Computers and the Humanities **29**: 191–209, 1995.

A Rationale for the TEI Recommendations for Feature-Structure Markup[1]

D. Terence Langendoen *
Department of Linguistics, University of Arizona, Tucson, AZ 85721, USA
e-mail: langendoen@linguistics.arizona.edu

and

Gary F. Simons **
Summer Institute of Linguistics, 7500 West Camp Wisdom Road, Dallas 75236, TX, USA
e-mail: gary@sil.org

Key words: TEI, SGML, linguistic tagging, feature structure, features, lexical tagging, interlinear text analysis

Abstract

In this paper, we concentrate on justifying the decisions we made in developing the TEI recommendations for feature structure markup. The first four sections of this paper present the justification for the recommended treatment of feature structures, of features and their values, and of combinations of features or values and of alternations and negations of features and their values. Section 5 departs briefly from the linguistic focus to argue that the markup scheme developed for feature structures is in fact a general-purpose mechanism that can be used for a wide range of applications. Section 6 describes an auxiliary document called a "feature system declaration" that is used to document and validate a system of feature-structure markup. The seventh and final section illustrates the use of the recommended markup scheme with two examples, lexical tagging and interlinear text analysis.

One of the goals of the Text Encoding Initiative is to provide a markup scheme that will permit scholars to encode linguistic analyses of any text in any language to any desired degree of detail. We envision all sorts of ways of using the TEI recommendations for linguistic markup, ranging from making occasional notations about the grammatical analysis of selected words in a document to preparing a full-blown phonological, morphological, syntactic, and semantic analysis of an entire document. The mechanisms that we provide are designed to express the content of commonly used representations of linguistic analysis, including tree diagrams, feature-structure diagrams, and interlinear glossing. Thus the markup scheme can be used to represent explicitly the implicit linguistic structure of a text (with respect to a particular analysis). The result is a machine-readable and machine-interpretable version of an explicit analysis presented according to familiar human-readable and human-interpretable graphical and typographical conventions.

* Terry Langendoen is Professor and Head of the Department of Linguistics at The University of Arizona. He was Chair of the TEI Committee on Analysis and Interpretation. He received his PhD in Linguistics from the Massachusetts Institute of Technology in 1964, and held teaching positions at The Ohio State University and the City University of New York (Brooklyn College and the Graduate Center) before moving to Arizona in 1988. He is author, co-author, or co-editor of six books in linguistics, and of numerous articles.

** Gary Simons is Director of the Academic Computing Department of the Summer Institute of Linguistics, Dallas, TX. He served on the TEI Committee on Analysis and Interpretation. He received his PhD in Linguistics (with minor emphasis in Computer Science) from Cornell University in 1979. Before taking up his current position in 1984, he spent five years in the Solomon Islands doing field work with SIL. He is author, co-author, or co-editor of eight books in the fields of linguistics and linguistic computing.

In developing the recommendations for the markup of linguistic analysis and interpretation within the Text Encoding Initiative, we have chosen to concentrate on providing markup for the data structures that linguists typically use to represent the results of their analysis and interpretation, rather than providing markup for particular analyses and interpretations. In this way, we enable encoders to represent any analysis they wish, and do not force them to use the constructs of predefined analyses which may not be suitable for the material being encoded and which may not be easy to alter or extend. Markup is provided for two data structures in particular: feature structures and tree structures. In fact, the two data structures can be combined; any node in a tree can itself be a feature structure.

In this paper, we concentrate on justifying the decisions we made in developing the TEI recommendations for feature structure markup. It is not within the scope of this paper to give a formal definition of the markup scheme. (For that and for other details concerning feature-structure, tree, and other markup being recommended for linguistic analysis and interpretation, see TEI P3. [Sperberg-McQueen and Burnard, 1994]) The first four sections of this paper present the justification for the recommended treatment of feature structures, of features and their values, and of combinations of features or values and of alternations and negations of features and their values. Section 5 departs briefly from the linguistic focus to argue that the markup scheme developed for feature structures is in fact a general-purpose mechanism that can be used for a wide range of applications. Section 6 describes an auxiliary document called a "feature system declaration" that is used to document and validate a system of feature-structure markup. The seventh and final section illustrates the use of the recommended markup scheme with two examples, lexical tagging and interlinear text analysis.

1. Justification for Treatment of Feature Structures

Within linguistics, feature structures were first used by Roman Jakobson (1949) for the representation of phonemes (the distinctive or contrastive sounds in a natural language) as bundles of what he called distinctive features. Each feature represented an essential articulatory and acoustic property of a phoneme. So, for example, Jakobson considered the phoneme s/in English to be a bundle of distinctive features, such as [-grave], [+anterior], [voice], [+strident], etc. (In Jakobson's theory, the values are restricted to the binary values "+" and "−"; see section 2 for a discussion of the markup of these and other possible feature values.)

The use of feature structures quickly spread to other domains within linguistics. Chomsky (1965) extended Jakobson's distinctive feature theory to morphology and syntax, and developed the idea that a feature value within a particular feature structure could make reference to another feature structure. Subsequent developments in the theory of feature structures (within both linguistics and computer science) have led to the current situation in which feature structures, either by themselves or together with directed graphs (such as phrase-structure trees) in which feature structures are associated with the nodes in the graphs, can be used to express any linguistic analysis whatever. The ability of feature structures to serve as a general purpose linguistic metalanguage is what led us to use them as the basis of linguistic encoding.

To represent the structure that groups together a bundle of features and their values, we need a markup element that groups together (or, in SGML parlance, whose content model is a group of) features. We call that element **<fs>** (for "feature structure"), and the element that it groups together **<f>** (for "feature"). Leaving aside details that need not concern us here, the content of an **<fs>** is zero or more **<f>**s.

