

The Formal Specification of the ISO *Open Document Architecture* (ODA) Standard

W. APPELT* AND N. TETTEH-LARTEY†

* Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany

† The National Computing Centre, Oxford Road, Manchester M1 7ED, UK

This paper presents a survey on an application of a formal description technique called IMC and language called IMCL in the context of an International Standardization project on Document Structure and Processing and gives a brief tutorial on the use of IMCL.

Received April 1992, revised July 1992

1. INTRODUCTION

1.1. What is ISO 8613—ODA?

ISO 8613—*Open Document Architecture (ODA) and Interchange Format*¹ is a multi-part International Standard [5], the development of which was carried out in harmony with the CCITT (the International Consultative Committee for Telephony and Telegraphy) Study Group VIII, who produced the T.410 Series of Recommendations—technically identical to ISO 8613. The driving force for the development of ODA was the need for *open transfer* of 'fixed' or 'revisable' documents. Open transfer means that a recipient and creator need no *a priori* agreements and thus need not have any understanding of each other's systems in order to exchange documents. For a complete and concise description of the ODA standard see [1].

ODA was produced to satisfy an increasingly rapid growing need for users to *easily* share electronically held information across many different computer systems. This is particularly true for large user organizations who typically have a variety of equipment each with a number of word/document processing packages. These organizations have turned to communication protocol Standards to provide interconnectivity between different equipment and harmonized interworking (i.e. *open transfer*).

To achieve *open transfer* it is expected that OSI (Open Systems Interconnection) protocols such as FTAM (ISO 8571—File Transfer, Access and Management) and MHS (CCITT's X.400 series of recommendations for Message Handling Systems) or at a simpler level just exchange of magnetic media will be employed. The ODA documents are represented using the *Open Document Interchange Format (ODIF)*, Part 5 of ISO 8613. ODIF (the transfer syntax) is encoded for system independent data transfer using the standard ASN.1 encoding rules (ISO 8825—the Specification of basic encoding rules for Abstract Syntax Notation One). At present, ODA sup-

ports three kinds of content, i.e. character, raster graphics and geometric graphics content which are based on ISO 6937 (the Coded character sets for text communication) for character content architecture, CCITT T.4, T.6 (Group 3 and Group 4 facsimile coding schemes) and a 'bitmap' encoding for the raster graphics content architecture and ISO 8623 (the Computer Graphics Metafile—CGM) for the geometric graphics content architecture. ODA can achieve all this through standardizing the semantics of the structural and content elements of documents.

The use of ODA is considered central by ISO and CCITT for the transfer of information within and between Open systems, and a framework of extensions to ODA is currently being developed by these organizations to allow a proper interrelationship with existing standards such as those for user-system interfaces and data bases.

1.2. The ODA information model

ODA clearly separates the logical aspects of documents containing elements such as author, chapter and paragraph and the layout aspects containing elements such as page, column and blocks from the type of content (character text, raster and/or geometric graphics) which is shared by both views. ODA thus clearly demonstrates its *object-oriented* architectural model concepts, which in the future will easily allow for additional content types such as audio, mathematical equations and video to be incorporated. According to this model a document is viewed as a set of objects with each object consisting of a set of attributes as shown in Figure 1. Here the components of the ODA information model (sometimes called the descriptive representation) are shown. These being the specific structure (or description) which refers to an instance of a document and not generic structure which refers to a class of documents. The logical structure (or description) groups the contents of a document according to its logical view. Conversely, the layout structure (or description) looks at a document from its physical appearance and groups its contents accordingly.

¹ The original title '*Office Document Architecture (ODA) and Interchange Formats*' had been changed by ISO in 1991 to reflect the broader scope of the Standard.

