The Application of ISO 10303-11 (the EXPRESS Language) in Defining Data Models for Software Design and Implementation.

Robert W. Schuler
April 2001

Abstract

International Standards Organization Technical Committee 184, Sub-committee 4, Working Group 3 (ISO TC184/SC4/WG3) is responsible for the development of ISO 10303 (informally known as the STandard for the Exchange of Product model data—STEP) and has a formalized methodology for capturing domain knowledge in the form of an Application Protocol that uses the EXPRESS language to define data requirements for the exchange of product model data between dissimilar automation systems. In a very real sense, an Application Protocol is a Software Requirements Specification that can be realized by compiling EXPRESS language specifications into object repositories that can be interfaced by additional software.

This paper provides an overview of the key EXPRESS constructs and shows how these key constructs are used within an ISO 10303 Application Protocol. This term paper was prepared for Computer Software Engineering course 648 (CSWE 648) " Software Design and Implementation" during the Spring Semester of 2001 at the University of Maryland University College (UMUC).

Introduction

As software systems continue to increase in complexity and scope, the need for more complex representations of real world objects also increases. Complex systems rely on complex data models to support them. Within the Engineering community, computers have become central to the design of complex systems and products like Aircraft, Ships, Automobiles, Process Plants, etc. In 1979 it became apparent that there was a need to exchange computer data describing complex systems between Computer Aided Design (CAD) tools running on disparate computer systems with different levels of sophistication and capability. The Initial Graphics Exchange Specification (IGES) was created to address this need and sanctioned by the American National Standards Institute (ANSI). (Kemmerer, 11)

In 1984, a second effort called the Product Data Exchange Specification (PDES) was launched to leverage the early lessons from IGES and to produce a more robust product data modeling framework. This effort soon became an international standards effort which in 1994 yielded an initial collection of documents known formally as “ISO 10303 Industrial automation system and integration – Product data representation and exchange” and informally as the STandard for the Exchange of Product model data (STEP). STEP follows a three-tier data modeling approach that captures knowledge in a model using

domain terminology, maps this data onto a common set of EXPRESS resources, and uses an independent representation tier to store/share information that conforms to the models in the first two tiers.

At the heart of this mapping effort is an Object-Oriented data modeling language, EXPRESS. The EXPRESS Language is formally defined in ISO 10303-11:1994(E) "Industrial automation systems and integration—Product data representation and exchange—Part 11: Description methods: The EXPRESS language reference manual". The two most common representations of EXPRESS data schemas are lexical and a graphical. The lexical form is stored in ASCII files similar to many programming languages. The graphical form is called EXPRESS-G and consists of semantically defined symbols comprised primarily of boxes, lines and small circular arrowheads. Tools are available to allow developers to convert from one form to the other enabling both forms to remain synchronized during a project.

Contents

This section of the paper provides a "road map" to the rest of the paper. This paper consists of the following sections:

- Abstract—Provides a brief overview of the paper's subject.
- Introduction—Provides some background and context useful for interpreting the paper.
- Contents—This section.
- EXPRESS—Defines the EXPRESS and EXPRESS-G languages in enough detail to understand the rest of the paper.
- STEP—Provides an overview of the layout and document numbers of the STEP standard. Also describes STEP's three tier modeling approach.
- Application Protocols—Shows the general contents of an Application Protocol (AP) and explains the process used by ISO to define them. Also highlights the parallels between AP development and canonical software development lifecycle models (SDLC).
- AP 227 Pipe Definition—Traces the definition/representation of a pipe in ISO 10303-227:2001(E) "Industrial automation systems and integration—Product data representation and exchange—Part 227: Application protocol: Plant spatial configuration" through all three tiers of STEP. This example shows the use of EXPRESS in its proper context within ISO 10303.
- Conclusions—Provides a summary of the paper and highlights some of the key ideas that can be concluded from the material presented here.
- References—Points to the reference material used in preparing this paper.

EXPRESS

Overview

ISO 10303-11:1994(E) "Industrial automation systems and integration—Product data representation and exchange—Part 11: Description methods: The EXPRESS language reference manual" "defines a language by which aspects of product data can be specified.

The language is called EXPRESS." (ISO 10303-11,1). The EXPRESS language reference manual defines two basic representations of the language with most of the emphasis being placed on defining the first as a lexical language. The second representation is a graphical form called EXPRESS-G and is defined in annex E of ISO 10303-11. This section of the paper provides a brief introduction to both representations of the EXPRESS language.

