that now face software developers working with CIF: conformance to agreed formats *versus* deviations from standards to improve performance, as well as cross-platform portability.

The Cambridge Crystallographic Data Centre was established in 1965 'to compile a database containing comprehensive information on small-molecule crystal structures, *i.e.* organics and metallo-organic compounds containing up to 500 non-H atoms, the structures of which had been determined by X-ray or neutron diffraction' (Allen, 2002). The Protein Data Bank was established at Brookhaven National Laboratory in 1971 as an archive of macromolecular structural information. The NIH/EPA Chemical Information System was established in 1975 as a confederation of databases including mass spectroscopy, NMR and the data from the CSD. The three resources, CSD, PDB and CIS, took different approaches to applications development. The CSD was an integrated software system centred on a database. Both the software and the database were distributed on magnetic tape for users to use on their local computers. The developers of the software had to be concerned with portability of the software across the multiple computer systems used by crystallographers, but retained control of the design of the retrieval software and a core suite of applications. The PDB was an archive, rather than a database. Some software and the data were distributed on magnetic tape, but the application development model was what would now be called 'open', with users and software developers taking the data and the PDB format specification and creating software that would do useful things with PDB entries. The CIS was a remotely accessed confederation of databases on a central computer. The developers of software for the CIS did not have to be concerned with cross-platform portability, or with changes in syntax or semantics of data files impacting on external software developers. Developers of software for the CSD and the PDB had to be concerned with strict compliance with the rules for the respective data formats, albeit on somewhat different timescales. Developers of software for the centralized CIS database could negotiate for immediate changes in the data format to improve performance of the relevant application.

The CSD had agreed internal formats (Cambridge Structural Database, 1978). However, as noted in Chapter 1.1, there were many different formats in use for small-molecule crystallography and related fields. One may conjecture that one of many causes for such divergence was the CCDC practice of acquiring much of its data from journals, after differences among data formats had been masked by the publication process. The transition from this Tower of Babel to CIF is described in Chapter 1.1, and that history will not be repeated here, but it is important to note that an application writer working in the domain of small-molecule crystallography still has to be aware of a wide variety of formats in addition to CIF.

In the beginning, the PDB went through a relatively rapid format change and then achieved a stable format for more than two decades. The PDB differed from the CSD in depending on user deposition of data prior to publication. The better a user conformed to PDB data-format conventions, the more efficiently could the data move from deposition to release. The initial standard PDB format (PDB, 1974) was derived from the format used in a popular refinement program of the day (Diamond, 1971) and used 132-character records identified by the character strings in the first six columns. Starting in 1976, the PDB spent more than a year (PDB, 1976*a*,*b*, 1977) converting to an 80-column format, extensions of which are still in use to this day. Many external programs were developed using this 80-column format and it has become a major *de facto* standard for macromolecular software applications. Most application packages producing crystallographic macromolecular structures made a gradual transition from having output options for producing 'Diamond format' to having output options for producing PDB format. Macromolecular applications working with other disciplines shared the small-molecule applications penchant for multiple formats.

The CIS, working in a completely closed, central service environment, had little direct impact on the formats to be used for applications. The CIS would acquire data from existing archives and databases and meld them into its master database. It would deliver its data as text on a CRT. Much of the impact of CIS data formats was to be restricted to its own internal application development.

Most of the formats resulting from these early efforts were fixed-field, fixed-order formats. The result was that adapting an application to a data format was simple if the processing flow of the application conformed to the fixed order of the data format. Frequently, the data flow did conform. When the processing flow did not conform, it was necessary to create internal data structures or temporary files to allow the unfortunately timed arrival of data to be time-shifted until it was needed. In general, the heaviest burden was imposed on applications that needed to write data conforming to one of the agreed formats. As the complexity of such time-shifting processes increased, it became clear that the cleanest solution was to base an application on an internal database and to populate the database as the data were processed. When data were to be written by an application, the data could be extracted from the database in whatever order was required.

In the 1970s and early 1980s, such a procedure was a serious burden to place on an application. With limited memory and processor speeds, there was a strong argument for adapting agreed formats to the 'natural' processing flow, reducing or avoiding the need for an internal database. As the speed and size of computers have changed and as programming language and operating-system support for dynamic allocation of resources has improved, the need to have agreed formats driven by applications has become less pressing.

We need to understand three major thrusts in data representation: the development of markup languages, of data-representation frameworks and of database application support. Modern applications can benefit from all three.

5.1.2.1. Markup languages

A markup language allows the raw text of a document to be annotated with interleaved 'markup' specifying layout information for the bracketed text. For document processing, the implicit assumption of the use of an internal database became formalized with the gradual adoption of agreed markup languages in the late 1980s and early 1990s [*e.g.* TEX (Knuth, 1986), SGML (ISO, 1986), RTF (Andrews, 1987), HTML (Berners-Lee, 1989)]. When used in this manner, such a language has the implicit ordering assumption of reading forward in the document. However, with modern demands for multidimensional layout and document reflow, applications managing such documents achieve the best performance and flexibility when they store the entire marked-up document in an internal data structure that allows random access to all the information.

5.1.2.2. Data-representation frameworks

A data-representation framework provides the concepts for managing data and data about the management of data ('metadata'). Such frameworks may be based on programming languages or markup languages or built from scratch. They provide a mechanism for representing data (*e.g.* as data sets, graphs or trees)