Fundamentals of STEP Implementation

David Loffredo
loffredo@steptools.com
STEP Tools, Inc., Rensselaer Technology Park, Troy, New York 12180

A) Introduction

The STEP standard documents contain such a large amount of extremely detailed information that it can be difficult for newcomers to sort it all out. This paper presents an overview of the software development process for someone who is interested in a project that makes use of STEP to exchange product model data. This paper is targeted at technically knowledgeable software developers who are not necessarily familiar with STEP or the issues involved in STEP implementation.

B) Structure Of STEP

The STEP standard is divided into many parts. These parts cover topics such as methods used to present the standard, implementation architectures, conformance testing procedures, resource information models, and application protocols. The STEP parts can be divided into Description Methods, Information Models, Application Protocols, Implementation Methods, and Conformance Tools.

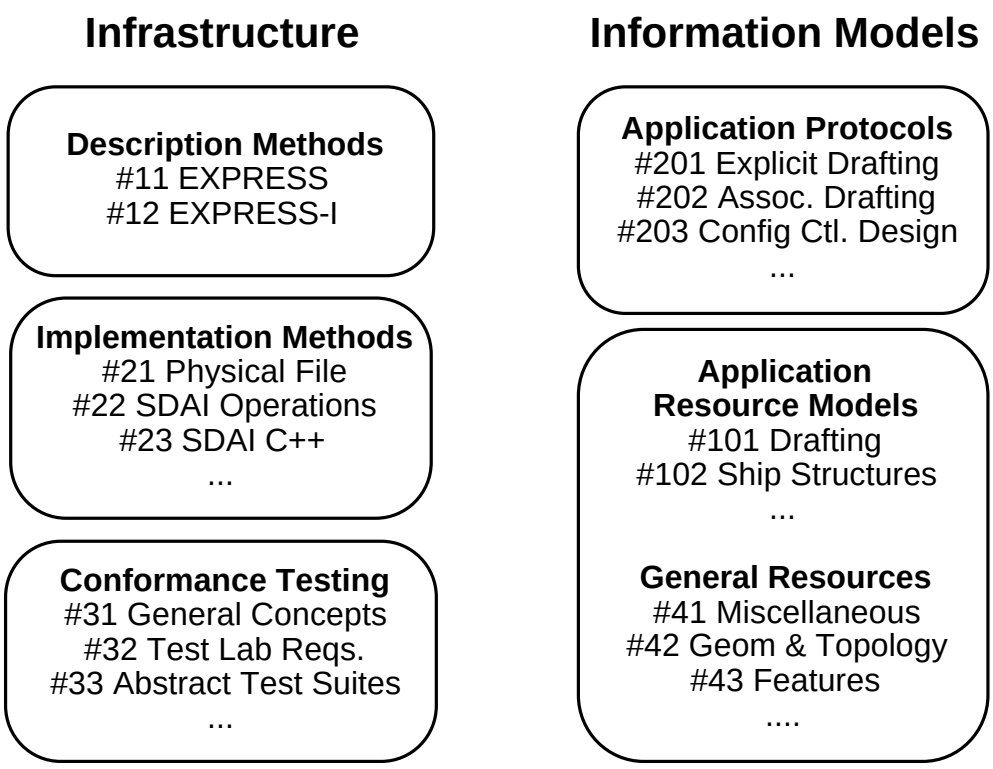

Figure One — High Level Structure of STEP

One of the most important aspects of STEP is its extensibility. This extensibility is the result of the decision to base STEP on an information modeling language. The EXPRESS language is the fundamental Description method of STEP. Future enhancements to STEP may introduce other description methods, but for right now, EXPRESS is the fundamental tool used to describe the information models and application protocols that are the bulk of the standard.

The Information Models and Application Protocols describe the data structures and constraints of a complete product model. Each application protocol combines one or more information models and places additional constraints on those models. For example, the application protocol for 2D drafting combines parts 42 and 46 and restricts part 42 so that it only describes two-dimensional data.

The Implementation Methods are protocols that are driven by the EXPRESS language. They are used to move real EXPRESS-defined application data between tools, and to make that data available to application developers. The first implementation method` is the STEP physical exchange file, often referred to as Part 21 file format. The Part 21 specification is the medium that STEP provides to move EXPRESS-defined data between databases and CAD systems.