The lexical form of EXPRESS is defined using a derivative of Wirth Syntax Notation (WSN). (ISO 10303-11,7). The outermost syntactical element in an EXPRESS schema is declared using the `SCHEMA` keyword. The WSN for `SCHEMA` is defined on page 56 of the specification and is repeated here for reference:

```
281 schema_decl = SCHEMA schema_id ';' schema_body END_SCHEMA ';' .
280 schema_body = { interface_specification } [constant_decl ]
                  {declaration | rule_decl} .
228 interface_specification = reference_clause | use_clause .
189 declaration = entity_decl | function_decl | procedure_decl |
type_decl .
```

The complete annotated listing of WSN for the EXPRESS language is contained in annex A of ISO 10303-11. For the purposes of this paper, it is sufficient to understand that the fist line of the listing above requires that an EXPRESS schema start with a syntactical statement similar to:

```
SCHEMA plant_spatial_configuration;
```

and ends with a syntactical statement similar to:

```
END_SCHEMA; -- plant_spatial_configuration
```

In this example, the `schema_id` is "plant_spatial_configuration". Notice that both of the semicolons in the previous two lines are quoted literals on line 182 in the previous listing and that the key words `SCHEMA` and `END_SCHEMA` are also listed directly. The double dashed comment is also declared in WSN within ISO 10303 and is similar to the '#' symbol in UNIX shell scripts. That is, from (and including) the double dash to the next carriage return is a comment.

In addition to the `SCHEMA` keyword, we will also consider the following keywords in the next few sections of this paper: `USE FROM`, `TYPE`, `ENTITY`, `WHERE`, `RULE`, and `FUNCTION`. Except for `USE FROM` and `WHERE`, all of these keywords define a block structure within an EXPRESS file. The block starts with the keyword and ends with the keyword repeated with the prefix "`END_`". For example, `TYPE` and `END_TYPE` demark a block of an EXPRESS file that defines a data type.

The general skeleton of an EXPRESS Schema is listed below. The following sections of the paper present each syntactical structure in more detail.

```
SCHEMA schema_name;
USE FROM other_schema_name (entity_name, entity_name2, . . .);
```

```
o o o
TYPE my_type = REAL;
      WHERE
            Some_assertion;
END_TYPE;
o o o
ENTITY entity_name;
      Attribute_1 : attribute_type;
      WHERE
      Rule_id : some_assertion;
END_ENTITY;
o o o
RULE rule_name FOR (global_language_assignment);
  WHERE
   SIZEOF(global_language_assignment) <= 1;
 END_RULE;
o o o
FUNCTION function_name(variable_name: variable_type): return_type;
  Function_body;
END_FUNCTION;
END_SCHEMA;
```

SCHEMA

The `SCHEMA` keyword surrounds all other syntax in an EXPRESS file. Each schema has a name that is used to uniquely identify the schema being defined. It is common to refer to the EXPRESS file by the name of the schema regardless of the actual filename in a computer's operating system. Within STEP there are two general kinds of schemas: short-form and long-form. The distinction is based on the `USE FROM` keyword and is explained later.

EXAMPLE OF SCHEMA

```
SCHEMA plant_spatial_configuration;
o o o
END_SCHEMA;
```

EXPRESS-G REPRESENTATION OF SCHEMA

There is no graphical symbol for a schema. In a sense the physical page represents the schema because it collects all the EXPRESS-G symbols that define the schema.

USE FROM

One schema can copy all or part of another schema's contents in a manner similar to the "`#include`" compiler directive in the C language. `USE FROM` does not have a matching "`END_USE`" keyword. It is simply terminated at the next semi-colon.

In STEP, schemas that contain `USE FROM` statements are called short-form schemas. Typically, an Application Protocol contains a short-form schema that uses syntax from the Generic Integrated Resource parts of the STEP standard (Generic Integrated Resources are explained later in this paper). Each Application Protocol also has an official long-form schema in which all of the `USE FROM`s have been resolved. In other words, a long-form schema has copied the necessary syntax directly into itself and is a complete and self-contained definition.

EXAMPLE OF USE FROM

```
USE FROM MYSCHEMA (ENTITY_3 AS myEntity_3);
```

This listing says that the current schema has an `ENTITY` named “myEntity_3” and that the definition for this entity is to be copied from (used from) the definition of the `ENTITY` named “ENTITY_3” in the `SCHEMA` named “MYSCHEMA”.

EXPRESS-G REPRESENTATION OF USE FROM

As shown in Figure 1 below, the `USE FROM` is represented graphically as a box with an oval inside. The text in the oval indicates the name of the schema being used and the name of the entity being copied. The text below the oval indicates the alias for the used entity in the current schema. The EXPRESS-G in Figure 1 says the same thing as the previous EXPRESS listing.

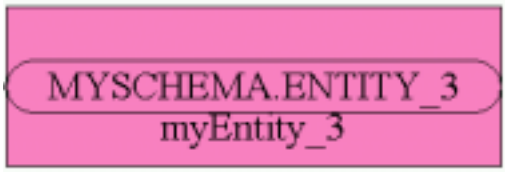

Figure 1 EXPRESS-G for USE FROM

TYPE

One of the more powerful aspects of the EXPRESS language is the flexibility of user-defined types. In addition to the “built-in” types of: String, Number, Integer, Real, Boolean, Logical, and Binary, a data modeler can define data types that add semantic meaning to attributes within a schema.