A linguistic analysis may consist of many different types of **<fs>** elements, each of which may group together different types of **<f>** elements. To distinguish among the different types of **<fs>** elements, we provide a *type* attribute, whose value is a string (such as "phonological structure" or "phrase structure") that specifies the feature-structure type. So, for instance, the markup of the distinctive feature analysis of a phoneme might look as follows:

```
<fs type="phonological structure">
   <f ... > ... </f>
      <!- -Representation for one distinctive feature- ->
   ...
   <f ... > ... </f>
```

```
        <!- -Representations for other distinctive features- ->
    </fs>
```

One important reason to distinguish among feature structure types is that different types of feature structures will contain different types of features. However, given that we are proposing to use SGML attributes to distinguish among feature-structure types (and also to a considerable extent among feature types), it will not be possible to use an SGML validator to determine whether the appropriate types of features are being used as the content of a feature structure of a particular type. To enable an encoder to specify explicitly what feature types go with what feature-structure type, we provide recommendations for the creation of a special file called a Feature System Declaration, that specifies, among other things, the correlations between feature-structure and feature types (see section 6).

2. Justification for Treatment of Features

In this section we justify the approach taken to marking up features, that is, we explain why the <f> tag is defined as it is. The two main issue here are the treatment of feature names and the treatment of feature values. These two topics are covered in the following two subsections. Throughout the discussion we build on a single example, namely, an analysis of the word *Kind* 'child', occurring in the sentence *Das Kind war einsam*, 'The child was lonely', appearing in a modern German text. The feature-structure analysis that we want to represent may be diagrammed as follows:[2]

```
+---                                     --+
| category  = noun                         |
|                                          |
| wordForm  = Kind                         |
|                                          |
| proper    = -                            |
|             +--                  --+     |
| agreement = | gender = neuter      |     |
|             | number = singular    |     |
|             | case   = nominative  |     |
+--           +--                  --+  --+
```

That is, the category feature has the value "noun", the wordForm feature has the value "Kind", the proper feature (for, "Is it a proper noun?") has the value "–" (or minus), and the agreement feature holds an embedded feature structure which groups the gender, number, and case features. The values for these represent neuter, singular, and nominative, respectively.

2.1 *Treatment of feature names*

SGML syntax suggests four alternatives for how the name of a feature might be marked up. The feature name could be: the name of an attribute of <fs>, the name of a tag embedded within <fs>, the content of a tag for feature name embedded in a general element for feature, or the value of an attribute of a general tag for feature. The following paragraphs consider the alternatives in turn and explain our reasons for rejecting the first three and favoring the last.

The attributes of an SGML element tag are very much like features. Could we use them to represent the features of a feature structure? For instance,

```
<fs  category=noun  wordForm=Kind
     proper=minus   agreement=???>
```

We immediately run into an insurmountable problem. Features must be able to embed feature structures as their values, but attributes do not allow tags to be embedded within them.

We could solve this problem by turning the attributes into tags. Thus each feature name would become a unique tag, as in:

```
<fs>
   <category>noun</category>
   <wordForm>Kind</wordForm>
   <proper>minus</proper>
   <agreement>
      <fs><gender>neut</gender>
         <number>sg</number>
         <case>nom</case>
      </fs>
   </agreement>
</fs>
```

The chief disadvantage of this solution is that it leads to an unbounded proliferation of tags. Every user of TEI markup would be required to modify the DTDs to add a new **<!ELEMENT ...>** definition for each needed feature. Note, too, that although embedding of feature structures within features becomes possible, it would only be allowed when the content model for a particular feature's **<!ELEMENT ...>** definition were written in such a way as to permit it.

To prevent the proliferation of tags, we need a single, general-purpose element to represent any feature. Its content model says once and for all that feature structures can be embedded within features. For this we have defined the **<f>** tag. Within the **<f>** tag, the name of the feature could be represented as the content of a tag for the feature name. For instance,

```
<fs>
   <f><fName>category</fName> ... </f>
   <f><fName>wordForm</fName> ... </f>
   <f><fName>proper</fName> ... </f>
   <f><fName>agreement</fName> ... </f>
</fs>
```

This was essentially the proposal in TEI P1. However, it turns out to be needlessly verbose. It also provides more power than we need, in that it suggests the possibility of embedded structure within the **<fName>** tag. In actual fact, no embedding is possible.

This leads to our final solution, namely, that a feature's name is the value of an attribute (called *name*) of the **<f>** tag. That is,

```
<fs>
   <f name=category> ... </f>
   <f name=wordForm> ... </f>
   <f name=proper> ... </f>
   <f name=agreement> ... </f>
</fs>
```

2.2 *Treatment of feature values*

Similarly, SGML syntax suggests three ways to markup the value of a feature. The feature value could be: the value of an attribute of the **<f>** tag, textual content of the **<f>** tag, or embedded tags in the **<f>** tag. The following paragraphs consider the alternatives in turn and explain our reasons for rejecting the first two and favoring the last.

The first possibility might be to treat *value* as an attribute of <f> analogous to the *name* attribute, as in:

```
<fs>
   <f name=category value=noun>
   <f name=wordForm value=Kind>
   <f name=proper value=minus>
   <f name=agreement value=???>
</fs>
```

This leads to the same dead-end as treating the feature name as an attribute name. The feature value must be able to embed a feature structure and this is impossible in the value of an attribute.

In order to allow feature structures to be the values of features, we must include the value of the feature as the content of the <f> tag. The simplest approach would be to treat values that were not feature structures as textual content of the <f> tag (that is, allow PCDATA or <fs> in the content model of <f>). For instance,

```
<fs>
   <f name=category>noun</f>
   <f name=wordForm>Kind</f>
   <f name=proper>minus</f>
   <f name=agreement>
      <fs><f name=gender>neut</f>
         <f name=number>sg</f>
         <f name=case>nom</f>
      </fs>
   </f>
</fs>
```

This solution falls short of being ideal in three respects. First, the content model for <f> exhibits the so-called "mixed content" problem. Although the SGML standard does not prohibit mixing PCDATA and tagged elements in the same content model, mixed content is known to pose problems for some processors and thus many practitioners advise against using mixed content models. Second, when the content model contains PCDATA, every space and line break within the contents of the tag is interpreted as part of the data; this makes it impossible to encode feature structures in "pretty-printed" indented formats that transparently reflect the structure of the embedded feature structures. Third, the markup treats all the textual content as the same kind of information, failing to distinguish symbols in closed sets (like noun) from arbitrary strings (like Kind) from universally defined concepts (like minus).