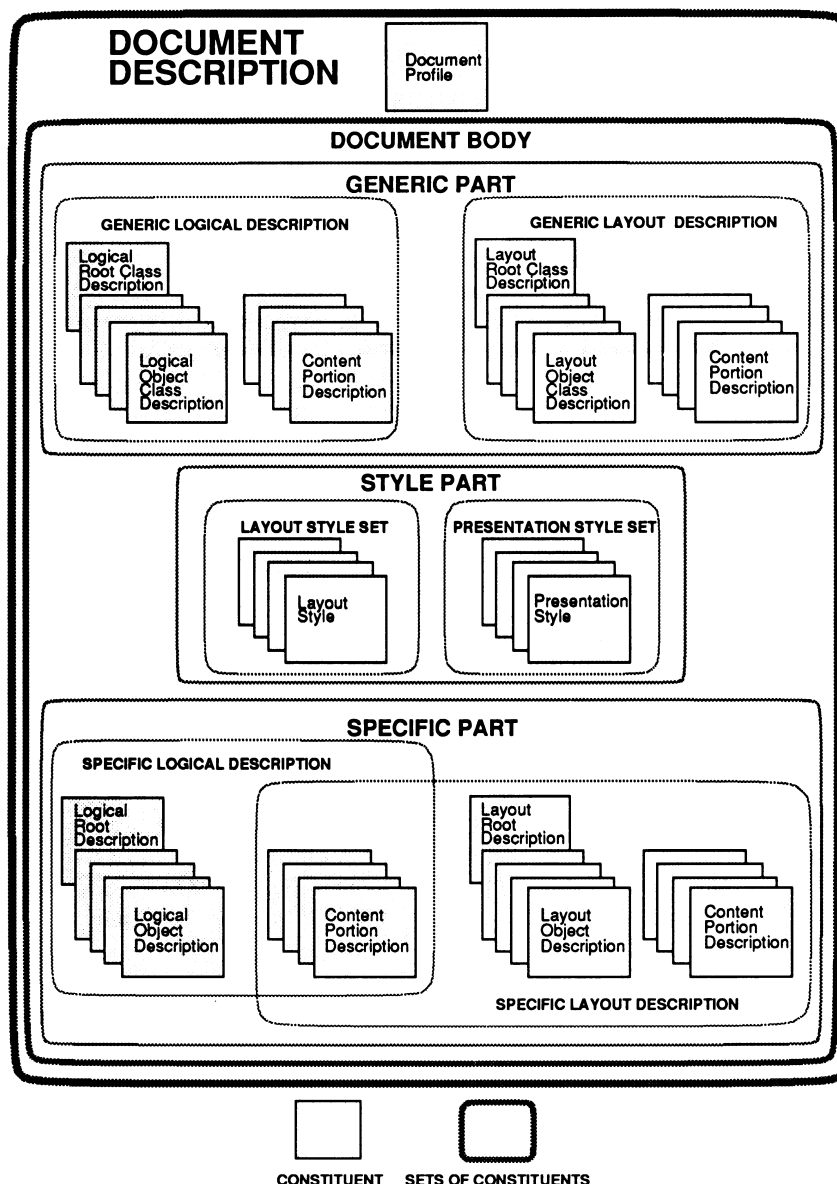


FIGURE 1. The descriptive representation of a document.

The figure uses the terms *specific* and *generic* to qualify layout and logical structures (e.g. the Generic Layout Structure or the Specific Logical Structure). Additionally there is a style part (a set of objects called *layout style set*) which contains information for logical-layout-relationships such as starting a chapter on a new page, or content-presentation features responsible for altering, for example, line spacings (these are called the *presentation style set*).

ODA documents may be in formatted, processable and formatted-processable form. The first form allows no processing, so that a document can be presented (printed or displayed) as intended by its creator; the second form allows a recipient to do whatever he/she wishes to a document; the final form satisfies both previous forms. In formatted form, for instance, an ODA document does not contain a specific or generic logical structure but must contain a specific layout structure.

In processable form, on the other hand, the specific layout structure will be missing.

1.3. Information Modelling by Composition Language (IMCL)

Information Modelling by Composition (IMC) is a technique for describing information structures and operations on those structures. It has an associated language (IMCL) which is based on well defined syntactical and semantic constructs [4]. The language is well suited to expressing information structures which are complex in nature, in a rigorous and unambiguous fashion. One of its principal strengths is that it was designed for a well delimited domain of application, hence it does not try to incorporate an over-ambitious ability to do *everything*, but rather applies itself to succinctly and accurately specifying what it was designed to do.

IMCL is a first-order language for modelling information objects and their transformations. It is not a system specification language although it can be used in combination with such a language, for example, as an annotation language for Petri Nets as demonstrated by Richter and Durchholz with an earlier version of IMCL [7]. So, IMCL does not address specification of protocols and systems in general as other formal description techniques do, e.g. SDL, LOTOS, Estelle, VDM, CSP, etc., an *up-to-date* review of these specification techniques (and others) can be found in [8].

To date IMCL has an important field of application namely in the area of information systems standardization and in particular to ISO 8613. However within and around that scope many independent projects have been initiated where IMCL has played an integral role. These projects will be surveyed in general later in this paper.

1.4. Applying IMCL to ODA

In 1985 the ISO initiated formal specification work on one of its Study Committee 18 projects on *Text and Office Systems*² namely *Document Structures*. It was decided a special working group (SWG) be formed, the purpose of which was to carry out a feasibility study on formalizing one of its standards—ISO 8613. A group report was produced, several formal approaches were considered and it was concluded that a formal specification of the standard would be very useful for verification of the standard and for future implementors. The report also recommended that initial efforts should adopt an approach that would be relatively easy to understand, noticeably close to the standards textual description and not necessarily dependent on any particular methodology. The SWG produced a first draft of the specification, which although under-developed, did give a framework for the full formal specification of the main part of the standard, and also provided detailed terminology and notation sections. This formal work was subsequently endorsed by working group 3 (WG3) which is responsible for the ODA standard production (see Section 3 for details).