For example, a data modeler can define a type “name” to hold a string value. This allows the data model to impose constraints on the strings that are names. Simple constraints can be imposed using the keyword `WHERE` as explained later in the paper.

In addition to types that resolve to individual literals, EXPRESS also allows data modelers to define enumerations and a special data type known as a “SELECT TYPE”. A SELECT TYPE defines a compound data type that may assume any legal value of any of its constituents.

EXAMPLE OF TYPE

```
TYPE DEFINED_1 = STRING;
  END_TYPE;

  TYPE DEFINED_2 = INTEGER;
  END_TYPE;

  TYPE SELECT_1 = SELECT
    (DEFINED_1,
     DEFINED_2,
     ENUMERATION_1);
  END_TYPE;
```

```
TYPE ENUMERATION_1 = ENUMERATION OF
  (ONE,
   TWO,
   THREE);
END_TYPE;
```

This listing shows four user-defined data types named “DEFINED_1”, “DEFINED_2”, “SELECT_1”, and “ENUMERATION_1” respectively. DEFINED_1 and DEFINED_2 are simple types in that they simply add semantics to a string and an integer. In this case the added semantics are rather abstract. ENUMERATION_1 is an enumeration data type. As shown in the listing, the legal values for ENUMERATION_1 are: “ONE”, “TWO”, or “THREE”. SELECT_1 is a select type and defines a data type whose legal values include the legal values of any of the other three user-defined data types described so far.

EXPRESS-G REPRESENTATION OF TYPE

Figure 2 below shows the EXPRESS-G representation of the previous EXPRESS listing. In all cases, `TYPE`s are represented as boxes with text inside. Native EXPRESS types (e.g. `STRING` and `INTEGER`) are represented as solid-lined boxes with a solid line on the right side. User-defined types are represented as dash-lined boxes. Select types (e.g. SELECT_1) are represented by dash-lined boxes with a dashed line on the left side of the box. Enumerations (e.g. ENUMERATION_1) are represented as dash-lined boxes with a dashed line on the right side. The values allowed for the enumeration are not shown in EXPRESS-G.

Lines are used to show relationships between the boxes. The circle on the end of the line can be thought of as an arrowhead. For example, DEFINED_1 is a STRING because there is a line pointing from DEFINED_1 to STRING.

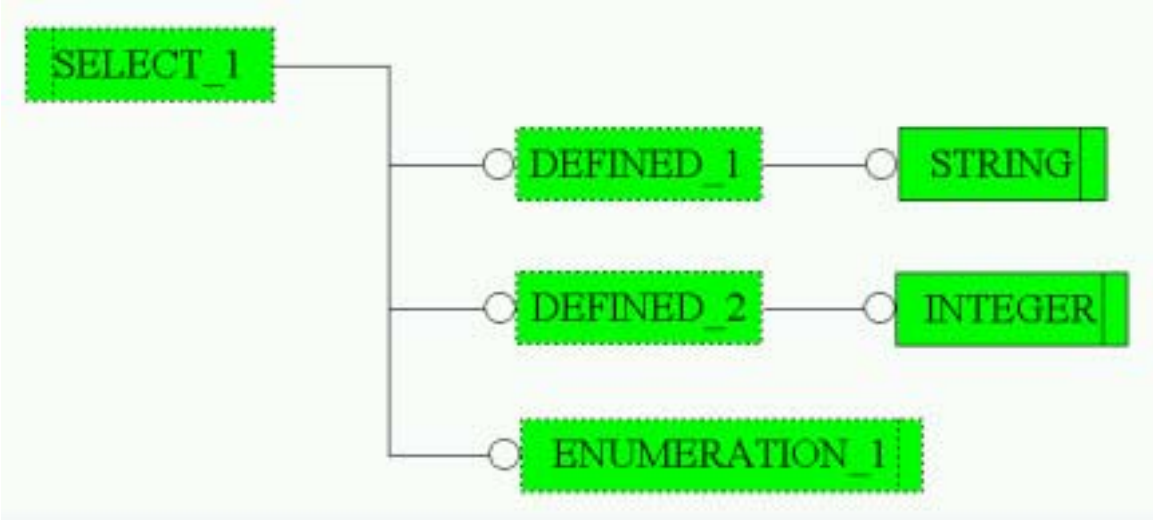

Figure 2 EXPRESS-G for TYPE

ENTITY

`ENTITY`s are the heart of an EXPRESS schema. They collect attributes and constraints together in a manner similar to an ENTITY in IDEF1x or a CLASS in C++. EXPRESS supports single and multiple inheritance such that a child entity inherits all of its parents’

attributes and constraints. Furthermore, an ENTITY may serve as another ENTITY's attribute, thus expanding the TYPE construct described above.