These problems are solved by requiring that all feature values be contained in tags. Therefore, the recommended markup for our example (following TEI P3) is as follows:

```
<fs>
   <f name=category><sym value=noun></f>
   <f name=wordForm><str>Kind</str></f>
   <f name=proper><minus></f>
   <f name=agreement>
      <fs><f name=gender><sym value=neut></f>
         <f name=number><sym value=sg></f>
         <f name=case><sym value=nom></f>
      </fs>
   </f>
</fs>
```

The content model for the definition of <f> in TEI P3 allows for the value of a feature to be any of the following:

(1) One of the universally defined feature values **<plus>** or **<minus>**. These are defined as empty tags (that is, they have no content or end tag).
(2) One of the (empty) tags for underspecified or uncertain values; **<any>** indicates that the value may be any valid value for the feature, **<default>** indicates that the value is the default value for the feature, and **<unknown>** indicates that the analyst does not know what the value is.
(3) A "typed" atomic value; namely, **<sym>** for a symbol drawn from a closed list, **<str>** for an arbitrary string, or **<nbr>** for a number.
(4) An embedded feature structure, that is, **<fs>**.
(5) Other kinds of structured values such as a list of values, or an alternation (disjunction) of values, or the negation of a value. The justification for our representation of such values is provided in the next section.

3. Justification for Treatment of Combinations of Features and Their Values

In this section, we justify the way in which we express the possibility that a feature may have more than a single value, or may have one of several alternative values. We also justify our method of indicating that a feature structure may have any of several different combinations of features. Finally, we justify our method of specifying that the value of a particular feature is other than some specified value.

Certain features that are used for linguistic analysis are understood to have more than one simple value. For example, a coordinate compound phrase such as *Bob and Carol* may be analyzed as having a members feature, whose value is a list of the phrases that are coordinated. Similarly, for a plural-referring expression, one may wish to provide an index feature, whose value is a set of numerical indices which abstractly indicate what things the expression refers to. One way of encoding such values would be to provide explicit tags such as **<list>** and **<set>** for enclosing their multiple values. For instance,

```
<f name=members>
   <list>
      <fs type=phrase>
         <f name=category><sym value=noun></f>
         <f name=barLevel><nbr value=2></f>
         <<f name=form><str>Bob</str></f>
      </fs>
      <fs type=phrase>
         <f name=category><sym value=noun></f>
         <f name=barLevel><nbr value=2></f>
         <f name=form><str>Carol</str></f>
      </fs>
   </list>
</f>
<f name=index>
   <set><nbr value=37><nbr value=128><nbr value=354></set>
</f>
```

However, such use of **<list>** and **<set>** tags, especially if **<list>** and **<set>** are permitted to nest within each other, provides for more descriptive power than is needed for linguistic representations. This is because whenever a list is contained as a value within another list of values, it does not do so directly but indirectly as the value of a feature within a feature structure in the original list. For instance, consider the coordinate compound phrase *Bob and Carol and Ted and Alice*, in which the members are the two compound subphrases *Bob and Carol* and *Ted and Alice* (one of eleven possible syntactic analyses of the whole phrase). The encoding of this would embed two <**list**>s (for the subphrases) within a <**list**> (for the whole phrase). However, the embedding is not direct; the members feature of the whole phrase would contain a list of two feature structures that would in turn have a members feature containing a list.

We conclude, therefore, that it is not necessary to provide a method of allowing a set or list to occur directly as an element of another set or list. We can then simplify the encoding of set and list values by providing the <**f**> tag with an attribute called *org* (for "organization"). Its possible values are drawn from "unit", "list", "set", and "bag", where a bag is defined as a collection of values that may contain repetitions but in which the order of the values is not significant. Thus, we encode the list and set values in our original examples as follows:

```
<f name=members org=list>
   <fs type=phrase>
      <f name=category><sym value=noun></f>
      <f name=barLevel><nbr value=2></f>
      <f name=form><str>Bob</str></f>
   </fs>
   <fs type=phrase>
      <f name=category><sym value=noun></f>
      <f name=barLevel><nbr value=2></f>
      <f name=form><str>Carol</str></f>
   </fs>
</f>

<f name=index org=set>
   <nbr value=37><nbr value=128><nbr value=354>
</f>
```

4. Alternation and Negation of Features and Values

In this section we justify our method of indicating that a feature structure may have any of several different combinations of features and that a feature may have any of several different combinations of values. We also justify our method of specifying that the value of a particular feature is other than some specified value.

4.1 *Alternation*

First, consider the problem of representing alternative values for a given feature, where we either do not know which value the feature has, or we wish to indicate explicitly that the structure containing that feature is ambiguous. For example, suppose that we wish to encode the fact that the case feature of the German noun *Kind* may have either the value "nom" or the value "acc" (where these values do not exhaust all the possible values for the case feature, so that the <**any**> tag cannot be used). One way that one might try to do this is simply to list both values, as in:

```
<f name=case><sym value=nominative><sym value=accusative></f>
```

However, this method of encoding multiple feature values has been reserved for the situation in which the feature is defined as taking a list or set of values, and we assume that "case" is not such a feature. Rather, we assume that "case" is defined as taking only a single value.

