
1:2	number:NP	sing	*[”a computers”]

ti-output
ti-comment-

TEST ITEM
ti-id *1c*
ti-input *a money*
ti-wellformedness *0*
ti-length *2*
ti-error

pos	attribute	value	LI
1:2	number:NP	sing	*[“a money”]
2:2	sem_feature:	count	*[“money”]

ti-output
ti-comment-

TEST SUITE MODULE
tsm-id *1*
phenomenon *NP agreement*
type *definite + noun*

subtype
phenomenon-code
presupposition
interaction *none*
tsm-comment

TEST SET
ts-id 2
ts-info
  author *SX*
  date *Sept 94*
  purpose *test/evaluation/ ...*
  restrictions *neutral*
  centrality *core*
  relevance
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type *spec of text type*
      frequency *scale*
      weighting *scale*
    system 2
    etc..
  origin *invented*
  register *neutral* ts-category *NP*
ts-format *none*
ts-centrality *core*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *the computer*
ti-wellformedness *1*
ti-length *2*
ti-analysis

pos	cat	li	function	fd
1:1	Det	[“the”]	functor	2:2
2:2	N	[“computer”]	arg1	1:1

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
1:2	number:NP	sing	[“the computer”]
1:1	number:functor	sing	[“the”]
2:2	number:arg1	sing	[”computer”]

ti-output
ti-comment*there is no obvious ill-formed output here*

TEST ITEM
ti-id *1b*
ti-input *the computer*
ti-wellformedness *1*
ti-length *2*
ti-analysis

pos	cat	li	function	fd
1:1	Det	[“the”]	functor	2:2
2:2	N	[“computers”]	arg	1:1

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
1:2	number:NP	plural	[“the computer”]
1:1	number:functor	plural	[“the”]
2:2	number:arg1	plural	[”computers”]

ti-output
ti-comment*there is no obvious ill-formed output here*

TEST SUITE MODULE
tsm-id *3*
phenomenon *NP agreement*
type *possessive pron + noun*
subtype
phenomenon-code
presupposition
interaction *none*
tsm-comment

TEST SET
ts-id 1
ts-info
author *SX*
date *Sept 94*
purpose *test/evaluation/ ...*
restrictions *neutral*
centrality *core*
relevance

parser *central*
grammar checker *central*
controlled language checker *central*
user specifics
system 1
text type *spec of text type*
frequency *scale*
weighting *scale*
system 2
etc..
origin *invented*
register *neutral* ts-category *NP*
ts-format *none*
ts-centrality *core*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *her computer*
ti-wellformedness *1*
ti-length *2*
ti-analysis

pos	cat	li	function	fd
1:1	Det	[“her”]	functor	2:2
2:2	N	[“computer”]	arg	1:1

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
1:2	number:NP	sing	[“her computer”]
1:1	number:functor	sing	[“her”]
2:2	number:arg1	sing	[”computer”]

ti-output
ti-comment

TEST ITEM
ti-id *1b*
ti-input *hers computers*
ti-wellformedness *0*
ti-length *2*
ti-error

pos	attribute	value	LI
1:2	number:NP	sing	*[“hers computer”]
1:1	number:functor	sing	*[“hers”]

ti-output

ti-comment -

TEST SET
ts-id 2
ts-info
    author *SX*
    date *Sept 94*
    purpose *test/evaluation/ ...*
    restrictions *neutral*
    centrality *core*
    relevance
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type *spec of text type*
            frequency *scale*
            weighting *scale*
        system 2
        etc..
    origin *invented*
    register *neutral* ts-category *NP*
ts-format *none*
ts-centrality *core*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1a*
ti-input *her computers*
ti-wellformedness *1*
ti-length *2*
ti-analysis

pos	cat	li	function	fd
1:1	Det	[“her”]	functor	2:2
2:2	N	[“computers”]	arg	1:1

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
1:2	number:NP	plural	[“her computers”]
1:1	number:functor	plural	[“her”]
2:2	number:arg1	plural	[”computers”]

ti-output
ti-comment

TEST ITEM
ti-id *1b*
ti-input *hers computers*
ti-wellformedness *0*
ti-length *2*
ti-error

pos	attribute	value	LI
1:2	number:NP	plural	*[“hers computers”]
1:1	number:functor	plural	*[“hers”]

ti-output
ti-comment *Possessive pronouns do not inflect for number*

5.2.2 Verb Valency

TEST SUITE
language *en*

TEST SUITE MODULE
tsm-id *1*
phenomenon *valency*
type *divalent*
subtype *sentential arguments*
phenomenon-code S:DI:VAL
presupposition *NPs, non-sentential verb arguments, subject attributes*
interaction *none*
tsm-comment

