Data Types in OOA[1]

Sally Shlaer
Stephen J. Mellor

http://www.projtech.com

1. Introduction

Now that code generation from OOA models has become a reality, we have been motivated to re-examine how to specify the legal values for the data elements (attributes, event data items, and data elements that appear on ADFDs and/or in an action language). While formerly we required that a set of legal values (a domain) be specified for each attribute and that every event data item and data element on an ADFD be an attribute, we now require only that

RULE: All data elements that appear in the OOA models of a domain must be typed.

Data types in OOA are based on a two-level scheme:

- Domain-specific data types. Domain-specific data types are defined by the analyst in order to capture ideas such as power, voltage, position, temperature, etc. for a particular domain. The analyst defines an appropriate set of domain-specific data types for the domain, stating a name for each data type and other pertinent information such as precision, range, and the like.

- Base data types. Base data types are defined as part of the Shlaer-Mellor Method. The base data types include numeric, symbolic, enumerated, and similar concepts.

When an analyst defines a domain-specific data type, he or she does so by referring to an appropriate base data type. The selection of a base type is very important, in that it restricts what you can do with a data element that has that type: You can do arithmetic on elements that are based on the numeric type; you can add duration to time to get a new time, but you cannot add two times. If you have an enumerated data type, you cannot do arithmetic using elements that are so typed—even if the legal values of the enumeration are 1, 2, 3, 4. According, we now examine the base data types and their properties.

2. Using the Base Data Types

The base data types defined for OOA are[2]:

• enumerated	allows for a finite set of data values
• boolean	used to type data elements that can have values of True or False
• extended boolean	used to type data elements that can have values of True, False, and Non-comparable
• symbolic	used to type data elements that have the nature of names and descriptions
• numeric	supports data elements that can be used in arithmetic computation
• ordinal	allows for data values that express order.
• time	absolute time, in the sense of time and date

[1] This article is an excerpt from our upcoming book on Recursive Design.
[2] This list is extended in subsequent chapters by the addition of metatypes—a set of base types required for building bridges.

- duration an amount of time
- arbitrary used for identifiers that have no intrinsic meaning

The base types are used for two different purposes:

- as a basis for definition of the domain-specific data types in OOA
- for mapping into the implementation types native to the programming language(s) used for code generation in RD[3].

Exactly how you go about defining a domain-specific data type depends on the particulars of your automation tools; only the required information content is specified here. In the following discussion,

- A name supplied by the analyst is indicated as <some name>.
- Alternative items (meaning that the analyst must select one of the alternatives) are shown as [alt1 | alt2 | ...]
- Optional items are shown in parentheses

As a general strategy, we suggest that the analyst provide as much of the optional information as possible so that the most effective choices can be made when the mapping is made to the implementation types.

Enumerated data types. If a data type permits a finite set of values, define it as:

data type <name> is enumerated
values are <value_1>, <value_2>, . . . <value_N>
(default value is <value_k>)[4]

as in

data type IC color is enumerated
values are red, blue, black, green, silver

The only operations permitted for data elements of an enumerated data type are the comparison operations, represented as = (identical in value) and ≠ (not identical in value). The result of either comparison yields a data element of type boolean.

Boolean and extended boolean data types. The boolean base data type is exactly what you expect: a pre-defined enumerated data type with values True and False. An extended boolean base data type is very similar, and permits the values True, False, and Non-comparable. The motivation for the extended boolean type will become apparent when we discuss ordinal types, below.

To define a domain-specific data type based on a boolean or extended boolean base type, write

data type <name> is boolean
(default value is <value>)

or

---

[3] Base types are a concept different from that of implementation types. As a reminder to the analyst, we have therefore chosen names for the base types that are reasonably independent of those used in current programming languages.

[4] In general, we avoid the notion of defaults throughout the method, reasoning that an analyst should be encouraged to think through and make each decision explicitly. However, in deference to analysts who must state default values -- as in 2167A and similar standards -- OOA supports the concept.
If the analyst does not specify a default value for a data type, the value "UNDEFINED" is assumed.

data type <name> is extended boolean
(default value is <value>)

The operations permitted for data elements based on these base types include the comparison operations, represented as = (identical in value) and ≠ (not identical in value). The result of either comparison yields a data element of base type boolean.