Thus it seems reasonable to define a tag which encloses the alternative values that a particular instance of a feature may have. The tag we have defined for this purpose is <**vAlt**> (for "value alternation"). Then the encoding of the case feature becomes:

```
<f name=case>
   <vAlt>
```

```
        <sym value=nominative>
        <sym value=accusative>
    </vAlt>
</f>
```

In addition to specifying alternations of values, one may also wish to specify alternations of features within a feature structure, as in encoding the grammatical structure of the English verb form *were*, which may be understood as having either second person or plural number. Since the potential content of this alternation tag must be different from that of the <**vAlt**> tag, we must give it another name, for example <**fAlt**> tag (for "feature alternation"), as follows:

```
<fAlt>
    <f name=person><sym value=second></f>
    <f name=number><sym value=plural></f>
</fAlt>
```

A more difficult problem is posed by the third-person German pronoun *sie*, which may be understood as having either nominative or accusative case and as having either singular number and feminine gender or plural number and any gender. How might one encode the grammatical structure of this word as a single feature structure? A first attempt, which makes use of both the <**vAlt**> and the <**fAlt**> tags, is the following:

```
<fs type='word structure' n=sie>
    <f name=category><sym value=pronoun></f>
    <f name=person><sym value=third></f>
    <f name=case>
        <vAlt><sym value=nominative><sym value=accusative></vAlt></f>
    <fAlt>
        <f name=number><sym value=singular></f>
        <f name=gender><sym value=feminine></f>
        <f name=number><sym value=plural></f>
        <f name=gender><any></f>
    </fAlt>
</fs>
```

However, since any number of features can combine to form a feature structure, this encoding does not make clear how many feature alternatives there are, and where one alternative ends and the next one begins.[3] One way to specify how many feature alternatives there are and where each one ends and the next one begins would be to insert a special empty tag at the points of demarcation. If we call that tag <**delim**> (for "delimiter"), the encoding of our example becomes:

```
<fs type='word structure' n=sie>
    <f name=category><sym value=pronoun></f>
    <f name=person><sym value=third></f>
    <f name=case>
        <vAlt><sym value=nominative><sym value=accusative></vAlt></f>
    <fAlt>
        <f name=number><sym value=singular></f>
        <f name=gender><sym value=feminine></f>
        <delim>
        <f name=number><sym value=plural></f>
        <f name=gender><any></f>
    </fAlt>
```

```
</fs>
```

This solution, however, does not identify the terms of the alternation (that is, the feature groups which are in alternation) as markup elements with which attributes (such as an *id* attribute, which is useful to indicate disambiguation) can be associated. To overcome these difficulties, we simply use untyped <**fs**> tags to enclose each group of features in the alternation. Our encoding for this example is as follows:

```
<fs type='word structure' n=sie>
   <f name=category><sym value=pron></f>
   <f name=person><sym value=third></f>
   <f name=case>
      <vAlt><sym value=nom><sym value=acc></vAlt></f>
   <fAlt>
      <fs>
         <f name=number><sym value=sg></f>
         <f name=gender><sym value=fem></f>
      </fs>
      <fs>
         <f name=number><sym value=pl></f>
         <f name=gender><any></f>
      </fs>
   </fAlt>
</fs>
```

Thus the content mode1 for <**fAlt**> allows both <**f**>s and <**fs**>s; the alternation is among all the top-level elements occurring within the <**fAlt**>.

We face a similar problem of potential ambiguity when there is a value alternation for a feature that takes multiple values. For instance, consider the example given above of a feature named index which holds a set of numbers that give referential indices to the persons or things referred to by some expression. If we had a plural pronoun which ambiguously referred either to referents 37 and 52 or to referents 128 and 354, we might encode it as follows using a <**vAlt**> with the empty tag <**delim**> to delimit the terms of the alternation:

```
<f name=index org=set>
   <vAlt>
      <nbr value=37>
      <nbr value=52>
   <delim>
      <nbr value=128>
      <nbr value=354>
   </vAlt>
</f>
```

However, as explained above, we have not adopted such a <**delim**> tag. Our solution in this case is to normalize the complex <**vAlt**> into an <**fAlt**> by distributing the feature specification over both terms of the altemation. Our encoding for this example is as follows:

```
<fAlt>
   <f name=index org=set>
      <nbr value=37>
      <nbr value=52>
   </f>
   <f name=index org=set>
      <nbr value=128>
      <nbr value=354>
    </f>
  </fAlt>
```

4.2 *Negation*

Next, suppose that we wish to represent the value for a feature in a particular feature structure as being distinct from some specific value; for instance, that the value for case is distinct from "nominative". One possibility would be to provide a <**neg**> (for "negation") tag as part of the content model for feature values, so that it could be used as in the following example:

```
<f name=case><neg><sym value=nominative></neg></f>
```

However, there are a number of reasons not to specify distinctness from a particular value by means of a separate tag such as <**neg**>. First, it is not desirable to permit <**vAlt**> within the scope of such a tag. Second, there are other relations of distinctness that are useful to encode besides distinctness from a particular value. For example, for a feature that takes number values, it would be handy to be able to state that the value is greater than (or less than, or greater than or equal to, or less than or equal to) the value specified in the <**nbr**> tag. More importantly, for feature-structure values, one would like to be able to specify, among other things, that the feature value is any value that is subsumed by the one that actually appears, or is any value that the actually appearing value does not subsume.[4]

These considerations suggest that the specification for distinctness from a particular value should be expressed as the value of an attribute on the value tag itself. We name the attribute *rel* (for "relation"), and define the property of being distinct from the specified value as the "ne" (for "not equal") value for the *rel* attribute. Other relations appropriate for numerical and string values are "gt" (for "greater than"), "ge" (for "greater than or equal to"), and "It" and "le" for the "less than" and "less than or equal to" relations.

The default value for *rel* for value tags other than <**fs**> is "eq" (for "equal"); for <**fs**> values, it is "sb" (for "subsumes"). This means that when the desired feature value is the value actually specified, or in the case of an <**fs**> value, is any value subsumed by the specified value, the rel attribute can be omitted. Accordingly, our current example may be encoded as follows, and none of our preceding examples need altering to take into account the *rel* attribute.

```
<f name=case><sym rel=ne value=nominative>
```

Of course, this encoding by itself does not state what the possible values for the case feature are. To determine what they are, we need to supplement the feature-structure encoding with a feature system declaration (see section 6) which declares what the possible values of the case feature are. If they are declared to be the symbolic values "nominative", "genitive", "dative", and "accusative", then the above encoding is equivalent to the following:

```
<f name=case>
  <vAlt>
    <sym value=genitive><sym value=dative><sym value=accusative>
  </vAlt>
</f>
```

