International Tables for Crystallography (2006). Vol. G, Section 5.1.3.1, pp. 483–484.

and a mechanism for representing metadata (*e.g.* as dictionaries or schemas). Four are of particular importance in crystallography: CIF, ASN.1, HDF and XML.

As noted in Chapter 1.1, CIF was created to rationalize the publication process for small molecules. It combines a very simple tag–value data representation with a dictionary definition language (DDL) and well populated dictionaries. CIF is table-oriented, naturally row-based, has case-insensitive tags and allows two levels of nesting. CIF is order-independent and uses its dictionaries both to define the meanings of its tags and to parameterize its tags. It is interesting to note that, even though CIF is defined as order-independent, it effectively fills the role of an order-dependent markup language in the publication process. We will discuss this issue later in this chapter.

Abstract Syntax Notation One (ASN.1) (Dubuisson, 2000; ISO, 2002) was developed to provide a data framework for data communications, where great precision in the bit-by-bit layout of data to be seen by very different systems is needed. Although targeted for communications software, ASN.1 is suitable for any application requiring precise control of data structures and, as such, primarily supports the metadata of an application, rather than the data. ASN.1 can be compiled directly to C code. The resulting C code then supports the data of the application. ASN.1 notation found application in NCBI's macromolecular modelling database (Ohkawa *et al.*, 1995). ASN.1 has case-sensitive tags and allows case-insensitive variants. It manages order-dependent data structures in a mixed order-dependent/order-independent environment.

HDF (NCSA, 1993) is 'a machine-independent, self-describing, extendible file format for sharing scientific data in a heterogeneous computing environment, accompanied by a convenient, standardized, public domain I/O library and a comprehensive collection of high quality data manipulation and analysis interfaces and tools' (http://ssdoo.gsfc.nasa.gov/nost/formats/hdf.html). HDF was adopted by the Neutron and X-ray Data Format (NeXus) effort (Klosowski *et al.*, 1997). HDF allows the building of a complete data framework, representing both data and metadata. Two parallel threads of software development, focused on the management and exchange of raw data from area detectors, began in the mid-1990s: the Crystallographic Binary File (CBF) (Hammersley, 1997) and NeXus. The volumes of data involved were daunting and efficiency of storage was important. Therefore both proposed formats assumed a binary format. CBF was based on a combination of CIF-like ASCII headers with compressed binary images. NeXus was based on HDF. The first API for CBF was produced by Paul Ellis in 1998. CBF rapidly evolved into CBF/imgCIF with a complete DDL2 dictionary and a fully CIF-compliant API (Chapter 5.6). As of mid-2010, NeXus was still evolving (see http://www.nexusformat.org/).

XML is a simplified form of SGML, drawing on years of development of tools for SGML and HTML. XML is tree-oriented with case-sensitive entity names. It allows unlimited nesting and is order-dependent. Metadata are managed as a 'document type definition' (DTD), which provides minimal syntactic information, or as schemas, which allow for more detail and are more consistent with database conventions. In fields close to crystallography, the first effort at adopting XML was the chemical markup language (CML) (Murray-Rust & Rzepa, 1999). CML is intentionally imprecise in its ontology to allow for flexibility in development. The CSD and PDB have released their own XML representations (http://www.ccdc.cam.ac.uk/support/documentation/relibase/3_0/relibase_DPG/toc.html; http://pdbml.rcsb.org).

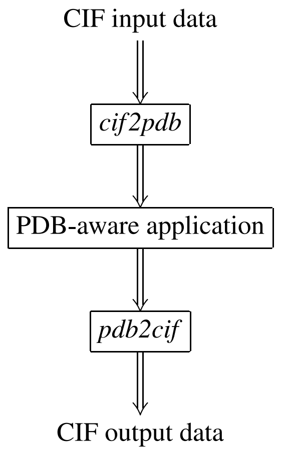

Fig. 5.1.3.1. Example of using filters to make a PDB-aware application CIF-aware.

It may seem from this discussion that the application designer faces an unmanageable variety of data frameworks in an unstable, evolving environment. To some extent this is true. Fortunately, however, there are signs of convergence on CIF dictionary-based ontologies and the use of transliterated CIFs. This means that an application adapted to CIF should be relatively easy to adapt to other data frameworks.