The Standard Data Access Interface (SDAI) is another implementation method that

software developers can use to manipulate data defined by EXPRESS. The specification defines programming language bindings for C, C++, and Fortran.

Some of the STEP parts are listed below:

Part	Document Title
1	Overview & Fundamental Principles
11	EXPRESS Language
12	EXPRESS-I Language Reference Manual
21	Clear Text Encoding of the Exchange Structure
22	Standard Data Access Interface
23	SDAI C++ Language Binding
24	SDAI C Language Binding
25	SDAI FORTRAN Language Binding
31	Conformance Testing - General Concepts
32	Conformance Testing - Test Lab Requirements
33	Conformance Testing - Abstract Test Suites
34	Conformance Testing - Abstract Test Methods
41	Product Description and Support
42	Geometric and Topological Representation
43	Representation Structures
44	Product Structure Configuration
45	Materials
46	Visual Presentation
47	Shape Tolerances
48	Form Features
49	Process Structure and Properties
101	Draughting Resources
102	Ship Structures
103	Electrical/Electronics Connectivity
104	Finite Element Analysis
105	Kinematics
201	Explicit Draughting
202	Associative Draughting
203	Configuration Controlled Design
204	Mechanical Design Using Boundary Representation
205	Mechanical Design Using Surface Representation
206	Mechanical Design Using Wireframe Representation
207	Sheet Metal Dies and Blocks
208	Life Cycle Product Change Process
209	Design Through Analysis of Composite and Metallic Structures
210	Electronic Printed Circuit Assembly, Design and Manufacturing
211	Electronics Test Diagnostics and Remanufacture
212	Electrotechnical Plants

213.....Numerical Control Process Plans for Machined Parts
214.....Core Data for Automotive Mechanical Design Processes
215.....Ship Arrangement
216.....Ship Moulded Forms
217.....Ship Piping
218.....Ship Structures
219.....Dimensional Inspection Process Planning for CMMs
220.....Printed Circuit Assembly Manufacturing Planning
221.....Functional Data and Schematic Representation for Process Plans
222.....Design Engineering to Manufacturing for Composite Structures
223.....Exchange of Design and Manufacturing DPD for Composites
224.....Mechanical Product Definition for Process Planning
225.....Structural Building Elements Using Explicit Shape Rep
226.....Shipbuilding Mechanical Systems

The PDES/STEP standard and EXPRESS are important to the design and manufacturing community for several reasons:

- The standard contains a **large body of definitions for engineering data**. The scope of the definitions include mechanical and electrical CAD, manufacturing processes, and specialty domains such as composite materials and shipbuilding.

- The EXPRESS language defines **constraints** as well as data structures. These constraints describe a correctness standard that must be met before an engineering data set is sent to any other application.

- The standard defines a logical description that is **technology independent**. Today many believe object-oriented database systems to be the best kind of database for engineering data. In the past, most would have advocated relational systems. In the future a new kind of system may be invented. Furthermore, the "best" system may vary from application to application.

- The standard allows engineering applications to exchange information about product and process designs.

C) What Does it Mean to Implement STEP?

The STEP standard covers the exchange of product model data. A STEP implementation is an application that uses this standard to exchange product information, or makes it possible for other applications to do so. Thus, STEP implementations cover the range from CAD systems, bill of materials systems and so forth, to stand-alone translators, to packages that make it possible to develop the above systems.

The STEP standard categorizes the various types of product data around Application

Protocols (AP). Each AP is a formal document describing the activities in the lifecycle of a product, called the Application Activity Model (AAM); the pieces of product information that are needed for the activities, called the Application Reference Model (ARM); and finally, a formal EXPRESS information model that captures everything in the ARM and ties it to a library of pre-existing definitions. This is referred to as the Application Interpreted Model (AIM). In addition, STEP defines collections of common definitions that can be shared between Application Protocols. These are called Application Integrated Constructs (AICs) and may be of interest if you are planning to use data defined by several APs in your application.

In general, translators and interfaces to specific engineering systems will support for specific application protocols, while more general tool kit packages can usually be tailored to any AP.

The AP defines the collection of information required as a base exchange unit. Often, in existing environments, the product data described by an AP is distributed between a number of different engineering systems. In such a situation, it may be necessary to build interfaces to several systems, as well as tools to assemble the resultant partial data into one complete exchange unit.