5. From Linguistic Markup to General Data Markup

As argued in section 1, feature structures are a fundamental mechanism for analysis in the field of linguistics, used not only in many subdisciplines of linguistics but also used across a wide variety of competing theories. In fact, "feature structures" have a much wider applicability than just linguistics. They can provide for the encoding of information of nearly any sort, since the data structure in question is the familiar and general one described by Knuth (1968) which consists of "nodes" (here called feature structures) and "fields" (here called features). About such structures, Knuth writes:

> ... the ideas we have encountered are not limited to computer programming alone; they apply more generally to everyday life. A collection of nodes containing fields, some of which point to other nodes, appears to be a very good abstract model for structural relations of all kinds; it shows how we can build up complicated structures from simple ones, and we have seen that corresponding algorithms for manipulating the structure can be designed in a natural manner. (p. 462)

A similar sentiment is echoed by Shieber (1986) who is writing specifically about the application of feature structures in the representation of grammatical formalisms. After explaining how six different formal approaches to grammar can be handled by the single computational model of unification of feature structures, he says:

> In fact, viewed from a computational perspective, it is not surprising that so many paradigms of linguistic description can be encoded directly with generalized feature/value structures of this sort. Similar structures have been put forward by various computer scientists as general mechanisms for knowledge representation (Ait-Kaci, 1985) and data types (Cardelli, 1984). Thus we have hardly constrained ourselves at all even though limited to this methodology. (p. 10)

Feature structures with features are essentially equivalent to many familiar schemes of data organization like records with fields, objects with attributes, frames with slots, property lists with properties, and even abstract data types with access functions. <**fs**> and <**f**> can thus be used to markup records, objects, frames, property lists, or instances of abstract data types, if such a perspective is more to the liking of the encoder. In fact, the tag names <**fs**> and <**f**> were deliberately chosen to allow the alternate reading of "field structure" and "field". For instance, the following might be the encoding of a bibliographic record:

```
<fs type=book>
   <f name=author><str>Goldfarb, Charles</str></f>
   <f name=year><nbr value=1990></f>
   <f name=title><str>The SGML Handbook</str></f>
   <f name=publisher><str>Clarendon Press</str></f>
   <f name=pubPlace><str>Oxford</str></f>
</fs>
```

We hasten to note that we have made no attempt in developing <**fs**> markup to include all the features of a database system. Rather, we have only sought to make the markup elements developed for linguistic markup as generally useful as possible. There are two aspects of the markup that particularly cater to the needs of more general use: (1) The <**fs**> element has a *type* attribute which makes it possible to distinguish different types of records or objects. (Note that the above example specifies type=book.) (2) An <**fsLib**> (for "<**fs**> library") element is also defined. Its content model allows a set of <**fs**> elements, permitting the encoder to encapsulate a whole data table (in which all the structures are of the same type) or even an entire database (containing structures of multiple types).

6. Justification for Feature System Declarations (FSDS)

One of the chief benefits of SGML markup is that the DTD (document type definition) provides formal documentation of the markup system used in an encoded document. SGML-aware software can in turn use the DTD to validate the tagging in the document. Much of this benefit is lost in the proposed mechanism for feature-structure markup. As explained in section 2, in order to avoid the proliferation of tags and the requirement that every user of feature-structure markup would need to define a customized DTD, a completely generic approach was devised. It treats all application-specific names as values of *type*, *name*, and *value* attributes. As a result the DTD for feature-structure markup can only validate the syntactic integrity of <**fs**> and <**f**> constructs, but it cannot validate their semantic integrity. In fact, it would not even be possible to write a customized DTD to fully validate a particular system of feature structure markup since there is no way in a DTD to specify how the value of an attribute should constrain the content model of its tag or the attribute values in embedded tags. Such capabilities would be needed, for instance, to ensure that only valid types of values were specified for each feature in a marked up document, or that all the features specified in any given feature structure were applicable to that type of structure.

To fill this gap we have devised an auxiliary document called a feature system declaration, or FSD. An FSD is an SGML-encoded document which provides a formal means for the developer of a particular system of feature-structure markup to specify the following:

1) a list of all the feature names with a brief prose description of what they represent;
2) a declaration of what values are valid for each feature;
3) a list of all the feature-structure types with a brief prose description of what they represent;
4) a declaration of what features are applicable to each type of feature structure;
5) a declaration of well-formedness constraints on feature structures which either require or prohibit cooccurrence of specified feature-value pairs.

We presume that application software will be developed that can parse the FSD and use the information in it to validate the integrity of every feature structure in a marked-up document.

An FSD fills another kind of gap. When linguists use feature structures, they typically specify the minimum set of features that distinctively encode an object or concept. Redundancy rules are used to define the appropriate values for the unspecified features. In order to make it possible for <**fs**> markup to follow this practice, an FSD also allows the designer of a system of feature-structure markup to specify:

6) a default value for each feature. The default may be a constant value, or it may be an expression that generates a value contingent on other feature values in the feature structure.

Application software that parses the FSD could use this information to generate the missing feature values in underspecified feature structures. Another aspect of underspecification is the <**any**> value of a feature. Application software can use the specification of feature value ranges in the FSD (number 2 in the above list) to expand <**any**> into a disjunction of the valid feature values.

We do not here explain the details of FSD syntax; the reader is referred to the chapter on Feature System Declarations in TEI P3 for the formal definition. However, an example is given below in section 7.2.

7. Using the Recommendations

We close by illustrating how the recommendations for linguistic markup can be used in practice. The first illustration demonstrates tagging all the words in a corpus for lexical category. The second demonstrates the encoding of an interlinear analysis of a text down to the morpheme level.

7.1 *Lexical tagging*

There are many different reasons why one may wish to mark up text for linguistic analysis. One, which motivated some of the very earliest schemes for linguistic markup like the one used for the tagged Brown Corpus (Kucera and Francis, 1967), is to provide enough information about the words in a text to at least partially disambiguate them. For instance, a given occurrence of the sequence *wash sinks* in an English text might be analyzed as a singular noun followed by a present-tense verb, or as a "bare" verb form followed by a plural noun. A text in which the words are marked up to indicate how they are to be grammatically interpreted could support more sophisticated methods of information retrieval and of text-to-speech mapping than one not so marked up. They could also be used in an effort to induce a grammar from textual material.