EXAMPLE OF ENTITY

```
ENTITY ENTITY_1;
    ATTRIBUTE_1 : INTEGER;
    ATTRIBUTE_2 : STRING;
    ATTRIBUTE_3 : ENTITY_2;
END_ENTITY;

ENTITY ENTITY_2;
END_ENTITY;

ENTITY ENTITY_3
  SUBTYPE OF(ENTITY_1);
END_ENTITY;
```

This listing shows the definition of three ENTITYs named "ENTITY_1", "ENTITY_2", and "ENTITY_3" respectively. ENTITY_1 has three attributes, one of which is another ENTITY. ENTITY_3 is a subtype of ENTITY_1. This means that an instance of ENTITY_3 will also have three attributes defined because ENTITY_3 inherits all of ENTITY_1's attributes.

EXPRESS-G REPRESENTATION OF ENTITY

Figure 3 shows the EXPRESS-G representation of the previous EXPRESS listing. Each ENTITY is represented as a box with solid borders that contains text. The text is the ENTITY's name. The thin lines with circles show ENTITY_1's attribution. These thin lines represent the *has a* relationship. The thick line between ENTITY_1 and ENTITY_3 shows inheritance. This thick line represents the *is a* relationship.

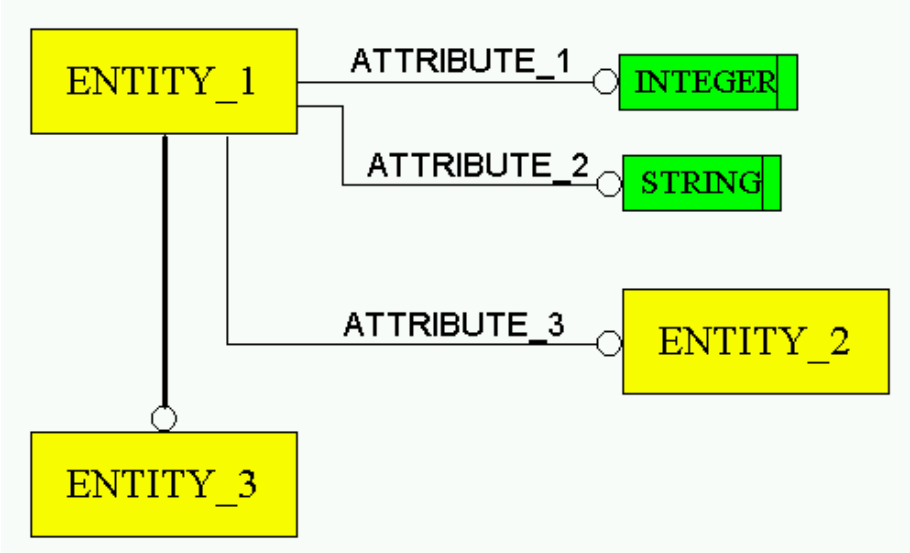

Figure 3 EXPRESS-G for ENTITY

WHERE

Constraints are placed on the sets of legal values for ENTITY attributes by using WHERE. The constraints applied using WHERE are commonly referred to as "where rules" which are different in scope from the global RULEs as discussed later in the paper.

Where rules may be applied to ENTITYs or TYPEs. In both cases, the WHERE clause appears after the primary part of the declaration and before the "END_". RULES may also be applied to a whole collection of entity instances as described in the next section of the paper.

EXAMPLES OF WHERE

```
TYPE day_in_month_number = INTEGER;
WHERE
  wr1: ((1 <= SELF) AND (SELF <= 31));
END_TYPE; -- day_in_month_number
```

The listing above shows how a where rule is used to require that a valid day_in_month_number be an integer between 1 and 31. This constraint applies to all ENTITYs that use this as a data type for an attribute.

```
ENTITY offset_curve_2d
  SUBTYPE OF (curve);
    basis_curve    : curve;
    distance       : length_measure;
    self_intersect : LOGICAL;
  WHERE
    wr1: (basis_curve.dim = 2);
END_ENTITY; -- offset_curve_2d
```

The where rule in the above listing is a little more interesting. It says that the attribute named "basis_curve" is of type "curve", which is an ENTITY; the curve has an attribute named "dim"; and a valid instance of offset_curve_2d must have a value of 2 for dim.

EXPRESS-G REPRESENTATION OF WHERE

There is no graphical symbol for WHERE.

RULE

EXPRESS allows constraints to be applied to whole collections of ENTITYs as well as to individual TYPEs and ENTITYs. These constraints are captured as global rules using the RULE keyword.