TEST SET
ts-id *1* ts-info
  author *CT*
  date *070994*
  purpose
  restrictions *neutral*
  centrality *core*
  relevance
    parser *central*
    grammar checker *central*
    controlled language checker *central*
  user specifics
    system 1
      text type
      frequency
      weighting
  origin *invented*
  register *neutral*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1* ti-input *The lawyer agrees that the company is large.*
ti-wellformedness *1*
ti-length *9*
ti-analysis

pos	cat	li	function	fd
1:2	NP	[‘the lawyer’]	arg1	3:3
3:3	V	[‘agrees’]	functor	1:8
4:8	SBAR	[‘that the company is large’]	arg2	3:3
9:9	PUNC	[‘.’]		

ti-tag

LI tag (X)

ti-parameter

pos	attribute	value	LI
2:2	arg2:S	thatS(finite)	[‘agree’]
4:8	thatS	finite	
7:8	tense	finite	[‘is’]

ti-output

ti-comment

TEST ITEM

ti-id *2*

ti-input *The lawyer agrees that the company to be large.*

ti-wellformedness *0*

ti-length *10*

ti-error

pos	attribute	value	LI
2:2	arg2:S	thatS(*finite)	[‘agree’]
4:8	thatS	*finite	
7:8	tense	*finite	[‘to be’]

ti-output

ti-comment

5.2.3 Passive

TEST SUITE
language *en*

TEST SUITE MODULE
tsm-id *1*
phenomenon *passive*
type *monotransitive*
subtype *prepositional verbs*
phenomenon-code *PVERB:MONO:PAS*
presupposition *NPs, Subj/V-Agreement, Valency in active sentences*
interaction *none*
tsm-comment –

TEST SET
ts-id *1*
ts-info
    author *SX*
    date *120994*
    purpose
    restrictions *neutral*
    centrality *core*
    relevance
        parser *central*
        grammar checker *central*
        controlled language checker *central*
    user specifics
        system 1
            text type
            frequency
            weighting
        system 2
    origin *invented*
    register *neutral*
ts-category *S*
ts-format *none*
ts-difficulty *1*
ts-comment -

TEST ITEM
ti-id *1*
ti-input *The company was invested in.*
ti-wellformedness *1*
ti-length *6*
ti-analysis

pos	cat	li	function	fd
NP	1:2	[‘the company’]	arg1	3:5
Aux	3:5	[‘was invested in’]	functor	1:5
Punc	6:6	[‘.’]		

ti-tag

LI	tag (X)

ti-parameter

pos	attribute	value	LI
3:5	surface subcat	subcat(arg1)	
1:2	case:arg1	nom	[‘the company’]
4:5	verbtype	prepverb	
5:5	prepverb:prep	in	[‘in’]

ti-output
ti-comment*Preposition stays with the verb*

TEST ITEM
ti-id *2*
ti-input *In the company was invested.* ti-wellformedness *2*
ti-length *6*
ti-error

pos	attribute	value	LI
1:1	preptype	particle	in
4:5	verbtype	prepverb	[‘invested $*$–’]

ti-output
ti-comment*Passive inversion does not affect the position of prepositions in prepositional verbs*

5.3 French Prototype Data

5.3.1 NP Agreement

This prototype data has been written according to an earlier version of the annotation scheme. As no soft copy is available we include the original here.

6 Conclusion

The annotation scheme we present in this report is the result of Task 2.2, but the work itself is meant to be continued in later work packages of the project. Thus, it should not be considered as a fossilized set of immutable attributes. However, assuming that TSNLP must provide a structured and systematic way to construct test suites for various purposes, the scheme in its present form is taking into account all the major issues that had been foreseen in the Technical Annex and during the course of WP1 and WP2.1, and also a number of other issues which have arisen during the course of this task. While some of them have been taken in detail in various sections of this report, others have not been adequately addressed yet.

Issues that have been the main focus of attention

- multilinguality:

  The problem is how to relate test suites written for different languages. TSNLP users might want to see how the same phenomenon is treated by a system in the different languages, and they may also want to compare the coverage of an application for the different languages.

  This issue arises both at the phenomenon level and at the lexical level. We want to build parallel lists of phenomena and draw up a list of equivalences between phenomena in the different languages. On the lexical level, we want to build sets of test items, which are not necessarily translations of each others, but which are in some sense comparable (e.g. using vocabulary from the same domain and lexical items with similar syntactic requirements).

  A related question is whether it is appropriate to provide translations for the test items. We have for now decided that it was unnecessary, but the TSNLP database might include one or several translations if the need arises.[9]

- list of phenomena:

  This question has barely been tackled in the writing of the prototype data, but has been a constant source of concern during the task of designing the annotation scheme, since it comes up in several contexts. Indeed, an agreed upon multilingual list of phenomena is necessary not only for the definition of the test items themselves, but also in the specification of the phenomena which are taken to have been already tested in the progressive building of the test suite, and in the specification of interactions that are included on purpose in particular test items.

[9] If MT is one of the applications, translations will of course have to be provided in the specification of the output.

