# Comments on Feature Structure Draft

Éric de la Clergerie
INRIA – Eric.De_La_Clergerie@inria.fr

July 17, 2003

This very preliminary document collects a few comments about the current draft about Feature Structures Representation and Declaration (ISO TC37 SC4 N33).

## 1 About FS Lite

The revised draft should show either explicitly or through its structure that there are several layers of increasing complexity when using the proposed Feature Structure representation scheme.

For instance, we can identify the following layers:

1. Using untyped feature structures with basic values and no recursion (only need of *fs* without attribute and *f* with attribute *name*). This layer should actually cover most usages where people only want to express a set of basic properties on objects.
2. Adding disjunctions on values (the other kinds of disjunctions may come later)
3. Adding libraries and compact notations
4. Adding types
5. Adding recursion and reentrency
6. Adding complex grouping (alternatives, bags, lists and sets)
7. Adding declaration mechanisms

I think the 3 or 4 first layers should cover most common usages.

## 2 About atomic types

I don't believe that a draft about Feature Structures should enumerate a list of atomic types. Such a list would overlap with specification of atomic types in other place and would never be complete. For instance, someone may wish at some point to have dates as atomic type or any kind of formatted strings. XML schema are an alternative where atomic types may be defined. The FS draft should rather focus on how to use atomic types.

# 3 About reentrency

The current draft does not precise how reentrency is handled. One may think that XML references (through IDs) are a solution. However, I am afraid they are not, because of *renaming* problems and because of the use of FS libraries. Indeed, the reentrency points present in several occurrences of a same feature structure (from a library) and used in different places should be considered as distinct.

Xpointer notation seems to be a better alternative.

```
<fsLib>
  <fs id="fs1">
        <f name="f"><fs id="fs2"> ... </fs></f>
        <f name="g">
              <fs id="fs3">
                    <f name="h" sharing="../../f[@name=
                        f]"/>
                    ...
              </fs>
  </fs>
  ...
</fsLib>
```

Notes:

- should look for a linguistic example
- not sure about the best notation (an attribute *sharing* of *f*) or some new element inside *f*

An alternate and more readable notation exists, not using Xpointer, is possible.

```
<fsLib>
  <fs id="fs1">
        <f name="f" var="X"><fs id="fs2"> ... </fs></f>
        <f name="g">
              <fs id="fs3">
                    <f name="h" var="X"/>
                    ...
              </fs>
  </fs>
  ...
</fsLib>
```

This notation has the advantage of beeing symmetric but says nothing about the scope of variable **X**. For simple cases, it may be assumed that it is the topmost feature structure containing the variable but this is not a fully acceptable answer.

# 4 Looking for examples

Examples about complex value organization (such as bags and sets) and about reentrency may be found within HPSG grammars. Other examples may also be found in LFG. In particular, LFG could illustrate some use of attribute *rel* to handle $=_c$ check condition (subsumption relation).

# 5 XML as a third kind of FS representation

Actually the proposed XML representation is essentially a rephrasing of the AVM notation.

This remark suggests that, maybe, some extensions of the formalism should be added to handle the alternate *path notation* for FS where you may have the following kinds of equations:

- $\mathrm{path} = \mathrm{value}$
- $\mathrm{path} = \mathrm{path}$

Actually, the really missing part is about stating equations between paths and this issue is also related to the problem of reentrency (e.g., naming nodes in a graph structure).

A possible notation, with a compact variation

```
<FSpath><f name="f"><f name="g"/></f></FSpath>
<FSpath path="f␣g"/>
```

The path notation often allows *path uncertainty* where parts of a path may be left unspecified, using star notation or some kind of regular expression (as may be found within LFG). In a way, this is similar to the XPath notation.

```
<FSpath>
    <f name="f">
        <fstar>
            <alt>
                <f name="g"/>
                <f name="h"/>
            </alt>
        </fstar>
    </f>
</FSpath>
```

Maybe the best path notation (but not the most readable one) would be to use the Xpath notation.

```
<FSpath path="f[@name='f']/f[@name='g']"/>
<FSpath path="f[@name='f']/(f[@name='g']|f[@name='h'])*
    "/>
```

**Note**: I am not sure about the star notation in XPath.

# 6 About declaring Feature Structure

At least two possibilities seem to exist.

The first one follows the current proposal and does not need anything else. Libraries may be used to define finite set of values and most general FS for each given type. It is even possible to state reentrency constraints on most general structure. Each occurrence of a FS for type $\tau$ in a document should then be an instance (by subsumption) of the declared one(s). There is usually one single most general FS for a given type but nothing really forbids to have several.

Declaring most general FS for each type is fine but may be a long process (when one has a long list of types and a long list of attributes for each type as found in HPSG). Furthermore, nothing is said about possible inheritance through the types. A second alternative may be to declare type hierarchies a la Carpenter. Each type specifies its sub-types and introduce (or refine) features associated with some most general types. The full set of features attached to a given type $\tau$ includes the features introduced by $\tau$ and all features introduced by its super types.

For instance, we could have something like below to specify lists of dates, using standof notation for recursion and multi-inheritance.

```
<FSHierarchy type="list_of_dates">
  <FStype name="top" id="top">
    <FStype name="list" id="list">
        <FStype name="e_list"/>
        <FStype name="ne_list">
           <f name="hd"> <FSatomic type="date"/> </f>
           <f name="tl"> <FStype idref="list"/> </f>
        </FStype>
    </FStype>
  </FStype>
</FSHierarchy>
```

Of course, for the special case of lists, we could have more efficient notations (to specify “list of something”).

The above notation is not incompatible with the specification of most general FS. A most general FS could be included in `FStype` (with no need to specify inherited features).

```
<FSHierarchy type="family">
  <FStype name="top" id="top">
    <FStype name="person" id="person">
        <f name="first_name"> <FSatomic type="string"/>
            </f>
        <f name="last_name"> <FStype idref="string"/> <
            /f>
        <FStype name="married">
          <fs type="married" var="X">
              <f name="last_name" var="N"/>
```

```
                <f name="spouse">
                    <fs type="married">
                        <f name="last_name" var="N"/>
                        <f name="spouse" var="X"/>
                    </fs>
                </f>
            </fs>
        </FStype>
    </FStype>
</FSHierarchy>
```

---

**Notes:**

- Maybe a more linguistic example could be found with head agreement in HPSG.
- This example should be rewritten using the correct syntax for reentrency.
- The interleave between `FStype` and `fs` is to precised.
- Maybe `f` elements inside `FStype` should be renamed into `fintro`.

# 7 A few philosophical remarks

It may noted a strong similarity between Feature Structure Representation and XML. Both representations are very general and describe tree structures with special mechanisms to handle reentrency. Both of them have types (element names for XML) and both of them need some kind of declaration mechanisms (DTDs or XML Schema for XML). In a way, FS are as complicated as XML to handle. On a other hand, it also means that XML techniques could be straightforwardly used to handle FS.