In addition to the EXPRESS language and the Application Protocols, STEP Part 21 defines an EXPRESS-driven exchange file format. Any implementation should be able to read and write these STEP exchange files. In addition, it may be desirable to use the APs as the definition for a shared database.

The first question to answer when planning a STEP implementation is what application protocol will be used. When this is clear, you should get a copy of the AP document from the appropriate sources and examine it closely.

The AP documents can be quite long, but most of this is the Application Interpreted Model. You will become familiar with this eventually, but initially you should begin by familiarizing yourself with the Application Reference Model. The ARM describes the basic Application Objects that an implementor should be concerned with. In familiarizing yourself with the ARM, you should try to understand what each application object represents and consider where the information contained in it will come from or go to.

The application objects are divided into subject areas called units of functionality. These provide a logically complete subset of information about some particular aspect of the product. For example, the Design Activity Control UOF in AP 203 is concerned with tracking modifications to a product over it's lifetime. The application objects in this UOF are: Change Order, Change Request, Start Order, Start Request, Work Order, and Work Request.

Looking at AP 203, we see that it contains about 36 application objects, distributed among nine units of functionality. The UOFs are Authorization, Bill Of Material, Design

Information, Effectivity, End Item Identification, Part Identification, Shape, and Source Control.

Once you have a handle on the ARM and the Application Objects, it is time to consider the AIM. The AIM is considerably more involved than the ARM, but it represents the same basic information. The AIM is an EXPRESS information model that formally describes the application objects in terms of a library of pre-existing definitions, called the generic resources or integrated resources. This highly normalized representation contains the structures as well as the constraints that those structures must obey. The AIM is used as the basis for the implementation and data exchange.

An easier way to become familiar with the AIM is to look at the EXPRESS-G diagrams in the AP document. EXPRESS-G is a formal diagrammatic form for the EXPRESS language. EXPRESS-G diagrams contain the data structures, inheritance relationships, attributes and relationships between structures in the EXPRESS information model. The EXPRESS-G diagrams are very good at conveying the data structures associated with an information model, but they do not contain the rules and constraints that apply to the structures.

At this stage you should have chosen the AP for your implementation. You will have reviewed and understand the ARM of the AP, and will have determined where the appropriate information will come from or go to in your system(s). Furthermore, you should have reviewed the AIM, and determined how this maps to the ARM objects that you are interested in. Now that you have an understanding of the application protocol and the issues involved, it is time to move on to the software system design and implementation.

D) Software System Design

STEP implementations fall into several categories. Translators take data from pre-existing systems and convert it into STEP AP defined data. The tool converts non-STEP data into STEP data. Other applications might take STEP data as input, and then perform some function on it, generating more STEP output. An example of such a thing would be an application that takes partial AP information from several sources, like geometry from a CAD system and configuration information from a CM system, and then merges them into a complete AP-203 exchange file.

Another category might be an application that takes specific AP data and performs some analysis on it, such as a finite element package or a geometry visualizer.

These applications should all work from STEP exchange files and possibly a shared database as well, so it is important to consider how the application will be tied to the EXPRESS information models of the various APs that they will work with. A number of techniques have been used successfully on previous implementations.

Each technique has characteristics that make it useful for a certain class of problems, and implications that affect where to allocate your resources. The sections below describe common STEP development tools along with several of the more common implementation architectures and the types of problems that they work best on.

Other documents exist describe the overall architectures for developing STEP systems from scratch (PTI017 -- "An Architecture for Implementing STEP"??)

E) STEP Development Tools

The task of producing a STEP implementation can be considerably simplified by making use of pre-existing STEP development tools. A number of STEP development tools and environments are available, each with their own special features. This section describes some of the tools and services you may wish to take advantage of.

It is possible to construct an implementation from scratch. In general, this requires a significant outlay of time and effort developing support software before you even begin to address your original tasks. This approach may not be the most efficient use of resources because of the amount of work involved in setting up a physical file reader/writer, creating all of the classes by hand, creating all the necessary routines to handle traversal, memory management and so forth.