The tradition in linguistic lexical markup has been to associate mnemonic codes with each lexical item in the text, where the code picks out a unique grammatical analysis for the lexical item. All such coding schemes reflect to varying degrees the fact that lexical structure can be broken down into discrete bundles of information (what we have here called "features" and their associated values), but none of them are fully systematic in the way in which they analyze lexical structure. All of them also use rather primitive means of associating structural markup with text elements. For example, a passage containing the sequence *The wash sinks* may be encoded as follows, where "$" followed by a three-letter code provides grammatical information about the immediately preceding word.

```
The$DET wash$NNS sinks$VBZ
```

It is evident that this encoding, together with a code book that explains the coding scheme, provides enough information to determine that *wash sinks* in this case is a noun-verb sequence rather than a verb-noun one. It also provides some additional information, for example, that the noun is singular in number and that the verb form has an *-s* ending. However, the amount of detail that can be provided is limited by the encoding scheme, and any extensions to provide for a more detailed grammatical analysis of individual words would require that the entire coding scheme be revised. Moreover, the method of associating grammatical representations with individual words cannot be extended to provide for the encoding of higher levels of grammatical structure, such as phase structure. By using feature-structure representations, we are able to encode lexical structure in an easily extensible way, and also to embed them in larger structures of the same sort in the representation of multiword units. In this section, we take up first the problem of representing lexical structures as feature structures, and second the problem of associating those structures with text elements.

The grammatical structure of the verb interpretation of *sinks* may be encoded as follows (note that this provides considerably more information than is contained in the code "$VBZ"):

```
<fs type=‘word structure’ id=vbidprx0sgp3>
   <f name=category><sym value=verb></f>
   <f name=mood><sym value=indicative></f>
   <f name=tense><sym value=present></f>
   <f name=auxiliary><minus></f>
   <f name=agreement>
      <fs type=‘agreement structure’ id=sgp3>
         <f name=number><sym value=singular></f>
         <f name=person><sym value=third></f>
      </fs>
   </f>
</fs>
```

We may also assume that a feature system declaration has been provided that both legitimates this particular feature structure and specifies the value ranges of all the features mentioned in it.

Next, consider the problem of associating this particular feature structure with a textual occurrence of the word *sinks*. Clearly it would be undesirable to insert the feature structure next to the occurrence of the word in the text, since compared with the codes used for traditional text tagging, the feature-structure representations are very large. However, as the example indicates, each feature structure can be given a unique identifier (by means of the *id* attribute). This allows it to be pointed to from within a tag associated with a text element.

Thus, suppose we use the <**w**> tag (the "word" tag provided in the TEI analytic tag set) to identify the lexical items in our text, and that we assign each one a unique *id* attribute. Suppose also that we collect up a single instance of every lexical feature-structure representation that we require for marking up our text, and put them in a section of the document introduced by the <**fsLib**> (for "feature-structure library") tag.[5] Then one can use the *ana* attribute (for "analysis") on the <**w**> tag to point to the feature structure that represents its lexical analysis, as in the following example (which assumes that the 447th word of the text is *sinks* and is used as a verb).

```
<w id=w447 ana=vbidprx0sgp3>sinks</w>
```

Given that we have provided every segment with an *id* attribute as well as an *ana* attribute, we can also use the *inst* attribute on each <**fs**> in the feature-structure library to identify every lexical instance of that particular feature structure. So, for example, if the segments identified as “w35”, “w292”, “w447” and “w601” are all associated with the feature structure under discussion, we can make that fact explicit in the begin tag for that feature structure as follows:

```
<fs type=‘word structure’ id-vbidprx0sgp3 inst=‘w35 w292 w447 w601’>
```

Just as we can have more than one lexical instance associated with a particular feature structure, so we can have more than one feature structure associated with a particular lexical instance. If we do, we would be explicitly representing lexical ambiguity. So, for example, if “nncmp1” is the value of the *id* attribute for the feature structure that represents the structure of a common plural noun, then our instance of the word *sinks* could be marked as ambiguous between these two interpretations as follows:

```
<w id=w447 ana=‘nncmpl vbidprx0sgp3’>sinks</w>
```

The method of relating lexical instances to lexical analyses by means of the *ana* and *inst* attributes on <**w**> and <**fs**> respectively, while superior to the method of including structural representations directly within texts, still has a number of drawbacks. It is quite cumbersome, it is very difficult to check for errors, and it may not be usable in a hypertext environment where the text and the feature-structure library are in different files.

A third method, which offers an easier approach to entering and editing relationships between text elements and structural elements and which may be used in a hypertext environment, employs the linking mechanism that has been provided for a variety of analytical purposes including the alignment of parallel texts. As with the method that uses *ana* and *inst* attributes as pointers, the linking mechanism requires that both the lexical instances and the feature structures be provided with unique identifiers. One may also identify the domains within which the links are being made; in the present case these would be the text containing the lexical instances whose analyses are being specified, and the feature-structure library which contains those analyses. The linking mechanism consists of a <**linkGrp**> tag whose content is a collection of <**link**> tags. The <**linkGrp**> tag has a *domains* attribute, which consists of a list of identifiers for the domains within which the links are being specified. For example, suppose that the text containing the lexical items whose analyses are being marked is a newspaper column whose begin tag is specified as follows:

```
<div3 type=‘column’ id=nyt840404s2plc3n2>
```

Suppose also that the begin tag for the relevant feature-structure library is the following:

```
<fsLib id=enlxst2>
```

Then the <**linkGrp**> begin tag would be specified as follows:

```
<linkGrp domains=‘nyt840404s2plc3n2 enlxst2’>
```

The actual association of specific lexical instances with feature structures is specified by <**link**> tags which contain a *targets* attribute which in this case would contain two pointer values, the first to an element within the identified newspaper column and the second to an element within the identified feature structure library. Thus, to indicate that the segment identified within the text as “w447” has the analysis identified within the feature-structure library as “vbidprx0sgp3”, one would provide the following <**link**> tag within the <**linkGrp**> tag mentioned above.

```
<link targets=‘w447 vbidprx0sgp3’>
```

