Tomáš Michalčík, Čuboňová Nadežda [1]

DATA TYPES IN THE EXPRESS LANGUAGE

Summary: Defined data types are provided as part of the language. Every attribute, local variable or formal parameter has an associated data type. The core information modelling language consists of data types that determine its modelling capabilities. Based on the data types of traditional programming languages, language EXPRESS absorbing mechanisms used in object-oriented technologies, such as inheritance mechanism which is used to describe the production data, using a wide range of data types. This article dealt with the data types of EXPRESS language, and describe their basic options.
Keywords: data type, capabilities, value, domain.

INTRODUCTION

EXPRESS language is defined by ISO 10303-11. It is a modelling language for data, which provides a sufficiently rich set of resources to define complex data types. This language is object oriented and easily understood by humans, and also workable with computer as well other computer languages. However this language cannot be considered to be a programming language. Unlike these languages, which are tools for application programming, language EXPRESS is used for object-oriented conceptual description of the products [1, 3]. Because it is highly versatile, it can be used to describe any objects or processes of the real world. The structure of language is similar to other object-oriented programming language, especially language, which define the structure of object-oriented databases.

Data types of the EXPRESS language are classified as [1]:

- simple data types,
- aggregation data types,
- named data types,
- constructed data types,
- generalized data types.

Data types are also classified according their usage as:

- instantiable data types,
- parameter data types,
- underlying data types.

I will be the relationships between these two classifications explain in the last chapter.

[1] University of Zilina, Department of Automation and Production Systems, Zilina, Slovak Republic.

SIMPLE DATA TYPES

The simple data types define the domains of the fundamental data units in EXPRESS. That is, they cannot be further subdivided into elements that EXPRESS recognizes. The simple data types are (ISO 10303-11):

- NUMBER,
- REAL,
- INTEGER,
- STRING,
- BOOLEAN,
- LOGICAL,
- BINNARY.

Number data types

The NUMBER data type has as its domain all numeric values in the language and shall be used when a more specific numeric representation is not important. Since we may not know the context of size, we do not know how to correctly represent it. The size of the crowd at a stadium, for example, would be an INTEGER, whereas the area of the pitch would be a REAL.

Real data type

The real data type has as its domain all rational, irrational and scientific real numbers. It is a specialization of the NUMBER data type. Rational and irrational numbers have infinite resolution and are exact. Scientific numbers represent quantities which are known only to a specified precision. A real number literal is represented by a mantissa and optional exponent. The number of digits making up the mantissa when all leading zeros have been removed is the number of significant digits. The known precision of a value is the number of leading digits that are necessary to the application [1].

Integer data type

The INTEGER data type has as its domain all integer numbers. It is a specialization of the real data type. This example (fig. 1) uses an INTEGER data type to represent an attribute named `elements`. The domain of this attribute is all integers, with no further constraint.

```
ENTITY too;
   components : INTEGER;
   ...
END_ENTITY;
```

Fig. 1. Integer data type

Logical data type

The LOGICAL data type has as its domain the three literals TRUE, FALSE and UNKNOWN. The following ordering holds for the values of the LOGICAL data type: FALSE < UNKNOWN < TRUE. The LOGICAL data type is compatible with the BOOLEAN data type, except that the value UNKNOWN cannot be assigned to a boolean variable.

Boolean data type

The BOOLEAN data type has as its domain the two literals TRUE and FALSE [1]. The BOOLEAN data type is a specialization of the LOGICAL data type. The same ordering holds for values of the BOOLEAN data type as for values of the LOGICAL data type, that is: FALSE < TRUE. In this example (fig. 2), an attribute named planar is represented by the BOOLEAN data type. The value for planar associated with an instance of surface can be either TRUE or FALSE.

```
ENTITY surface;
   planar : BOOLEAN;
   ...
END_ENTITY;
```

Fig. 2. Boolean data type

String data types

The STRING data type has as its domain sequences of characters. A string data type may be defined as either fixed or varying width (number of characters). If it is not specifically defined as fixed width (by using the fixed reserved word in the definition) the string has varying width. The domain of a fixed width string data type is the set of all character sequences of exactly the width specified in the type definition. The domain of a varying width string data type is the set of all character sequences of width less than or equal to the maximum width specified in the type definition.

The following defines a varying length string; values of which have no defined maximum length.

```
string1 : STRING;
```

The following defines a string that is a maximum of ten characters in length; values of which may vary in actual length from zero to ten characters.

```
string2 : STRING(10);
```

The following defines a string that is exactly ten characters in length; values of which must contain ten characters.

```
string3 : STRING(10) FIXED;
```

Fig. 3. String data types

If no width is specified, the domain is the set of all character sequences, with no constraint on the width of these sequences. The case (upper or lower) of letters within a string is significant. The width expression shall evaluate to a positive integer value. Examples are presented on fig. 3.