A development environment should provide libraries of functions to create, destroy, access and update EXPRESS-defined data. These libraries may take the form of one of the SDAI bindings, or some other binding, but all should provide I/O from either physical files or database systems. In addition, query and traversal functions, memory management, and other CASE-oriented services may be provided.

These packages are often written around a data-dictionary, so that they can handle structures defined by any EXPRESS schema. Libraries of functions for specific information models might also be provided, such as translations and transformations, facetting, solid modeling packages and so forth.

Since the EXPRESS language is meant to be computer sensible, a compiler is often provided to transform the EXPRESS into a variety of useful forms, such as C++ classes or SQL database definitions.

Other tools may browse STEP physical files, help with constraint validation, provide assistance in EXPRESS model development, EXPRESS-G diagram development, translations, geometry visualization.

F) EXPRESS Early-bound Applications

Once you have assembled the tools you plan to use, you can begin application development. In general, the implementation techniques can be classified into two groups, early and late binding, depending on how the original EXPRESS information model is made available to the software environment.

An early binding system makes the EXPRESS information model available as specific programming language data structures for each different definition in the EXPRESS model. For example, an early binding such as the SDAI C++ would contain specific C++ classes for each definition in the AP-203 AIM. One major advantage to this approach is that the compiler can do extensive type checking on your application, detecting conflicts at compile time. Special semantics or operations can also be captured as operations tied to a particular data structure.

Most early-bindings to date use C++ as the target language. C++ works well in an early binding because it has strong type checking, supports inheritance, and allows methods to be attached to classes. Depending upon your choice of tools, a library of classes for your AP may already exist. If it does not, you will need to generate one.

Early bindings are usually produced by an EXPRESS compiler. The compiler will parse, resolve, and check the AP AIM model, then passes control to a code generator to produce data structures for that model. EXPRESS entity definitions are usually converted to C++ classes, type definitions are are converted to either classes or typedefs, and the EXPRESS inheritance structure is mapped onto the C++ classes. Each class should have access and update methods for the stored attributes, possibly access methods for simple derived attributes, and constructors to initialize new instances.

Below is a very simple EXPRESS entity definition that would be translated to a C++ class. The details of the C++ class vary from package to package, but all would provide access and update methods for the X and Y attributes:

```
ENTITY Point;
     x : REAL;
     y : REAL;
     END_ENTITY;
```

Once you have the classes, you can begin to write your application code. The binding should let you create instances of the classes, populate them, and write them out to a STEP physical file. It should also be able to read a file and present the contents to you as instances of your classes.

In the example C++ code below, we create a Point object and fill in some of its attributes:

```
/* Create a point using the default constructor
```

```
 *  and use the update methods to set its values. */
SdaiModelH mod;
PointH point1 = SdaiCreate(mod,Point);
point1->x (1.0);
point1->y (0.0);
```

Here the object is created using a special version of the "new" operator that handles persistent objects, and the attribute values are set using specialized member functions for each. As you continue development, you may find it useful to attach additional methods to the C++ classes to handle the operations unique to your problem domain.

G) EXPRESS Data-Dictionary Applications

Another way to make the EXPRESS information model available to a software environment is through a data dictionary. A late binding system uses an EXPRESS data dictionary to handle access to data values.

If you chose to use a late binding such as the SDAI C, you will not need to generate special data structures. Generally only one data structure is used for all of the definition in the EXPRESS model. You still may need to compile the EXPRESS to produce the data dictionary.

Data values are found by queries against the data dictionary. As you write your application code, you will most likely get and retrieve values using a few simple functions. In the example SDAI C code below, we create a Point object and fill in some of its attributes:

```
/* create new instances */
SdaiAppInstance point1;
point1 = sdaiCreateInstanceBN (myModel, "Point");
sdaiPutAttrBN (point1, "X", sdaiREAL, 1.0);
sdaiPutAttrBN (point1, "Y", sdaiREAL, 0.0);
```

Here the object is created by a function that takes the name of the type, and the attributes are set using a similar function that takes the attribute names. The important point is that all access is done through the names of the types and attributes, rather than by specialized functions for each.

In general, late bindings are most useful in programming environments that do not support strong type checking, and in software that works on data from a common subset of multiple EXPRESS schemas. Also the EXPRESS model stored in the dictionary can change somewhat without necessarily affecting your application.