EXAMPLE OF RULE

```
RULE application_context_requires_ap_definition FOR
(application_context, application_protocol_definition);

  WHERE
    wr1: (SIZEOF(QUERY ( ac <* application_context | (NOT (SIZEOF(
          QUERY ( apd <* application_protocol_definition | ((ac :=: apd.
             application) AND (apd.
             application_interpreted_model_schema_name =
             'plant_spatial_configuration')) )) = 1)) )) = 0);
```

```
END_RULE; -- application_context_requires_ap_definition
```

The listing above collects a single where rule into a global RULE. This is an example of how ugly constraint specification can be in the EXPRESS language. The RULE says that "For each instance of application_context, there shall be exactly one instance of application_protocol_definition that references the instance of application_context as its application with a value of 'plant_spatial_configuration' as its application_interpreted-_model_schema_name."(ISO 10303-AP227,880).

EXPRESS-G REPRESENTATION OF RULE

There is no graphical symbol for RULE.

FUNCTION

FUNCTIONs allow complex constraints within where rules to be split out from the where rules in a manner similar to splitting out functions as SUBROUTINES within the FORTRAN language. Each function has a signature that defines a return type as well as a list of input types.

EXAMPLE OF FUNCTION

```
FUNCTION acyclic_curve_replica(
               rep: curve_replica;
               parent: curve
      ): BOOLEAN;
    IF NOT ('PLANT_SPATIAL_CONFIGURATION.CURVE_REPLICA' IN
             TYPEOF(parent))
    THEN
      RETURN(TRUE);
    END_IF;
    IF parent :=: rep THEN
      RETURN(FALSE);
    ELSE
  RETURN(acyclic_curve_replica(rep,parent\curve_replica.parent_curve));
    END_IF;
END_FUNCTION; -- acyclic_curve_replica
```

This function is named "acyclic_curve_replica", takes two input parameters (a curve_replica and a curve) and returns a BOOLEAN value (TRUE or FALSE). "The acyclic_curve_replica Boolean function is a recursive function which determines whether, or not, a given curve_replica participates in its own definition. The function returns FALSE if the curve_replica refers to itself, directly or indirectly, in its own definition." (ISO 10303-42,96).

EXPRESS-G REPRESENTATION OF FUNCTION

There is no graphical symbol for FUNCTION.

STEP

Overview

The primary purpose of ISO 10303 is to provide a series of Application Protocols that facilitate the exchange of data between dissimilar automation systems. Each Application Protocol maps domain terminology and data constructs onto a common ISO 10303 framework of integrated generic resources.

Document Numbering

ISO 10303 is organized as a series of parts, each published separately. The structure of this international standard is described in ISO 10303-1. The numbering of the parts of this International Standard reflects its structure:
— Parts 11 to 14 specify the description methods;
— Parts 21 to 29 specify the implementation methods;
— Parts 31 to 35 specify the conformance testing methodology and framework;
— Parts 41 to 50 specify the integrated generic resources;
— Parts 101 to 107 specify the integrated application resources;
— Parts 201 to 237 specify the application protocols;
— Parts 301 to 337 specify the abstract test suites;
— Parts 501 to 520 specify the application interpreted constructs.
A complete list of parts of ISO 10303 is available from the Internet:
http://www.nist.gov/sc4/editing/step/titles/

Application Protocols are the 200 series part of the international standard. The generic integrated resources are numbered from 41 to 50 and contain EXPRESS schemas. Each Application Protocol uses (`USE FROM`) combinations of these generic integrated resource schemas to form a short-form EXPRESS schema that meets the data exchange needs for a domain.

Three Tier Architecture

The STEP framework follows a three-tier approach to data exchange. The first tier captures a data model from the applications domain using a data modeling format with which the domain experts feel comfortable. Many Application Protocols use EXPRESS and EXPRESS-G for this first tier modeling effort. Others use IDEF1x and there is currently an initiative to use the Unified Modeling Language (UML) for this tier.

The second tier maps all of the data requirements and constraints from the first tier onto a set of generic integrated resources. These generic integrated resources are defined using the EXPRESS language and every Application Protocol is mapped onto the same set of generic integrated resources. In theory, similar data requirements from different domains should map to the same EXPRESS constructs in the generic integrated resources. In practice this ideal has not yet been proven.

The third tier is defined by parts 21 through 29 and defines the actual patterns of computer symbols that get exchanged or shared. ISO 10303-21:1994, “Industrial automation systems and integration – Product data representation and exchange – Part 21: Clear text encoding of exchange structure” defines a physical file format that allows data repositories conforming to EXPRESS schemas to be exchanged as ASCII files. Part 22 defines a Standard data access interface specification (SDAI) that allows CORBA-like sharing of data repositories conforming to EXPRESS schemas across networks and between applications.

Application Protocols (APs)

STEP is made up of many parts and its primary value stems from Application Protocols (Parts numbered from 200 to 236). Application Protocols map domain specific data models into EXPRESS schemas based on STEP’s generic integrated resources. Data is shared or exchanged between systems by invoking implementation methods specific to data conforming to EXPRESS schemas.

