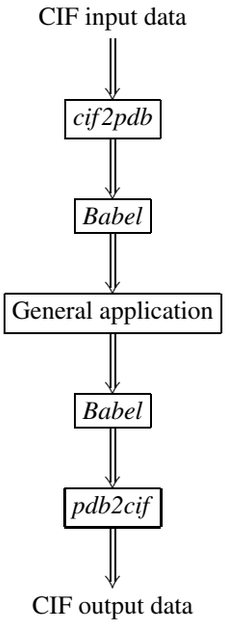

Fig. 5.1.3.2. Example of using filters to make a general application CIF-aware.

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

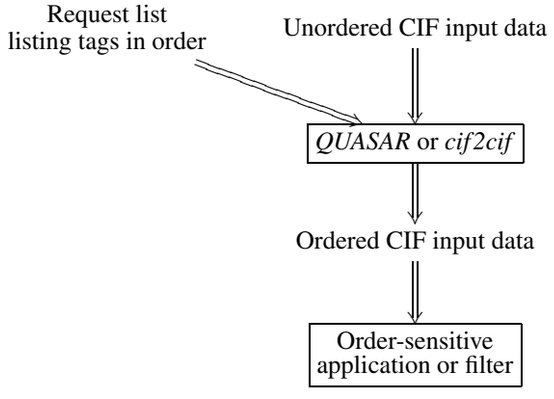

Fig. 5.1.3.3. Using *QUASAR* or *cif2cif* to reorder CIF data for an order-dependent application or filter.

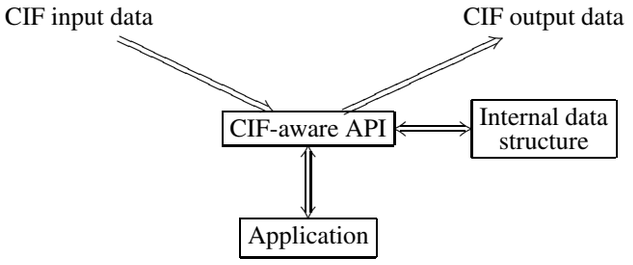

Fig. 5.1.3.4. Typical dataflow of a C-based CIF API.

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

Copyright © 2006 International Union of Crystallography