All of the links provided within this particular **<linkGrp>** are one-to-one.[6] Thus each of the associations between a particular lexical instance and a particular feature structure are specified by a separate **<link>** tag. Lexical ambiguity may be indicated by separate **<link>** tags with the same first identifier for the lexical instance and different second identifiers for the feature-structure instances. In the case of the lexical instance identified as "w447", to say that it may be represented either as having the analysis identified as "vbidprx0sgp3" or as "nncmpl", one would provide both the preceding and the following **<link>** tags.

```
<link targets='w447 nncmpl'>
```

An important advantage of the linking mechanism is that no structural information needs to be recorded in the text itself, other than the identification of segments to be aligned with their analyses; nor does information about the text have to be represented in the begin tag of particular feature structures. To return to our hypothetical text whose 447th word is "sinks" used as a verb, we need only to provide a segment tag with a unique *id* attribute, as in:

```
<w id=w447>sinks</w>
```

Similarly, the begin tag of the feature structure for the relevant inflected form of the verb requires no *inst* attribute, as in:

```
<fs type='word structure' id=vbidprx0sgp3>
```

To represent the information that the segments identified as "w35", "w292", "w447", and "w601" are all associated with this particular feature structure, a **<linkGrp>** with at least the following **<link>** tags is needed:

```
<linkGrp domains='nyt840404s2plc3n2 enlxst2'>
  ...
  <link targets='w35 vbidprx0sgp3'>
  <link targets='w292 vbidprx0sgp3'>
  <link targets='w447 vbidprx0sgp3'>
  <link targets='w601 vbidprx0sgp3'>
  ...
</linkGrp>
```

As a result, when an analysis is changed or extended, no changes need to be made to the text (other than to add **<w>** tags with unique identifiers). Only the links need to be changed or extended, and feature structures changed or added.

Another important advantage of the linking mechanism is that it can be easily adapted to the situation in which one wishes to associate a variety of different types of information with a particular lexical instance. For example, suppose one also wished to associate with each lexical instance a pointer to a dictionary entry. Assuming that the dictionary containing the relevant entries has a unique identifier that can be pointed to from within the document containing the **<linkGrp>**, one would add that identifier to the list of identifiers in the value of its *domains* attribute. Then within each **<link>** in that **<linkGrp>**, appropriate reference would be made within the *targets* attribute to the identifier for the dictionary entry.

7.2 *Interlinear text*

Linguists, anthropologists, literary scholars, and others who deal with large corpora of foreign-language text, have traditionally used interlinear annotation as a mechanism both for developing and for presenting the morphological analysis and literal glossing of running text; see, for instance, Antworth (1993). As an example, consider the following sentence in the Eskimo language of northwest Alaska:

Akutchilighmik-uvva uqaaqtullangniaqtunga. 'I am going to tell a story about making Eskimo ice cream.'

(The digraphs *ng* and *gh* are used here to represent the "eng" and "dotted-g", respectively, of the standard orthography.) Consider that this sentence is the first sentence in a text that we want to analyze. An analysis of this sentence in interlinear format might look something like this:

1.	Akutchilighmik-uvva	uqaaqtullangniaqtunga.
	akut -chi-ligh-mik =uvva	uqaaqtu -llang-niaq-tunga
	akutuq -si -liq -mik =uvva	uqaaqtuq -llak -niaq-tunga
	icecream-RSL-GER -s.MOD=now	tell story-DUR -INT -ls.I
	about making Eskimo ice cream	I am going to tell a story

The first line gives the baseline text. The second line inserts morpheme breaks into the surface forms of the words; a hyphen breaks off a suffix and an equal sign breaks off an enclitic. The third line gives the underlying form for each morpheme. The fourth line gives a gloss for each morpheme; grammatical functors are glossed with technical abbreviations. The last line gives a gloss for each word.

In this example, there are structures at three levels; there is one sentence, two words, and many morphemes. In the encoding, these levels correspond to different types of feature structures. Thus, the sentence is encoded as an instance of <**fs type=sentence**>, and so on. The structure for the sentence has three features: "text" gives the text of the sentence in normal orthography, "translation" gives the free translation of the sentence, and "analysis" breaks the sentence into a sequence of analyzed words (that is, instances of <**fs type=word**>). The reference number for the sentence can be encoded as the value of the *n* attribute that is built into all TEI elements. The structure for the words also has three features: "form", "gloss", and "analysis". The sentence would then be encoded as follows (omitting for the moment instances of the <**fs type=morpheme**> tag, which encode the analysis of each word):

```
<fs type=sentence n=1>
   <f name=text>
      <str>Akutchilighmik-uvva uqaaqtullangniaqtunga.</str></f>
   <f name=translation>
      <str>I am going to tell a story about making Eskimo ice cream.</str></f>
   <f name=analysis>
      <fs type=word>
         <f name=form><str>akutchilighmik-uvva</str></f>
         <f name=gloss><str>about making Eskimo ice cream</str></f>
         <f name=analysis>...</f>
      </fs>
      <fs type=word>
         <f name=form><str>uqaaqtullangniaqtunga</str></f>
         <f name=gloss><str>I am going to tell a story</str></f>
         <f name=analysis>...</f>
      </fs></f>
</fs>
```