AP Table of Contents

Just as every EXPRESS schema follows the generic skeleton described above, every STEP Application Protocol (AP) follows the same basic format as outlined in the following skeletal Table of Contents:

```
   Introduction
1  Scope
2  Normative references
3  Terms, definitions, and abbreviations
4  Information requirements
     4.1 Units of functionality
     4.2 Application objects
     4.3 Application assertions
5  Application interpreted model
     5.1 Mapping table
     5.2 AIM EXPRESS short listing
6  Conformance requirements
Annex A (normative) AIM EXPRESS expanded listing
o o o
Annex F (informative) Application activity model
```

Application Protocol Development

Application Protocol development follows the basic outline of the Table of Contents listed above. That is, first the information requirements are gathered and recorded in an application reference model (ARM)—clause 4. Next these requirements are mapped into an application interpreted model (AIM)—clause 5. Finally, the AIM EXPRESS short listing is “compiled” into the AIM EXPRESS expanded listing (or long-form schema) in Annex A.

Actually there is a step in AP development that precedes the ARM. Annex F of an Application Protocol contains an application activity model (AAM), which is a SADT (or IDEF0) decomposition of the domain being supported. The generation of this model usually precedes the development of the ARM. Keeping this artifact in an annex of the AP is useful for people who need to get a quick sense of what parts of the domain are being modeled.

The application objects defined in the ARM are divided into Units of Functionality (UoFs). This helps trace data requirements from the ARM to the AIM and is necessary because most APs have more than 250 application objects.

Parallel between AP Development and the SDLC

The process of developing an AP (AAM→ARM→AIM→Short-form EXPRESS→Long-form EXPRESS) follows that of a software development lifecycle model (SDLC) that "defines before design" and "designs before building". Unlike a software project, however, the product of Application Protocol Development is a data standard not a functioning piece of software. The data standard may itself become a requirement in other software development projects.

It is interesting to note that the EXPRESS language allows for a "compilation" of a short-form EXPRESS model into a long-form EXPRESS model. Furthermore, there are several compilers that translate the long-form EXPRESS into a collection of C++ or Java classes, which can then be compiled as part of a larger software product. The tracking of domain requirements from the ARM to the Long-form EXPRESS entities is very similar to the tracking of user requirements to software functionality in a typical software development project.

By using STEP standards as part of the requirements for new software development efforts, organizations (both developers and customers) can leverage very large data modeling efforts for the cost of purchasing the ISO Standards and the cost of learning how to read the ISO standards.

AP 227 Pipe Definition

This section of the paper traces the definition/representation of a pipe in ISO 10303-227:2001(E) "Industrial automation systems and integration—Product data representation and exchange—Part 227: Application protocol: Plant spatial configuration" through all three tiers of STEP: Application Reference Model, Application Interpreted Model, and Physical File.

Tier 1—Application Reference Model (ARM)

At the application reference model (ARM) tier of the STEP, a pipe is defined in Clause 3, is assigned to a Unit of Functionality in Clause 4.1, and is defined as an object in Clause 4.2 within AP 227. Specifically, Clause 3.3.25 defines a pipe as, "a plant item (see 3.3.32) that is hollow and approximately cylindrical, that may have a constant cross-section along its extent, and that conveys fluid, vapour, or particulate material (see 3.3.22)", and notes that, "Heating, ventilation, and air conditioning (HVAC) duct that has a rectangular cross section is not a pipe." (ISO 10303-227, 11).

Clause 4.1.5 defines the piping_component_characterization Unit of Functionality (UoF). "The piping_component_characterization UoF describes the individual elements of the Piping_system within a Plant. Piping_component objects include pipes, fittings, valves, in-line equipment, and othervelements that regulate, control, or convey Piping_system

fluids." (ISO 10303-227, 22) Clause 4.1.5 also assigns the ARM object Pipe to the piping_component_characterization UoF. This is significant, because locating the pipe object in the mapping table in clause 5 requires knowing to which UoF it belongs.

Finally, clause 4.2.154 defines the Pipe object. "A Pipe is a type of Piping_component (see 4.2.157) that is a hollow cylindrical conveyance, with a constant radius for the cross-sectional circle, for directing fluid, vapour, or particulate flow. Each Pipe may be one of the following: a Mitre_bend_pipe (see 4.2.142), a Nipple (see 4.2.143), a Straight_pipe (see 4.2.232), or a Swept_bend_pipe (see 4.2.248)." This clause also notes that, "In most cases, the Pipe will conform to the dimensional requirements for nominal pipe size as tabulated in national standards such as American National Standards Institute (ANSI) B36.10 and ANSI B36.19.", and that, "This definition does not exclude tubing and flex hoses from consideration as Pipe." (ISO 10303-227, 95)

Tier 2—Application Interpreted Model (AIM)

Figure 4 shows the mapping table entry that maps the Pipe ARM object onto both a piping_component_definition and a piping_component_class in the AIM. This mapping is reflected in the two left-most columns in the table. The right-most column is called the "Reference path" and uses a rather cryptic (but entirely ASCII) syntax to show all of the STEP integrated resources that are required to represent a pipe. If you look closely about 3/4 of the way down the reference path you will see that the "name" attribute of the piping_component_class is required to equal the literal 'pipe'.

