Overview of ISO STEP

By Chuck Eastman

In the 1980s, several of the international standards bodies came to the conclusion that the (then) current generation of translation methods, based on files formats—such as DXF and IGES—were not capable of supporting complex data models of engineering products. These organizations included American National Standards Institute (ANSI), ISO and other national standards groups in Europe. The US and European efforts began in parallel, but soon merged. They adopted the following now relevant goals:

- incorporate new programming language concepts, especially those dealing with object-oriented programming;
- incorporate formal specifications of the structures defined, independent from the implementation, using the new data-modeling languages;
- separate the data model from the physical file format and support mapping into multiple implementation formats, including files and databases;
- support subsets of an overall model, so that clusters of applications could be integrated without the overhead of having to deal with parts of a model irrelevant to a task;
- incorporate reference models that are common shared subsets of larger standard models.

These goals led to the development of a new technology for developing data exchange formats, generally described in Figure 1.

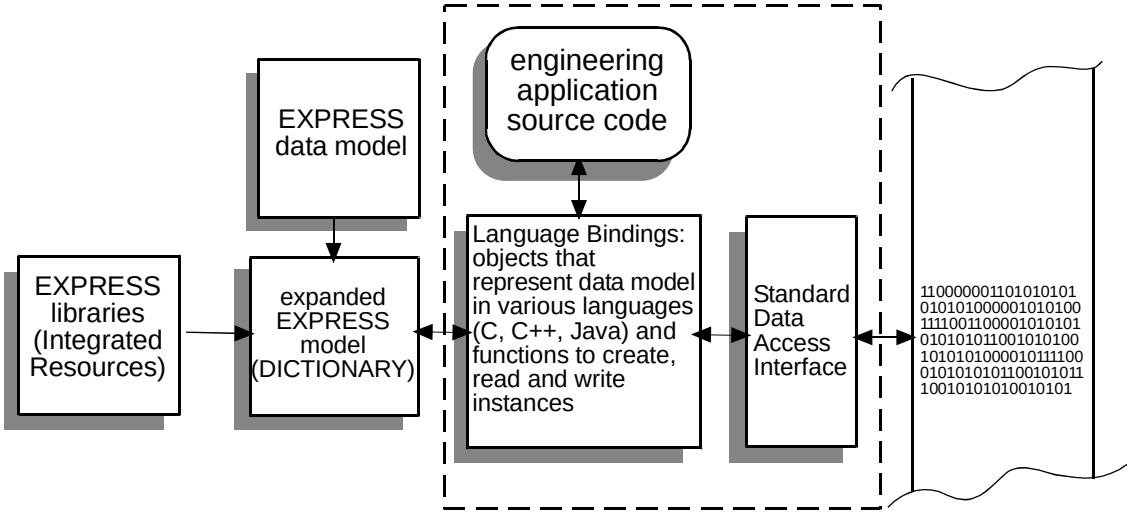

Figure 1: Overview of the ISO-STEP data interchange development model. It indicates how an exchange interface can be implemented.

A new language for defining implementation-independent data models was developed by Douglas Schenck, called EXPRESS. A graphical form of the language was also defined, called EXPRESS-G. EXPRESS was used to define an exchange model in some target domain, such as steel structures, building geometry, piping systems, and so forth. Separately, many common Entities (EXPRESS uses the term Entity to refer to object classes) dealing with geometry, materials, measurements, addresses, money, and other common attributes that are repeatedly used across multiple data models were defined as libraries early and called STEP Integrated Resources. By sharing these common aspects, later integration of any one model with others should be simplified.

The major advantage resulting from this new architecture was the commercial development of a number of software toolkits, that facilitated the implementation of interfaces. Among the facilities these toolkits provided are:

- Parsing and verifying the completeness and syntactic correctness of an EXPRESS model
- Mapping an EXPRESS-G diagram to the textual EXPRESS language or the reverse, generating diagrams of a textual EXPRESS model
- Defining equivalent C, C++ or Java objects for an EXPRESS model
- Facilities to instantiate, update, or read data carried in the language-implemented object model
- Facilities to read/write the object instances from/to a file
- Facilities to define database object schemas corresponding to a data model defined in EXPRESS
- Facilities to read/write object instances from/to a database

Different EXPRESS toolkits provide different subsets of the above functionality.

In order to implement an EXPRESS-based exchange capability for an engineering application, one needs the following:
The data model to be used (such is CIS/2), defined in EXPRESS
A toolkit with the language bindings corresponding to implementation language of the application receiving the interfaces

An EXPRESS-based exchange takes place as shown in Figure Two.

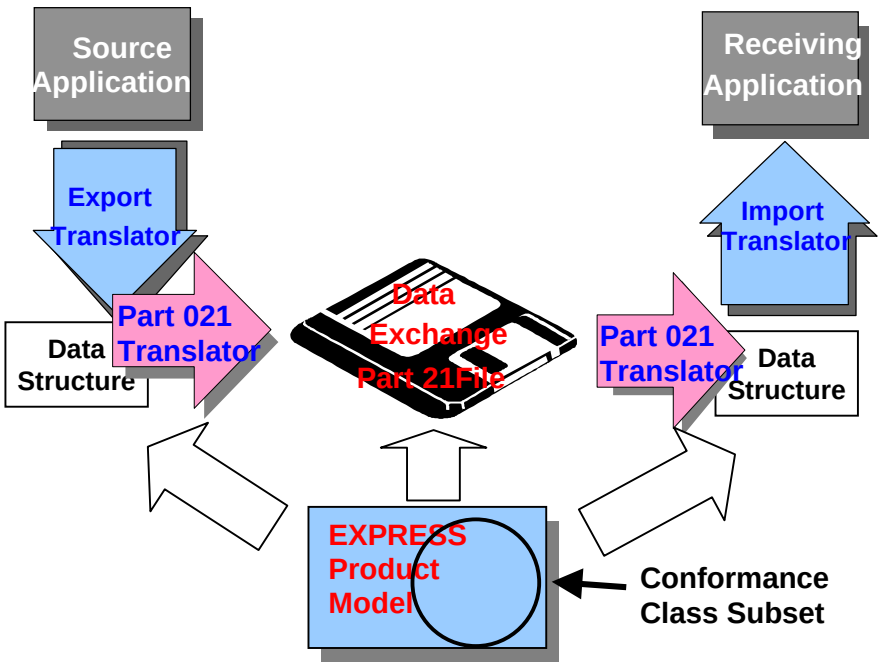

Figure Two: The components of a simple data exchange between two applications.

At the implementation level, each of the exchanges between two applications takes a form defined by the EXPRESS product model, shown at the bottom of Figure Two above. Most applications will only use a small part of an overall EXPRESS exchange model. Subsets of an exchange model are called a Conformance Class. The Conformance class subset of the EXPRESS model defines the data structures that should be read into and written from in the translating

applications. The STEP toolkits also implement a standard file format for storing the data, allowing reading and writing to/from the file. While the EXPRESS toolkits help in automating the in-memory definition of an EXPRESS model, to create object instances and to read and write to a Part 021 file (or other storage mechanism), the export and import translator code to and from the data structures of the application must be custom written.