In Germany the Gesellschaft für Mathematik und Datenverarbeitung (GMD—the German National Research Centre for Computer Science) was responsible for the development of the formal methodology (IMC), description technique and language (IMCL). In addition GMD has carried out studies on the application of IMC/IMCL to describe test cases for testing conformance to the Standard and submitted them to ISO for consideration.

In the UK the National Computing Centre (NCC) and in Germany DANET have adopted the formal specifications of ISO 8613 as a basis for conformance test tools which have been developed to test implementations of the standard.

In Canada Carleton University carried out a translation of a subset of IMCL into Prolog for its Government's Department of Communications (DOC), thus giving executable semantics of the language for the first time. The project resulted in the creation of a specification of the standard being implemented using this IMCL-to-Prolog translation process, giving an executable and traceable version of its formal specification.

2. IMCL TUTORIAL

2.1. Introduction

IMCL is based on the observation, that even very complex structures, e.g. structures in data bases or structures that model certain objects in the 'real world', can usually be considered composite in nature, but ultimately being constructed from very simple basic elements.

IMCL semantics are given by specifying the meaning of each expression (formula or term) in the language. As usual, each entity of a given application is assigned to a particular constant, a predicate on entities to each predicate symbol and an operator on entities to each operator symbol. Based on these assignments one finally specifies how the meaning of any syntactically correct expression is derived. There are several semantics possible for IMCL, depending on how such specifications are carried out. Any application may choose its own semantics according to the concepts relevant to its problem area. However, IMCL comes with a body of common semantic rules called the *basic semantic specifications*, independent of the particular application and valid for all applications. Semantics beyond the basic semantic specifications are known as *supplementary semantic specifications*. Some predicates and operators of the basic semantic specifications are given in Sections 2.4 and 2.4.

Provision has been made in IMCL to define new predicates and operators as and when needed. In fact, the Formal Specifications of ODA (FODA) is in essence a large number of such *application oriented* definitions (supplementary from an IMCL point of view), which extend IMCL's basic semantics for the *world* of ODA.

IMCL uses only a few basic element types, called *atoms*. The actual choice of these atoms depends on the application, i.e. on the nature of the objects which are to be modelled by IMCL. There are only three different composition principles in IMCL (called *collection*, *catenation* and *nomination*) for building compound objects using atoms. Although this number seems rather small it turns out to be sufficient for modelling many different kinds of structures, in particular those found in ODA documents. The IMCL objects and the composition principles will be explained in Section 2.2.

In many applications it is not sufficient to provide a formal description for concrete structures only, but it is also necessary to formally describe general 'rules' on

² This includes *Open Systems*.

how such structures are composed. For instance, in an application describing data base structures it is not sufficient to say: 'The date field in a data base entry consists of 1990-12-24'. Rather, one wants to express the general rule(s) for such an entry, e.g. 'The date field is a sequence of three numbers, the first giving the year, the second giving the month (1 ... 12) and the third giving the day of a month (1 ... 31 or 1 ... 30 or 1 ... 29 or 1 ... 28)'. To achieve the *formal expression* of such rules IMCL provides a number of additional features such as first-order predicate logic and operators.

Predicate logic is usually applied to express certain 'facts' about objects in a specific IMCL application. There are a few pre-defined or *basic* predicates in IMCL, but using these *basic* predicates additional ones can be defined for each application. In fact, using IMCL for formally describing objects or structures in a certain application will often primarily consist of the specification of application dependent predicates. The basic predicates of IMCL will be discussed in Section 2.3.

IMCL comprises a number of operators which can be applied to objects, e.g. arithmetical operations or set theoretical operations. Importantly, several of the operators provide for the addressing or selection of substructures within composite structures. In addition, application dependent operators can be built on top of the basic ones which are already predefined in IMCL. The basic operators will be described in Section 2.4.

Since a fully detailed description is not possible in this paper we shall concentrate on those aspects which are important for the application of IMCL to FODA. A complete description of IMCL can be found in [4].

2.2. IMCL objects and composition principles

The formal objects in IMCL which are used for modelling certain structures in specific applications are called *constructs*. There are two types of constructs, i.e. *atoms*, which are not composed of other constructs, and *composite constructs*, which are composed of atoms and/or other composite constructs. The objects which are considered as atomic (i.e. *atoms*) are dependent on the application. For instance, in FODA the following objects are IMCL atoms:

- Numbers, since the values of some attributes in an ODA document are numbers.
- Characters from some defined character set or sets, because obviously characters play a central role in a document processing standard.
- Attribute and parameter names, since a name of an attribute or parameter in an ODA document can be considered as a terminal symbol, i.e. there is no substructure in such a name from ODA's point of view.