Table 6 - Mapping table for piping_component_characterization UoF (continued)

Application element	AIM element	Source	Rules	Reference path
PIPE	#1: (piping_component_definition) #2: (piping_component_class)	227 227	15, 19	(piping_component_definition => product_definition [product_definition product_definition_formation ->] product_definition_formation product_definition_formation.of_product ->] [product classification_item = product classification_item <- applied_classification_assignment.items[1] applied_classification_assignment <- classification_assignment classification_assignment.assigned_classification ->] (group) (group => group_relationship.related_group group_relationship group_relationship.relating_group ->] group) group.name = 'pipe')] [product product.frame_of_reference[1] => product_context=> application_context_element application_context_element.name = 'plant item')]) (piping_component_class => [characterized_object] [group])

ISO/IS 10303-227:2005(E)

© ISO 2005 – All rights reserved

Figure 4 Mapping Table Entry for PIPE

Figure 5 shows an EXPRESS-G representation of the ENTITYs required by the reference path. The key entities to review are the piping_component_definition, which inherits all of the attributes from product_definition, and piping_component_class, which inherits attributes from both group and characterized object. The full long-form EXPRESS schema from which these ENTITYs where copied was available at the time this paper was written at the following URL: http://www.nist.gov/sc4/step/parts/part227/is/wg3n904.exp

NOTE: The two literal values ‘pipe’ and ‘plant item’ from the “Reference path” in Figure 4 are annotated on Figure 5.