- sentence patterns:

  The idea of providing a description of the sentence patterns which are instantiated by particular test items was already underlying the discussion in WP2.1, although it was not explicitly formulated in [2]. It has been discussed during the course of this task and is addressed by the attributes TI-ANALYSIS and TI-TAG. However, as was mentioned in section 3.2, we do not rule out the possibility that an additional attribute to express sentence patterns in a more abstract way might be added.

Outstanding issues

- frequency, weighting:

  The question is how to integrate in the annotations given with a test item information about the relevance and the importance of the phenomenon under consideration in this item. This information may concern its relevance for the type of application under testing or for the type of texts to be treated.

  Discussions of this issue has built on the work done on corpora studies (see the annex to D-WP2.1 [2]), but were of necessity very limited in depth. Indeed, while it was relatively obvious how to devise annotations addressing this question, it is clear that any actual use of these annotations assumes that a number of other issues have been resolved. In particular, it presupposes that the determination of frequency and of weights in actual texts is available and that it has been possible to specificy how to use this frequency and weights when using test suites.

- description of expected output:

  While we have provided in the annotation scheme a number of predefined possibilities for expressing what the expected output of the system under consideration should be, it is up to the test suite users to specify those elements which must be part of their evaluation. Thus, though we already know what kind of information would be necessary for each *type* of application, we have not attempted in our prototype data to fill in that section of the annotation scheme in any detail. This exercise would have been pointless in the abstract, and will be performed during WP3, when we build real test data with specific applications in mind, and WP4 when we test our test suites on real systems.

- semantics:

  The annotation scheme allows semantic annotations (e.g. the *mass/count* distinction necessary for NP agreement), and the annotations provided for a given input may reflect semantic facts, but this aspect of the test data has not been studied in great detail.

In addition to the above, one of the major concerns in this task has been to construct the annotation scheme in such a way that it allows the test suite writer to control the derivation of ill-formed test input. The issue has been addressed from different angles in several places in this report and will certainly continue to be a subject for discussion and future work. The solution we have proposed here, while helping greatly in solving some of the problems, still requires a lot of thought on the part of the test suite writer and cannot be automatically implemented.

References

[1] Arrarte G., T. Redondo, M. Sobejano & I. Zapata (1991). "A Management Tool for Test Corpora". IBM Madrid Scientific Center, Spain. also in *Proceedings of the Evaluators' Forum.* Les Rasses.

[2] Balkan, L., S. Meijer et al. (1994), *Test Suite Design — Guidelines and Methodology*, Report to LRE 62-089 (D-WP2.1), University of Essex.

[3] Estival, D., K. Falkedal et al. (1994), *Analysis of Existing Test Suites*, Report to LRE 62-089 (D-WP1), University of Essex.

[4] Flickinger, D., J. Nerbonne, I.A. Sag & T. Wasow (1987). "Toward Evaluation of NLP Systems". Hewlett-Packard Laboratories, distr. at the 24th Annual Meeting of the Association for Computational Linguistics (ACL), Stanford.

[5] Garside Roger, Geoffrey Leech & Geoffrey Sampson (1987). *The computational analysis of English. A corpus-based approach.* London: Longman.

[6] Grammatik (1991). *GRAMMATIK Windows, Version frangaise 1.0. Guide d'utilisation* Reference Software International. San Francisco.

[7] Marcus, Mitchell, Beatrice Santorini & Mary Ann Marcinkiewicz (1993). *Building a large annotated corpus of English: the Penn Treebank Project. Computational Linguistics,* 19.3, pp.313–30.

[8] Neal, Jeannette G., Elissa L. Feit, Douglas J. Funke & Christine A. Montgomery (1992). *An Evaluation Methodology For Natural Language Processing Systems* RL-TR–92-308. Rome Laboratory.

[9] Nerbonne, J., D. Flickinger, T. Wasow. (1988). "The HP Labs Natural Language Evaluation Tool". in *Proceedings of the workshop on Evaluation of Natural Language Processing Systems*, Wayne, Philadelphia December 8-9.

[10] Nerbonne, J. K. Netter, A. Kader Diagne, J. Klein, L. Dickman (1991). *A Diagnostic Tool for German Syntax.* DFKI D-92-03, Saarbrücken. Also in: J. Neal / S. Walter. eds. (1991): *Natural Language Processing Systems Evaluation Workshop, Berkeley.* Rome Laboratory, RL-TR-91-362, New York.

[11] Sampson, G.R. (1993). "The need for grammatical stocktaking". *Literary and Linguistic Computing,* 8.267-73.

[12] Sampson, G.R. (forthcoming). "SUSANNE: A Domesday Book of English Grammar", in Oostdijk, N. & de Haan, P.: *Corpus-based research into language.*

[13] Sánchez Léon, Fernando (1994). "Spanish tagset for the CRATER project." Laboratorio de Linguïstica Informática. Universidad Autónoma de Madrid, March 1994.