For the boolean base data type, the logical operations ¬, ∧, and ∨ (not, and, inclusive or, respectively) are defined in the standard way. For the extended boolean base type, the operations are defined as given in the following tables. Here T, F, NC, and UNDEF represent True, False, Non-comparable, and Undefined, respectively

A	¬A
T	F
F	T
NC	NC

A	B	A∧B
T	T	T
T	F	F
T	NC	UNDEF
F	T	F
F	F	F
F	NC	F
NC	T	UNDEF
NC	F	F
NC	NC	UNDEF

A	B	A∨B
T	T	T
T	F	T
T	NC	T
F	T	T
F	F	F
F	NC	UNDEF
NC	T	T
NC	F	UNDEF
NC	NC	UNDEF

Symbolic data types. For data elements that have the nature of names, we need to be able to define symbolic data types:

data type <data type name> is symbolic
length is (from <minimum number of characters> to) <maximum number of characters>
(default value is <character string>)

The analyst specifies the maximum and minimum number of characters required based on his or her knowledge of the longest and shortest plausible values. Hence

data type gas name is symbolic
length is from 2 to 15
default value is Helium

The operations defined for symbolic data types are

- concatenate (represented as +); the result of concatenation is a data element of base type symbolic.
- comparison for identical value, represented as = (identical in value) and ≠ (not identical in value). The result of such a comparison yields a data element of base type boolean.

- comparison for position in a collating sequence[5], represented as < (before), > (after), ≤ (before or identical), and ≥ (identical or after). The result of such a comparison yields a data element of base type boolean.

Numeric data types. If a data type is numeric in nature, write

```
data type <data type name> is numeric (base <N>)
    range is from <low limit> to <high limit>
    units are <unit symbol>
    precision is <smallest discriminant>
    ( default value is <value> )
```

where base N specifies the base of the quantities <low limit>, <high limit>, <smallest discriminant> and <value>. If base N is omitted, base 10 is assumed. Hence

```
data type ring diameter is numeric
    range is from 0 to 39
    units are cm
    precision is 0.01

data type bit pattern is numeric base 8
    range is from 0 to 177777
    units are octal bits
    precision is 1
```

Note that the analyst does not specify whether a numeric data type will be implemented as an integer or a real number. This will ultimately be determined by the architecture, based on the native types available in the implementation language, the word length of these native types, and the range and precision required for the data type. As a result, the OOA models of any domain are entirely decoupled from the implementation technology, thereby maximizing the potential for reuse across a wide range of platforms and implementation languages.

The operations permitted for numeric data types are:

- the standard arithmetic operations +, -, * (multiplication), / (division), % (division modulo N), and ** (exponentiation). The result of such an operation is again of base type numeric.
- the standard arithmetic comparisons of =, ≠ , <, >, ≤, and ≥. The result of such a comparison yields a data element of base type boolean.

Ordinal data types. Ordinal data types are used to express order, such as first, second, and so on. However, the subject of ordering is a lot more interesting that just this common example; hence the following digression.

An *ordering* is always applied to a set of elements. The set can be finite or infinite.

There are two types of orderings to consider. The first is the most familiar; it is a *complete ordering*. What this means is that you can express the concept of "before" (represented as <) between any two

---

[5] A collating sequence prescribes the order of all the characters in a specified character set, typically including letters, numbers, and punctuation marks. Collating sequences are defined in the implementation environment, and may vary from country to country depending on the concept of "alphabetical order" and the repertoire of characters or symbols used in the natural language.

members of the set. Hence, 7 is before 26 (7 < 26). A complete ordering has the property of transitivity: If A is before B, and B is before C, then A is before C.

A practical example would be the ordering of the cars that make up a freight train. Assume we define a first car. Then we could pick any two cars and easily determine which one was before the other.

Far more interesting are the partial orderings. Consider this sketch of a partial ordering.

```
A  ⇒  B  ⇒  C  ⇒  D
            ⇓
            E  ⇒  F  ⇒  G
```

Using the obvious interpretation, we can say that A < B (A is before B), C < D, C < E, and E < F. But we cannot say anything about the relationship between D and F: They are non-comparable.

Examples of structures that are partially ordered include PERT charts, trees used for any purpose, interlock chains, the connectivity of an electric grid, and the like. All of these can be modeled in complete detail using standard OOA relationships; for examples see [Starr96] and Chapter 4 of *Shlaer-Mellor Method: The OOA96 Report*. Note, however, that when modeling such a structure, one frequently finds it necessary to employ quite a number of ancillary objects (such as root node, parent node, child node, and leaf node) together with a significant set of relationships—all required to express a generally well-known concept. While this can be quite satisfying when one is in a purist frame of mind, the pragmatist points out that such constructions are often of limited value, obscuring, as they can, the true purpose of the model. This becomes particularly pertinent when constructing models of an architecture, where ordering is a particularly prominent theme (see *The Timepoint Architecture* chapter). Hence we have defined the ordinal base data type, leaving it to the judgment of the analyst as for when to use an ordinal attribute as opposed to using more fully expressive OOA objects and relationships.