Binary data type

The BINARY data type has as its domain sequences of bits, each bit being represented by 0 or 1. A BINARY data type may be defined as either fixed or varying width (number of bits). If it is not specifically defined as fixed width (by using the FIXED reserved word in the definition) the binary data type has varying width. The domain of a fixed width BINARY data type is the set of all bit sequences of exactly the width specified in the type definition. The domain of a varying width BINARY data type is the set of all bit sequences of width less than or equal to the maximum width specified in the type definition. If no width is specified, the domain is the set of all bit sequences, with no constraint on the width of these sequences. The width expression shall evaluate to a positive integer value. The following example (fig. 3) might be used to hold character font information.

```
ENTITY character;
   representation : ARRAY [1:20] OF BINARY (8) FIXED ;
END_ENTITY;
```

Fig. 4. Binary data type

AGGREGATION DATA TYPES

Aggregation data types have as their domains collections of values of a given base data type. These base data type values are called elements of the aggregation collection. EXPRESS provides for the definition of four kinds of aggregation data types (ISO 10303-11):

– ARRAY,
– BAG,
– LIST,
– SET.

An ARRAY is a fixed-size ordered collection. It is indexed by a sequence of integers. A transformation matrix (for geometry) may be defined as an array of arrays (of numbers).

A LIST is a sequence of elements which can be accessed according to their position. The number of elements in a list may vary, and can be constrained by the definition of the data type. The operations of a process plan might be represented as a list. The operations are ordered, and operations can be added to or removed from a process plan.

A BAG is an unordered collection in which duplication is allowed. The number of elements in a bag may vary, and can be constrained by the definition of the data type. The collection of fasteners used in an assembly problem could be represented as a bag. There might be a number of elements which are equivalent bolts, but which one is used in a particular hole is unimportant.

A `SET` is an unordered collection of elements in which no two elements are instance equal. The number of elements in a set may vary, and can be constrained by the definition of the data type. The population of people in this world is a set.

Array data type

An array data type has as its domain indexed, fixed-size collections of like elements. The lower and upper bounds, which are integer-valued expressions, define the range of index values, and thus the size of each array collection. An array data type definition may optionally specify that an array value cannot contain duplicate elements. It may also specify that an array value need not contain an element at every index position. This example (fig. 4) shows how a multi-dimensioned array is declared.

```
Sectors  : ARRAY [ 1 : 10 ] OF                  -- first
dimension
             ARRAY [ 11 : 14 ] OF               -- second
dimension
             UNIQUE something;
```

Fig. 5. Array data type

The first array has 10 elements of data type `ARRAY [11:14] OF UNIQUE something`. There is a total of 40 elements of data type `something` in the attribute named `sectors`. Within each `ARRAY [11:14]`, no duplicates may occur; however, the same `something` instance may occur in two different `ARRAY [11:14]` values within a single value for the attribute named `sectors`. If the unique keyword is specified, each element in an array value of this data type shall be different from (that is, not instance equal to) every other element in the same array value.

List data type

A `LIST` data type has as its domain sequences of like elements. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a `LIST` data type. A `LIST` data type definition may optionally specify that a list value cannot contain duplicate elements. This example (fig. 5) defines a list of arrays. The list can contain zero to ten arrays. Each array of ten integers shall be different from all other arrays in a particular list.

```
complex_list : LIST[0:10] OF UNIQUE ARRAY[1:10] OF INTEGER;
```

Fig. 6. List data type

If the unique keyword is specified, each element in a list value of this data type shall be different from (that is, not instance equal to) every other element in the same list value.

Bag data type

A bag data type has as its domain unordered collections of like elements. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a bag data type. This example (fig. 7) defines an attribute as a bag of point (where point is a named data type assumed to have been declared elsewhere).

```
a_bag_of_points : BAG OF point;
```

Fig. 7. Bag data type1

The value of the attribute named `a_bag_of_points` can contain zero or more points. The same point instance may appear more than once in the value of `a_bag_of_points.` If the value is required to contain at least one element, the specification can provide a lower bound, as in fig. 8.

```
a_bag_of_points : BAG [1:?] OF point;
```

Fig. 8. Bag data type2

The value of the attribute named `a_bag_of_points` now must contain at least one point.