The biggest advantage to this approach may be simplicity. An average AIM can contain upwards of 200 definitions, each of which may translate to one or more C++ classes in an

early binding. All of these classes must be generated, compiled and linked before producing your application. A data dictionary-driven binding requires less initial work and lends itself to faster prototyping.

The two disadvantages to late binding approaches are the lack of compile-time type checking and difficulty of programming. These are not major issues in prototypes or small applications, but will eventually surface in larger systems.

H) Other Approaches

The two approaches described above are not the only possibilities. A mixed binding approach provides the advantages of an early binding (compile-time type checking and and semantics as functions on a class) without the complexity introduced by an extreme number of classes.

A mixed binding takes advantage of the observation that applications rarely use all of the structures defined by an AP AIM. The subset of structures that are used, called the working set, can be early-bound, while the rest of the AP is late-bound. All data is still available, but the application development process is simplified. The number of classes and files that are needed are reduced dramatically, resulting in quicker compiles, simpler source control, and more rapid development.

Another, more labor-intensive, approach would be to hand-generate an early binding for a particular AP. Such a binding would not need to be based directly on the AIM EXPRESS model, but could be customized to provide a simplified view of the data. Any data created using this binding should be mapped into the underlying AIM entities when it is written out.

Although this approach might provide a simplified programming interface, there are some drawbacks to be aware of. Aside from the increased labor involved in defining and implementing the binding, this method requires that you understand the AP AIM completely, and be able to predict how it will be used. A specialized binding would also need to be documented as thoroughly as the AP AIM it replaces. Each AP document is an extremely detailed book -- the AP-203 spec is over 650 pages long. Reproducing that level of detail could be difficult.

I) Scaling Up

Initially, you should choose a very simple subset of your problem area for a prototype application. For example, with a CAD interface, you might try transferring one or two pieces of simple geometry. This will let you focus on the mechanics of application development, such as the target language, compiling, linking, and so forth.

Once a prototype is running, you should continue to expand the range of information handled. You will be more familiar with the bindings, and any initial set up problems will be behind you. You can concentrate on the semantic issues rather than the mechanical ones, and continue to expand your coverage of the problem area.

Some projects may need to move an implementation from shared files to a central transaction-controlled database. This process is not as straightforward as the development of a file-oriented system. Different databases are suited to different kinds of applications. For example, object oriented databases and file systems offer navigational access and the performance needed to implement CAD applications, while relational database contain offer query-based access to large repositories of information.

Although it is technology-neutral, EXPRESS allows for information models that challenge the capabilities of many existing database systems. Implementations may require heavy encoding to represent the full range of EXPRESS structures within a target database system. Tools are available from various vendors to help you do this. You should have a clear understanding of the features you are looking for in a database implementation and the tradeoffs they will imply.

J) Constraints and Correctness

The EXPRESS language has the ability to capture rules and constraints. These rules and constraints are the most formal representation of the original design intent of the AP developers, and can be used to check the data that you are producing.

You can try to apply these rules and constraints to your implementation by hand, either by looking at the data directly or writing code to evaluate the conditions set forth by the rules. Another way would be to use an automated EXPRESS tool that can interpret the AIM source and evaluate the rules and constraints on a data set.

At this stage of the project, you should begin exchanging data with test sites to detect any semantic problems and to gain confidence in your implementation. You may also want to search for a conformance suite for your chosen application protocol. As time goes on suites will appear for each of the application protocols. These conformance suites will emphasize the semantics associated with the application protocol as well as the syntax of the exchange files.

K) Summary

As we have seen, the first task in a STEP implementation is to determine the proper application protocol for your problem area. After familiarizing yourself with the correct application protocol, you should examine your project requirements and select tools that

will help you to meet those requirements. We have discussed some of the tools available, along with software implementation styles like early and late bindings.

Once the AP and support tools are in place, you should produce a simple prototype to gain experience, and troubleshoot the mechanics of compiling, linking and so forth. As your confidence grows, you should introduce more and more functionality. As your project develops, you should use the EXPRESS rules and constraints, exchange data with test sites and make use of conformance test suites to verify the correctness of your implementation.

At this point your implementation should be running and well along the way towards completion. The major elements of a STEP implementation should all be in place and only incremental improvements related to the problem domain should be needed. Time to start planning your next project!