Returning now to the main theme, an ordinal data type is defined by:

data type <data type name> is ordinal

The operations permitted for ordinal data types are

- the comparisons = and ≠ (identical and not identical in value)
- the comparisons < (read as "before"), >, ≤, and ≥. Each such comparison yields a data element of base type boolean if the ordering is complete, and of base type extended boolean if the ordering is partial.

Time and duration. To define a data type that represents calendar-clock time, write

data type <data type name> is time
range is from <year-mon-day> (<hour:min:sec >) to <year-mon-day> (<hour:min:sec >)
precision is <smallest discriminated value> [year | month | day | hour | minute | second | millisec | microsec]

As an example, if you want to keep track of the delivery date of gas bottles to the nearest hour, write

data type delivery date is time
range is from 1990-1-1 to 2049-12-31 23:00:00
precision is 1 hour

Similarly, to define a data type that represents duration, write

data type <data type name> is duration
range is from <low limit> to <high limit>
units are [year | month | day | hour | minute | second | millisec | microsec]
precision is <smallest discriminated value>

For example

data type test window is duration
range is from 0 to 10
units are second
precision is .001

The operations permitted using data types based on time and duration are:

time := time ± duration
duration := duration ± duration
duration := duration * numeric
duration := duration / numeric
duration := time - time

as well as the standard comparisons of < (read as "before"), >, ≤, and ≥. Each such comparison yields a data element of base type boolean. Comparisons are defined only between elements of the same base type —that is, you can compare time with time and duration with duration, but not time with duration.

Arbitrary data types. To define a data type for data elements that represent arbitrary identifiers:

data type <data type name> is arbitrary

as in

data type reference time ID is arbitrary

The implementation of an arbitrary type—like all the base data types—is determined by the architecture domain. Hence the analyst should make no assumptions as to how this is done: the arbitrary type may be implemented as a handle, an integer, a character string, or by any other scheme the architects devise. For this reason, the analyst cannot specify a default value for the base data type arbitrary.

3. Using Domain-Specific Data Types

For each non-referential attribute on the Object Information Model, specify its domain-specific data type in the *Object and Attribute Descriptions* document. The domain-specific data type replaces the formerly required "attribute domain." For each referential attribute, as before, state the object and attribute that is being referred to, as in the following example:

OBJECT: DIGITAL OUTPUT POINT

Digital Output Point (Digital Output Point ID, Digital Output Register ID (R34), Starting Bit, Mask, Raw Data Value, Interpreted Value)

A Digital Output Point represents the latest command sent to some Actuator For Digital Output Point.

DIGITAL OUTPUT POINT ID. An identifier for the Digital Output Point.

Data type: output point ID

DIGITAL OUTPUT REGISTER ID. The Digital Output Register used to address the Digital Output Point.

Data type: refers to Digital Output Register.Digital Output Register ID

STARTING BIT. The lowest numbered bit (corresponding to the lowest numbered terminal) in the associated Terminal Group for Digital Output Register.

Data type: bit number
. . .

For each supplemental data item carried by an event specify a domain-specific data type. This is most conveniently done via the event list, using a form such as:

Event label	**Meaning**	**Supplemental data name**	**Supplemental data type**	**Source**
AIP1	Convert Analog Input Point	Raw Data Value	analog raw	UAIP TAIP
DIP1	Read Digital Input Point	none	- - -	client
TAIP1	Read Waveform Data	Offset, Client Return	time offset, client return	client
TAIP2	Data Present	none	- - -	PIO hardware (interrupt)
TAIP3	Conversion Complete	Engineering Unit Value	engineering unit value	AIP
UAIP1	Read Untimed Analog Input Point	none	- - -	client
UAIP2	Conversion Complete	Engineering Unit Value	engineering unit value	AIP

Finally, record the definitions of all domain-specific data types in a separate document. You may turn to any of the example chapters in Part IV to find examples of such a Domain-Specific Data Type document, as well as representative Object and Attribute Descriptions.

Acknowledgments. The original research that incorporated into OOA the concept of domain-specific data types as distinct from base and implementation data types was carried out by Mark Lloyd.