Set data type

A set data type has as its domain unordered collections of like elements. The set data type is a specialization of the bag data type. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a set data type. The collection defined by set data type shall not contain two or more elements which are instance equal. This example (fig. 9) defines an attribute as a set of points (a named data type assumed to have been declared elsewhere).

```
a_set_of_points : SET OF point;
```

Fig. 9. Set data type1

The attribute named `a_set_of_points` can contain zero or more points. Each point instance (in the set value) is required to be different from every other point in the set. If the value is required to have no more than 15 points, the specification can provide an upper bound, as in fig. 10.

```
a_set_of_points : SET [0:15] OF point;
```

Fig. 10. Set data type2

The value of the attribute named `a_set_of_points` now may contain no more than 15 points.

NAMED DATA TYPES

The named data types are the data types that may be declared in a formal specification. There are two kinds of named data types: entity data types and defined data types (ISO 10303-11).

Entity data type

Entity data types are established by entity declarations. An entity data type is assigned an entity identifier by the user. An entity data type is referenced by this identifier. This example (fig. 11) uses a point entity data type as the representation of an attribute.

```
ENTITY point;               ENTITY line;
   x, y, z : REAL;            p0, p1 : point;
END_ENTITY;                 END_ENTITY;
```

Fig. 11. Entity data type

The line entity has two attributes named `p0` and `p1`. The data type of each of these attributes is `point`.

Defined data type

Defined data types are declared by type declarations. A defined data type is assigned a type identifier by the user. A defined data type is referenced by this identifier. The following example (fig. 12) is a defined data type used to indicate the units of measure associated with an attribute.

```
TYPE volume = REAL;          ENTITY PART;
END_TYPE;                    ...
                                bulk : volume;
                             END_ENTITY;
```

Fig. 12. Defined data types

The attribute named `bulk` is represented as a real number, but the use of the defined data type, `volume`, helps to clarify the meaning and context of the real number; i.e., it means volume, rather than some other thing which might be represented by a REAL.

CONSTRUCTED DATA TYPES

There are two kinds of constructed data types in EXPRESS: ENUMERATION data types and SELECT data types. These data types have similar syntactic structures and are used to provide underlying representations of defined data types (ISO 10303-11).

Enumeration data types

An enumeration data type has as its domain an ordered set of names. The extent of this set of names is determined depending on the type of ENUMERATION data type. The following types of ENUMERATION data types are distinguished:

– ENUMERATION that is extensible,
– ENUMERATION that is extending an extensible ENUMERATION, in other words: that is based on an extensible ENUMERATION,
– ENUMERATION that is neither extensible nor extending.

This example uses enumeration data types to show how different kinds of vehicles might travel.

```
TYPE car_can_move = ENUMERATION OF
   (left, right, backward, forward);
END_TYPE;

TYPE plane_can_move = ENUMERATION OF
   (left, right, backward, forward, up, down);
END_TYPE;
```

Fig. 13. Enumeration data type1

```
TYPE colour  = EXTENSIBILE ENUMERATION;  END_TYPE;

TYPE stop_light  = ENUMERATION BASED_ON colour WITH (red,
yellow, green); END_TYPE;

TYPE Canadian_flag  = ENUMERATION BASED_ON colour WITH (red,
white); END_TYPE;
```

Fig. 14. Enumeration data type2

The enumeration item left has two independent definitions, one being given by each type of which it is a component. There is no connection between these two definitions of the identifier left. A reference to left or right, by itself, is ambiguous. To resolve the ambiguity, a reference to either of these values must be qualified by the type name, e.g., `car_can_move.left`. The following EXPRESS results in a single enumeration item named red as both `stop_light` and `canadian_flag` extend the domain of colour.

Select data type

A `SELECT` data type defines a data type that enables a choice among several named data types. The `SELECT` data type is a generalization of these named data types in its domain. The defined type for which the `SELECT` data type is the underlying representation, may add constraints on its domain by declaring local rules. A `SELECT` data type may be `EXTENSIBLE` or not.
Examples:
If `a` and `b` are subtypes of `c` and if they are related by an ANDOR expression, and if there is a data type defined by `SELECT (a,b)`, then it may be that the value of the `SELECT` data type is an `a` and a `b` at the same time.

```
TYPE attachment_method = EXTENSIBLE SELECT (nail, screw);
END_TYPE;

TYPE permanent_attachment = SELECT BASED_ON
attachment_method WITH (glue, weld);
END_TYPE;
……….
ENTITY wall_mounting;
   Mounting     : product;
   On      :    wall;
   Using   :    attachment_method;
END_ENTITY;
```