5.1.3. Strategies in designing a CIF-aware application

There are multiple strategies to consider when designing a CIF-aware application. One can use external filters. One can use existing CIF-aware libraries. One can write CIF-aware code from scratch.

5.1.3.1. Working with filter utilities

One solution to making an existing application aware of a new data format is to leave the application unchanged and change the data instead. For almost all crystallographic formats other than CIF, the Swiss-army knife of conversion utilities is *Babel* (Walters & Stahl, 1994). *Babel* includes conversions to and from PDB format. Therefore, by the use of *cif2pdb* (Bernstein & Bernstein, 1996) and *pdb2cif* (Bernstein *et al.*, 1998) combined with *Babel*, many macromolecular applications can be made CIF-aware without changing their code (see Figs. 5.1.3.1 and 5.1.3.2). If the need is to extract mmCIF data from the output of a major application, the PDB provides *PDB_EXTRACT* (http://sw-tools.pdb.org/apps/PDB_EXTRACT/).

Creating a filter program to go from almost any small-molecule format to core CIF is easy. In many cases one need only insert the appropriate '`loop_`' headers. Creating a filter to go from CIF to a particular small-molecule format can be more challenging, because a CIF may have its data in any order. This can be resolved by use of *QUASAR* (Hall & Sievers, 1993) or *cif2cif* (Bernstein, 1997), which accept request lists specifying the order in which data are to be presented (see Fig. 5.1.3.3).

There are a significant and growing number of filter programs available. Several of them [*QUASAR*, *cif2cif*, *ciftex* (ftp://ftp.iucr.org/pub/ciftex.tar.Z) (to convert from CIF to TEX) and ZINC (Stampf, 1994) (to unroll CIFs for use by Unix utilities)] are discussed in Chapter 5.3. In addition there are *CIF2SX* by Louis J. Farrugia (http://www.chem.gla.ac.uk/∼louis/software/utils/), to convert from CIF to *SHELXL* format, and *DIFRAC* (Flack *et al.*, 1992) to translate many diffractometer output formats to CIF. The program *cif2xml* (Bernstein & Bernstein, 2002) translates from CIF to XML and CML. The PDB

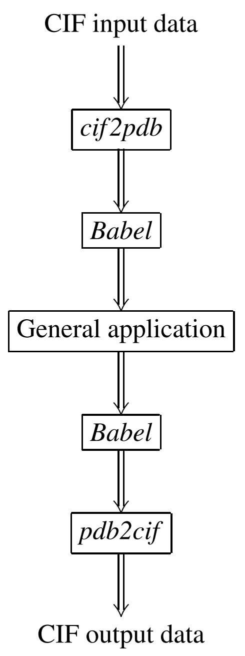

Fig. 5.1.3.2. Example of using filters to make a general application CIF-aware.

provides *CIFTr* by Zukang Feng and John Westbrook (http://sw-tools.pdb.org/apps/CIFTr/) to translate from the extended mmCIF format described in Appendix 3.6.2 to PDB format and *MAXIT* (http://sw-tools.pdb.org/apps/MAXIT/), a more general package that includes conversion capabilities. See also Chapter 5.5 for an extended discussion of the handling of mmCIF in the PDB software environment.

5.1.3.2. Using existing CIF libraries and APIs

Another approach to making an existing application CIF-aware or to design a new CIF-aware application is to make use of one (or more) of the existing CIF libraries and application programming interfaces (APIs). Because the data involved need not be reprocessed, code that uses a library directly is often faster than equivalent code working with filter programs. The code within an application can be tuned to the internal data structures and coding conventions of the application.

