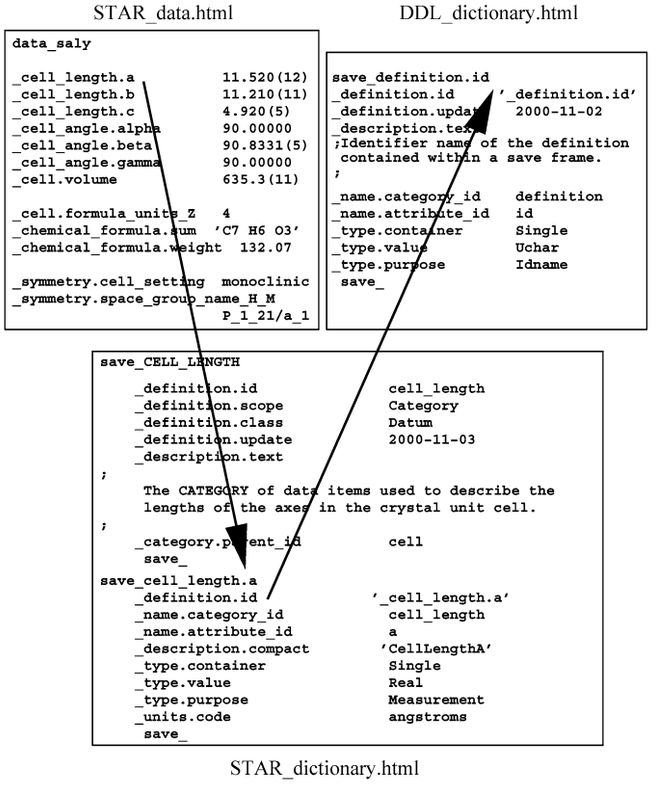

Fig. 5.2.5.1. Hyperlinking markup generated by *StarMarkUp*. The example is based on an extended relational dictionary definition language, StarDDL.

of the STAR File. The program understands only the rules of a legitimate STAR File.

StarMarkUp provides a number of hyperlinking facilities. During markup, it automatically hyperlinks frame-code values to the internal save-frame block to which they point. As part of the same process, *StarMarkUp* inserts anchors in all data and save-frame blocks. It does this in anticipation that there may be the need to hyperlink to these anchors from another STAR File. The most obvious application for this is in the marking up of the DDL dictionary. The list of anchors generated from this process can be passed on to the discipline dictionary during its markup phase (Fig. 5.2.5.1). In this way, the tags used in the discipline dictionary to define the data names can be made to point back to their entry in the DDL dictionary. At the same time that the discipline dictionary is being marked up, a list of its anchors is being generated (each anchor being to a data-item definition). This list is used when marking up an instance of the discipline STAR data file to hyperlink each data name back to its definition in the discipline dictionary.

StarMarkUp comprises a hand-crafted tokenizer employing a character buffer and one-character look-ahead to identify accepted tokens. *StarMarkUp* does not build an internal representation of the file, but functions as a streaming parser. The *GetToken()* function returns a structure consisting of the token type (an enumerated set) and the token value (the lexeme associated with that token). In most cases, the lexeme is marked up and injected into the output stream. If the token is associated with a STAR File block, the parser will recursively call a *MarkUpBlock()* function. A recursive descent parser makes it very easy to treat all STAR File blocks (`global_`, `data_` and `save_`) in identical fashion.

StarMarkUp is implemented in Tcl/Tk for novelty and not because of any particular superior qualities of the language and its API. It is, however, fast and sufficiently flexible, and extensions to the program can be rapidly implemented and tested.

5.2.6. Object-oriented STAR programming

The STAR File syntax is very simple but flexible, and suggests a number of well defined data structures. 'Scalar' values (*i.e.* single text strings identified by specific data names) are obvious, as are 'vectors' (multiple string values associated with a data name in a `loop_` declaration). 'Matrices' are constructed by associating identical-length vector items in the same `loop_` structure. Note that these data structures are not true matrices, but arrays or tables of column-addressable vectors. Column addressing arises from the need to associate each set of values with a distinct data name in the loop header declaration.

Through the nested loop construct in STAR, vector or matrix *elements* are permitted within vectors or matrices. Save frames provide addressable data structures containing one or more scalar, vector or matrix components; and data blocks contain similar data structures with the inclusion of embedded save frames.

This hierarchy of structure allows many other data models to be mapped onto the STAR syntax. For example, STAR does not in itself provide addressable *rows* in matrix structures, but such a concept may be achieved in application-specific ways. For example, the relational database structure of mmCIF and other DDL2 applications ensures that table rows can be identified by specific key values. Loops of multiple values associated with a single data name are similar to *lists* or one-dimensional *arrays* as defined in many computer languages. Simple *associative arrays*, or 'hashes', such as are used to great effect in Perl and Python, can be modelled by designating the first value of a two-item loop structure as the 'key' item, by analogy with the relational database example above (in such an application the key value of course needs to be unique). The hierarchy of nested loops can be used to model certain types of complex structures such as might be defined in a C-language 'struct', for example.

There is a natural possibility of representing particular well defined data structures as 'objects' and handling STAR Files by object-oriented programming techniques. Where applications associate properties of specific data items with external reference descriptions, as in the DDL dictionaries, the dictionary entries can be considered to express types, relationships and even methods associated with the object classes.

However, the versatility of the STAR syntax means that there are many ways of designing STAR objects and their relationships, and it is likely that the most useful efforts will be in designing applications for specific purposes or subject areas. Below we describe a number of prototypes and implementations of object-oriented STAR programming. Some are rather generic in outlook; others are informed by the crystallographic viewpoint embodied in CIF.

5.2.6.1. *OOSTAR*

The *OOSTAR* approach (Chang & Bourne, 1998) developed and tested an Objective-C (Pinson & Richard, 1991) toolset for STAR representation. The developers selected Objective-C over the more common C++ language because of its perceived advantages of (i) loose data typing, (ii) run-time type checking and (iii) message-passing ability. These were all seen to aid flexible software design, an asset for applications built on the very flexible structure of the STAR syntax itself.

Chang & Bourne constructed a set of classes built on the basic objects *item* and *value*. Their model builds upwards through a hierarchy of classes describing relationships between objects. The *ItemAssoc* class contains data members corresponding to the data item itself, its included value or set of values, and the data name that is used to reference the item; the class also has pointers to

references