Fig. 15. Select data type

A `wall_mounting` attaches a `product` onto a `wall` using an attachment method. The initial attachment method describes temporary attachment methods. These methods are then extended to add permanent attachment methods. A value of a `wall_mouting` will have a using attribute that is a value of one of `nail, screw, glue` or `weld.`

GENERALIZED DATA TYPES

The generalized data types are used to specify a generalization of certain other data types, and can only be used in certain very specific contexts. The generic type is a generalization of all data types. The aggregate data type is a generalization of all aggregation data types [1, 2]. The general aggregation data type is a generalization of the aggregation data types that relaxes some of the constrains normally applied to aggregation data types. A `generic_entity` data type is a generalization of all entity data types and a subtype of the generic data type. The `generic_entity` data type is used to constrain extensible select data types. It is also a valid parameter data type for the formal parameters in functions and procedures and for the type representation of attributes in abstract entity data type declarations (ISO 10303-11).

DATA TYPE USAGE CLASSIFICATION

Data types are used in six different ways in EXPRESS (ISO 10303-11):

- as data types of the elements of aggregation data types;
- as the numbers of a select list in defining or extending a select data type;
- as the underlying types of data types;
- as the data types of attributes of entity data types;
- as the data types of constants;
- as the data types of formal parameters and local variables in functions and procedures.

Data types are classified according to their usage as follows (tab. 1):

- instantiable data types are used as representations aggregation elements and as the data types of constants;
- parameters data types are used as representations for explicit and derived attributes and for formal parameters, function results, and local variables in functions and procedures;
- underlying data types are used as representations for defined data types;
- named data types are used as the members of a select list – the possible representations of a value of the SELECT data type.

Table 1. Use of data types

Type	a	b	c	d
Simple data types	•	•	•	
Aggregation Data Types	•	•	•	
Named Data Types	•	•	•	•
Constructed Data Types			•[1)]	
Generalized Data Types		•		

a) Instantiable data types – representation of aggregation elements and constants.
b) Parameter data types – representation of an explicit or derived attribute, a formal parameter, local variable, or function result.
c) Underlying data types – representation of a defined type.
d) Named data types – possible representations of a SELECT data type.
1)only the defined data type from the named data types may be used as an underlying data type.

Instantiable data types: instantiable data types are used as the representations of constants, as the representations of elements of aggregation data types, and as the representations of attributes of non-abstract entity data types.

Parameter data types: parameter data types are used as the representation of attributes of entity data types or as the representation of formal parameters to algorithms (functions and procedures). The parameter data types may also be used to represent the return types of functions and the local variables declared in algorithms.

Underlying data types: underlying data types are used as the representation for defined data types.

CONCLUSION

Express language contains all known and widely used data types. If these predefined types are not sufficient or satisfying, it is possible to create new based on the above. The aim of this article was an explanation of the data types in the EXPRESS language [2]. Data types are one of the foundations of language EXPRESS and therefore their knowledge is needed for my dissertation which is entitled: "Application of the file transfer program data for CNC machines using the EXPRESS language”. This is a theoretical basis for my dissertation thesis.

REFERENCES

1. The EXPRESS Definition Language for IFC Development. Available on the Internet: http://www.civ.utoronto.ca/sect/coneng/i2c/civ1283/civ1283-ref-final/civ1283-basic%20ref/ifc/express-101.pdf >
2. Michalčik T.: 2010. *Možnosti formátu STEP NC pri programovaní NC strojov:* Diploma thesis. Žilina: SjF ŽU, 2010. s. 47-54.
3. Xun H., Nee Y. C.: Advanced design and manufacturing based on STEP. Springer-Verlag London, 2009, 465.

RODZAJE DANYCH W JĘZYKU EXPRESS

Streszczenie
Zdefiniowane typy danych są dostarczane jako część języka programowania. Każdy atrybut, zmienna lokalna lub parametr formalny ma skojarzony typ danych. Kod źródłowy języka programowania zawiera typy danych, które określają jego możliwości programowania. W oparciu o typy danych z tradycyjnych języków programowania, język EXPRESS wykorzystuje mechanizmy stosowane w technologiach programowania obiektowego, takie jak mechanizm dziedziczenia, który jest używany do opisu procesu produkcyjnego, przy użyciu szerokiego zakresu typów danych.
Słowa kluczowe: typy danych, możliwości programowania, domena.

Note: The solved problem supports a partial solution of EU Operational Programme – Education: “A Flexible and Attractive Study on University of Žilina for Both Employment Market and Knowledge Society Requirements”; Code ITMS 26110230005.