To handle the morphemic structure of the words, we encode the analysis of each word as a sequence of <**fs type=morpheme**> tags. These feature structures contain four features: "type" identifies it as a root, suffix, and so on (thus encoding the information carried by the hyphens and equal signs); "form" gives the surface form of the morpheme; "lexForrn" gives the lexical (or underlying) form of the morpheme; and "gloss" gives its gloss. "Type" is drawn from a small closed set so its values are encoded as symbols; all others come from open sets and are encoded as strings. The markup for the first word in the sentence is thus:

```
<fs type=word>
   <f name=form><str>akutchilighmik-uvva</str></f>
   <f name=gloss><str>about making Eskimo ice cream.</str></f>
   <f name=analysis>
      <fs type=morpheme>
         <f name=type><sym value=root></f>
         <f name=form><str>akut</str></f>
         <f name=lexForm><str>akutuq</str></f>
         <f name=gloss><str>ice cream.</str></f></fs>
      <fs type=morpheme>
         <f name=type><sym value=suffix></f>
         <f name=form><str>chi</str></f>
         <f name=lexForm><str>si</str></f>
         <f name=gloss><str>RSL</str></f></fs>
      <fs type=morpheme>
         <f name=type><sym value=suffix></f>
         <f name=form><str>ligh</str></f>
         <f name=lexForm><str>liq</str></f>
         <f name=gloss><str>GER</str></f></fs>
      <fs type=morpheme>
         <f name=type><sym value=suffix></f>
         <f name=form><str>mik</str></f>
         <f name=lexForm><str>mik</str></f>
         <f name=gloss><str>s.MOD</str></f></fs>
      <fs type=morpheme>
         <f name=type><sym value=enclitic></f>
         <f name=form><str>uvva</str></f>
         <f name=lexForm><str>uvva</str></f>
         <f name=gloss><str>now</str></f></fs>
   </f>
</fs>
```

The FSD (feature system declaration) for the above system of markup would contain an <**fsDecl**> for each of the three types of feature structures. Each <**fsDecl**> contains an <**fDecl**> for each of the features that can occur in that type of feature structure. These are very straightforward in this example. Worthy of special note is the <**fDecl**> for the "type" feature of morphemes; it specifies a closed list of the values that can occur in Eskimo words. Note, too, that the <**fDecl**> tags for the "analysis" features specify that the *org* attribute has the value "list". This is what allows the content of those features in the marked-up text to contain a sequence of values, rather than just a single value. The feature-structure declarations (extracted from the full FSD) for this example are therefore as follows:

```
<fsDecl type=sentence>
   <descr>Represents the interlinear analysis of a sentence</descr>
   <fDecl name=text>
      <descr>The text of the sentence in normal orthography</descr>
      <vRange><str rel=sb></str></vRange></fDecl>
   <fDecl name=translation>
      <descr>A free translation of the sentence</descr>
      <vRange><str rel=sb></str></vRange></fDcl>
   <fDecl name=analysis org=list>
      <descr>The analysis of the words in the sentence</descr>
      <vRange><fs type=word></fs></vRange></fDecl>
</fsDecl>
<fsDecl type=word>
   <descr>Represents the interlinear analysis of a word</descr>
   <fDecl name=form>
      <descr>The word form in the text without capitalization or punctuation</descr>
```

```
    <vRange><str rel=sb></str></vRange></fDecl>
  <fDecl name=gloss>
    <descr>A literal translation of the word</descr>
    <vRange><str rel=sb></str></vRange></fDecl>
  <fDecl name=analysis org=list>
    <descr>The analysis of the morphemes in the word</descr>
    <vRange><fs type=morpheme></fs></vRange></fDecl>
</fsDecl>

<fsDecl type=morpheme>
  <descr>Represents the interlinear analysis of a morpheme</descr>
  <fDecl name=type>
    <descr>The type of morpheme</descr>
    <vRange><vAlt>
      <sym value=root><sym value=prefix>
      <sym value=suffix><sym value=enclitic></vAlt>
    </vRange></fDecl>
  <fDecl name=form>
    <descr>The surface form of the morpheme</descr>
    <vRange><str rel=sb></str></vRange></fDecl>
  <fDecl name=lexform>
    <descr>The lexical (underlying) form of the morpheme</descr>
    <vRange><str rel=sb></str></vRange></fDecl>
  <fDecl name=gloss>
    <descr>A literal gloss of the morpheme</descr>
    <vRange><str rel=sb></str></vRange></fDecl>
</fsDecl>
```

Notes

[1] The initial feature-structure recommendations were formulated by the Analysis and Interpretation Committee at a meeting in Tucson, Arizona in March 1990, following suggestions by Mitch Marcus and Beatrice Santorini. The authors received valuable help in the further revision and refinement of the recommendations from Steven Zepp.

[2] Note that we are supposing that the source text does not contain any explicit linguistic analysis. If the source text were to contain this very feature-structure diagram, we might also want to add instructions for how to reproduce it graphically. Provision for this has not been made in the TEI P3 recommendations.

[3] In our example, an applications program (though not an SGML validator) could use the fact that the names of the features in the two alternatives are the same to infer what the alternatives are. However, the alternatives in a given feature alternation may involve features with different names.

[4] For discussion of subsumption, see Shieber (1986) and the relevant sections of TEI P3.

[5] This is one method that is recommended for gathering together feature structures. Another that ultimately will prove more useful for scholarly purposes is to provide them in publicly accessible documents that can be associated with one's text by means of hypertextual or other kinds of links. See the chapters on segmentation and alignment in TEI P3 for further discussion.

[6] This is not a necessary feature of the linking mechanism proposed in TEI P3. For instructions about how to state one-many and many-many linking relations, see the chapter on Segmentation and Alignment.

References

Ait-Kaci, H. *A New Model of Computation Based on a Calculus of Type Subsumption.* Doctoral dissertation. University of Pennsylvania, 1985.

Antworth, Evan L. "Glossing Text with the PC-KIMMO Morphological Parser". *Computers and the Humanities,* 26 (1993), 389–98.

Cardelli, L. *A Semantics of Multiple Inheritance.* Technical Report. Murray Hill, NJ: Bell Laboratories, 1984.

Chomsky, Noam. *Aspects of the Theory of Syntax.* Cambridge, MA: MIT Press, 1965.

Jakobson, Roman. "The Identification of Phonemic Entities". *Travaux du Cercle Linguistique de Copenhague 5* (1949), 205–13.

Knuth, Donald E. *The Art of Computer Programming. Vol. 1: Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1968.

Kucera, Henry and W. Nelson Francis. *Computational Analysis of Present-day American English*. Providence, RI: Brown University Press, 1967.

Shieber, Stuart. *An Introduction to Unification-Based Approaches to Grammar*. Chicago: University of Chicago Press, 1986.

Sperberg-McQueen, C. M. and L. Burnard. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Chicago and Oxford: Text Encoding Initiative, 1994.