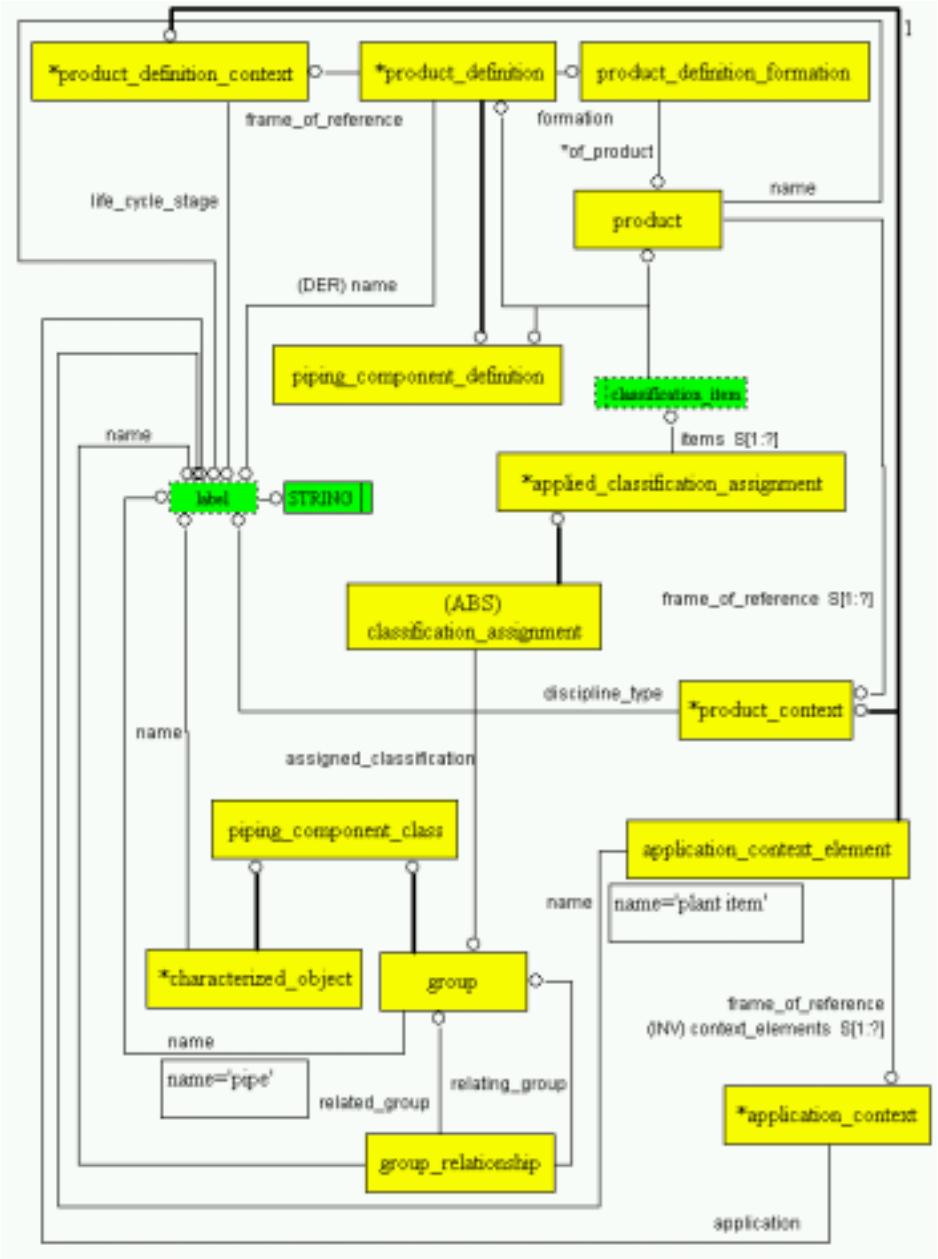

Figure 5 EXPRESS-G Sub-Schema for a Pipe

Tier 3—Physical File

A full discussion of the mapping of data from instances of an EXPRESS schema into a physical file is beyond the scope of this paper. This section of the paper is only intended to expose the reader to a glimpse of the third tier of the STEP framework. It is enough to note here that ISO 10303-21:1994(E) "Industrial automation systems and integration—Product data representation and exchange—Part 21: Implementation methods: Clear text encoding of the exchange structure" defines a mechanism for doing so. The following listing is the result of compiling the long-form schema (wg3n904.exp) mentioned in the Tier 2 discussion using a tool called Ecco (http://ecco.pdtec.de) to generate a simple repository interface application, and then using this application to create the necessary entity instances. The resulting entries in the physical file appear as follows:

```
#0=PIPING_COMPONENT_DEFINITION('dummy1', 'test entity', #3, #7);
#1=PRODUCT_CONTEXT('plant item', #4, 'test');
#2=PRODUCT('test product', 'product 1', 'test product', ());
#3=PRODUCT_DEFINITION_FORMATION('3', 'test
                               product_definition_formation', #2);
#4=APPLICATION_CONTEXT('MSWE646 Demonstration');
#5=APPLIED_CLASSIFICATION_ASSIGNMENT(#6, $, (#2));
#6=PIPING_COMPONENT_CLASS('pipe', 'test pipe', $, $);
#7=PRODUCT_DEFINITION_CONTEXT('pipe_thang', #4, 'before concept');
```

Conclusions

This paper has provided a high-level introduction to the EXPRESS data modeling language and has provided some insight into the application of this language to developing Application Protocols within the STEP (ISO 10303) framework.

By using the EXPRESS language, ISO 10303 Application Protocols meet several of the primary tenants of good software specification. Specifically they are traceable and traced and they are precise. Whether they are also clear and unambiguous is a matter for debate and is outside the scope of this paper. EXPRESS offers an advantage over other languages such as IDEF and UML in that it able to be machine processed and the relationship between its graphical and lexical forms is standardized.

The `USE FROM` mechanism is another advantage of EXPRESS. This mechanism allows patterns of data models to be shared in a standardized manner. STEP's use/abuse of `USE FROM` in its Generic Integrated Resources has resulted in the complexity reflected in Figure 5. Specifically, the STEP methodology led to the need to instantiate eight `ENTITY`s just to say "there is a pipe." Furthermore, to understand the meaning of these entities requires recognizing a patter that consists of the fourteen `ENTITY`s shown in Figure 5.

The process of mapping domain information into data structures is an essential part of any software development effort whether it follows a formal development lifecycle model or not. Software development efforts targeted towards the design of complex systems (like process plants, automobiles, or ships) can gain a great deal of leverage by incorporating STEP Application Protocols. This leverage is amplified by the fact that long-form EXPRESS schemas can be directly compiled into data repositories with application programming interfaces (APIs). As more applications attempt to share data

over the Internet, and as the complexity of the data being shared continues to increase languages like EXPRESS and frameworks like STEP will become even more essential.

References

ISO 10303-1:1994(E) “Industrial automation systems and integration—Product data representation and exchange—Part 1: Overview and fundamental principles”

ISO 10303-11:1994(E) “Industrial automation systems and integration—Product data representation and exchange—Part 11: Description methods: The EXPRESS language reference manual”

ISO 10303-21:1994(E) “Industrial automation systems and integration—Product data representation and exchange—Part 21: Implementation methods: Clear text encoding of the exchange structure”

ISO 10303-42:1994(E) “Industrial automation systems and integration—Product data representation and exchange—Part 42: Integrated generic resources: Geometric and topological representation”

ISO 10303-227:2001(E) “Industrial automation systems and integration—Product data representation and exchange—Part 227: Application protocol: Plant spatial configuration”

Kemmerer, Sharon J. editor, (1999), “STEP The Grand Experience”, NIST Special Publication 939, National Institute of Standards and Technology, CODEN: NSPUE2.