The approach to internal design depends on the language, data structures and operating environment of the application. A few years ago, the precise details of language version and operating system would have been major stumbling blocks to conversion. Today, however, almost every platform supports a variation

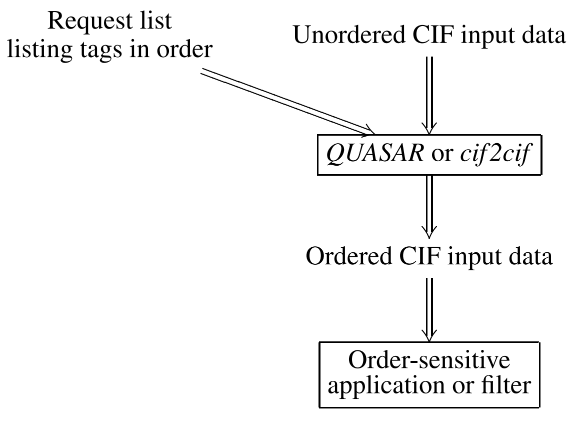
Request list listing tags in order
Unordered CIF input data
QUASAR or cif2cif
Ordered CIF input data
Order-sensitive application or filter

Fig. 5.1.3.3. Using *QUASAR* or *cif2cif* to reorder CIF data for an order-dependent application or filter.

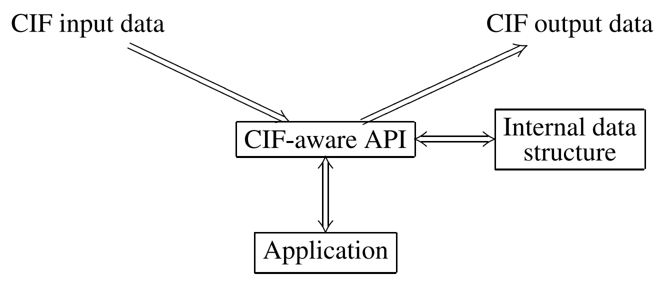

Fig. 5.1.3.4. Typical dataflow of a C-based CIF API.

of the Unix application programming interface and many languages have viable interfaces to C and/or C++. Therefore it is often feasible to consider use of C, C++ or Objective-C libraries, even for Fortran applications. *Star_Base* (Spadaccini & Hall, 1994; Chapter 5.2) is a program for extracting data from STAR Files. It is written in ANSI C and includes the code needed to parse a STAR File. *OOSTAR* (Chang & Bourne, 1998; Chapter 5.2) is an Objective-C package that includes another parser for STAR Files (http://www.sdsc.edu/pb/cif/OOSTAR.html). *CIFLIB* (Westbrook *et al.*, 1997) provides a CIF-specific API. *CIFPARSE* (Tosic & Westbrook, 1998) is another C-based library for CIF. *CBFlib* (Chapter 5.6) is an ANSI C API for both CIF and CBF/imgCIF files. The *CifSieve* package (Hester & Okamura, 1998) provides specialized code generation for retrieval of particular data items in either C or Fortran (see Chapter 5.3 for more details). The package *cciflib* (Keller, 1996) (http://www.ccp4.ac.uk/dist/html/mmcifformat.html) is used by the *CCP4* program suite to support mmCIF in both C and Fortran applications. If an application in Fortran is to be converted with a purely Fortran-based library, the package *CIFtbx* (Hall, 1993; Hall & Bernstein, 1996) is a solution. See Chapter 5.4 for more details.

The common interface provided in C-based applications is for the library to buffer the entire CIF file into an internal data structure (usually a tree), essentially creating a memory-resident database (see Fig. 5.1.3.4). This preload greatly reduces any demands on the application to deal with the order-independence of CIF, at the expense of what can be a very high demand for memory. The problem of excessive memory demand is dealt with in *CBFlib* by keeping large text fields on disk, with only pointers to them in memory. In some libraries, validation of tags against dictionaries is handled by the API. In others it is the responsibility of the application programmer. While the former approach helps to catch errors early, the second, ‘lightweight’ approach is more popular when fast performance is required.

The most commonly used versions of Fortran do not include dynamic memory management. In order to preload an arbitrary CIF, one needs to use one of the C-based libraries. Alternatively, a pure Fortran application can transfer CIFs being read to a disk-based random access file. *CIFtbx* does this each time it opens a CIF. The user never works directly with the original CIF data set. This provides a clean and simple interface for reading, but slows all read access to CIFs. In Fortran, compromises are often necessary, with critical tables handled in memory rather than on disk, but this may force changes in dimensions and then recompilation when dictionaries or data sets become larger than anticipated.

5.1.3.3. Creating a CIF-aware application from scratch

The primary disadvantage of using an existing CIF library or API in building an application is that there can be a loss of performance or a demand for more resources than may be needed. The common practice followed by most libraries of building and

references