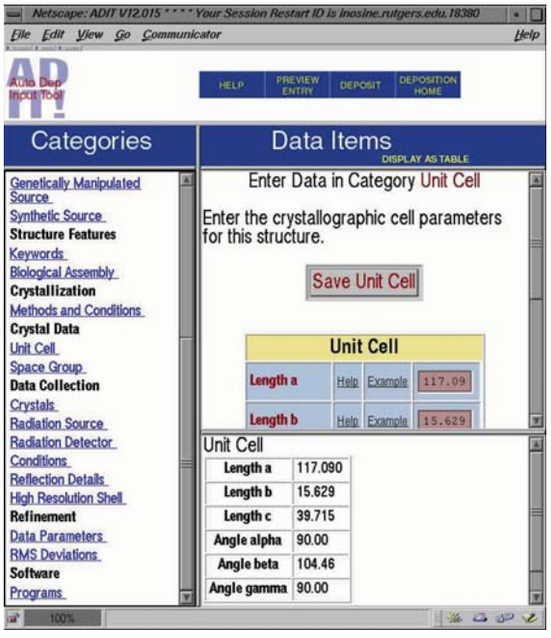

Fig. 5.5.3.3. Example *ADIT* data-input screen.

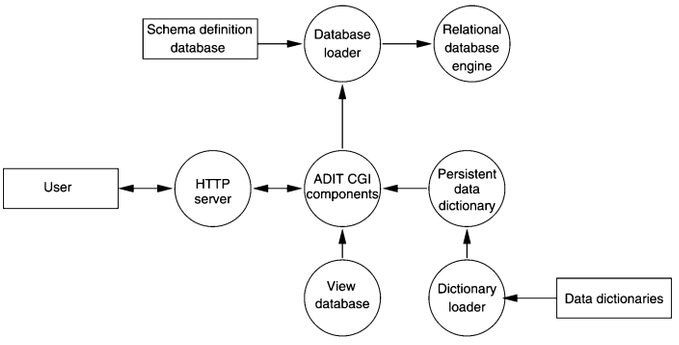

Fig. 5.5.3.4. Schematic diagram of *ADIT* database loading functions.

presentation details that are used in building the HTML input forms. An important use of the data view is to provide a simple and intuitive presentation of information for novice users which disguises the complex details of a data dictionary.

Fig. 5.5.3.3 shows an example *ADIT* editing screen for the crystallographic unit cell. The data dictionary category containing this information is named CELL, and the length of the first cell axis is defined in the dictionary as `_cell.length_a` (Fig. 5.5.2.2*b*). In this case, the data view has substituted *Unit Cell* and *Length a* for the dictionary data names. Although this example is simple, some dictionary data names are as long as 75 characters, and in these instances the ability to display a simpler name is essential.

Precise dictionary definitions and examples obtained from the data dictionary are accessible from the *ADIT* interface through buttons next to each data item. *ADIT* makes full use of the dictionary specification in data-input operations. Data items defined to assume only specific values have pulldown menus or selection boxes. Data type and range restrictions are checked when data are input and diagnostics are displayed to the user if errors are detected.

For performance reasons, the data dictionary is converted from its tabular text structure to an object representation using *CIFOBJ*. The class supporting the object representation provides efficient access functions to all of the data dictionary attributes. A dictionary loader is used to check the consistency of the data dictionary and to load the object representation from the text form of the data dictionary.

Any dictionary that complies with the dictionary description language (DDL2) can be loaded and used by *ADIT*. All *ADIT* software components gain their knowledge of the input data from the data dictionary and any associated data views. Consequently, *ADIT* can be tailored for use in virtually any data-input and data-processing application.

5.5.3.2. Generalized database support

In addition to the data editing and processing functions, *ADIT* also supports a versatile database loader (*mmCIF Loader*; http://sw-tools.pdb.org/apps/MMCIF-LOADER) that builds database schemata and extracts the processed data required to load database instances. The relation of the database loader to the central components of the *ADIT* system is shown in Fig. 5.5.3.4.

Schemata are defined in a metadata repository that is accessed by the loader application. In the simplest case, a schema can be constructed that is modelled directly from the data dictionary. Since the data model underlying the dictionary description language used to build *ADIT* data dictionaries is essentially relational, mapping a data dictionary specification to a relational schema is straightforward.

In other cases, a mapping is required between the target schema and the data dictionary specification. This mapping is encoded in the schema metadata repository. The database loader uses this mapping information to extract items from data files and translate these data into a form that can be loaded into the target database schema. The definition of the mapping operation can include: selection operations with equijoin constraints (*e.g.* the value of `_entity.type` where `_entity.id` = 1), aggregation (*e.g.* count, sum, average), collapse (*e.g.* vector to string), type conversions and existence tests.

Schema definitions are converted by the database loader into SQL instructions that create the defined tables and indices. Loadable data are produced either as SQL insert/update instructions or in the more efficient table copy formats used by popular database engines (*i.e.* DB2, Sybase, Oracle and MySQL). Loadable data can also be produced in XML.

5.5.3.3. Building a structure-determination data pipeline

One goal of high-throughput structural genomics is the automatic capture of all the details of each step in the process of structure determination. Fig. 5.5.3.5 shows a simplified structure-determination data pipeline. The essential details of each pipeline step are extracted and later assembled to make a data file for PDB deposition. The RCSB PDB data-processing infrastructure has been developed in anticipation of a data pipeline in which automated deposition would be the terminal step. The dictionary technology and software tools developed by the RCSB PDB to process and manage mmCIF data can be reused to provide the data-handling operations required to build the pipeline.

Dictionary definitions have been carefully developed to describe the details of each step in the structure-determination pipeline. These data items are typically accessible in electronic form after each program step. The information is either exported directly in mmCIF format or is printed in a program output file. To deal with the latter case, a utility program, *PDB_EXTRACT* (http://sw-tools.pdb.org/apps/PDB_EXTRACT), has been developed to parse program output files and extract key data values. In either case, the results of this incremental extraction of data from each program step must be merged to build a complete mmCIF