Except for the numbers, atoms are enclosed in quotes in IMCL notation. For example,
3 '3' 'object type'

are three atoms, the first one being the number 3, the second one the character 3 and the third one being an atom (a character string) which happens to be the name of an ODA attribute in the FODA application.

The first construction principle for composite constructs in IMCL is called a *collection*. A collection is a set of constructs which may be atoms or composite constructs. The term collection is equivalent to the mathematical concept of the set, except that the components of a collection must be constructs (such a restriction does not apply in set theory). In IMCL, collections are denoted by listing their components separated by semi-colons and enclosing this list in square brackets. For example,

[1; '1'; 'abc'] [1; ['1'; 'abc']] []

are three collections. The first collection has three components, i.e. the atoms 1, '1' and 'abc'. The second collection has two components, namely the atom 1 and a composite component which is itself a collection. The third collection is the empty collection, i.e. a collection without components.

The second construction principle for composite constructs is called a *catenation*. A catenation is a sequence of constructs which may be atoms or composite constructs. Again, as with sets, there is a mathematical equivalent of the catenation, the concept of the sequence, however the components of a catenation must be constructs. Catenations are denoted by listing their components separated by arrows and enclosing this list in square brackets. For example,

[→ 1 → '1' → 'abc' →]
[→ '1' → ['1'; 'abc'] →] [→]

are three catenations. The first catenation has three components, the first is the atom 1, the second the character '1' and the third is the atom 'abc'. The second catenation has two components, the first being the atom 1 and the second a composite component, i.e. a collection. The third catenation is the empty catenation, i.e. a catenation without components.

The third, and last, construction principle for composite constructs is called a *nomination*. For the FODA application it can be regarded as a set where each element of the set is a (name,construct)-pair, in the sense of a *mathematical mapping*. The name-part for each such pair is an atom and the construct-part may be an atom or a composite construct. Nominations are denoted by listing the (name,construct)-pairs separated by semi-colons and enclosing this list in square brackets. The name-part and the construct-part for each pair are separated by a colon. For example,

['abc' : 7; 'cde' : 'xyz'; 'def' : [→ 1 → 4 → 2 →]]

is a nomination with three (name,construct)-pairs. For the first pair the name-part is 'abc' and the construct-part is the number 7 (an atom). For the second pair the name-part is the 'cde' and the construct-part is the 'xyz'.

For the third pair the name-part is 'def' and the construct-part is a composite construct, namely a catenation consisting of three numbers (atoms).

As another example, consider the following composite construct:

```
['object type' : 'composite logical object';
 'object identifier' : [1 → 3 → 7 →];
 'subordinates' : [→ 0 → 2 →];
 'protection' : 'unprotected']
```

This composite construct is a nomination with four (name,construct)-pairs. For the names 'object type' and 'protection' the associated constructs are atoms, for the names 'object identifier' and 'subordinates' the associated constructs are catenations whose elements are numbers.

This example is an IMCL specification for a so-called *composite logical object* in an ODA document. In fact, it can be shown that any given ODA document can be described in IMCL using only collections, catenations and nominations, since all structures which may appear in an ODA document can be mapped using these three IMCL construction principles.

As mentioned in the introduction, it is usually not sufficient to formally describe specific structures only but rather it is more desirable to specify general rules on structures in a formal manner. For this purpose it is often necessary to 'point' to a certain component within a composite construct or to 'extract' a certain component or set of components from a composite construct. To this end, the unique concepts of *spot* and *spotset* (a set of spots) are defined in IMCL. To introduce these concepts consider the following example of a nomination t :

$$t = ['ab' : 1; 'cd' : 1; 'ef' : [→ 'a' → 'b' → 'a' →]]$$

The number 1 appears at two places (spots) in the nomination, namely at the construct-part of both the components 'ab':1 and 'cd':1. Similarly, the atom 'a' appears at the first position and at the last position of the catenation [→ 'a' → 'b' → 'a' →]. In other words, the spot where a certain component appears within a composite construct is often not identified by the component itself but additionally, 'routing information' might be necessary.

Spots within a composite construct are usually identified by selection criteria based on operators such as those described in Section 2.4. For instance

$$\hat{t}. 'ab'$$

addresses the spot 'ab':1 within t . (This notation, in particular the IMCL operators $\hat{}$ and \cdot , are described in Section 2.4).

In many cases a selection criterion will not give a single spot within a composite construct, but often a set of spots (spotset). For instance, selecting all (name,construct)-pairs in the previous example whose construct is the integer 1, would return the spots 'ab':1 and 'cd':1,

i.e. a spotset. In fact, the IMCL operators are in general defined for spotsets, not for spots, i.e. the argument for an operator may be a spotset, rather than a spot, and an operator may create a spotset, rather than a spot. The (exceptional) case of a single spot is dealt with by the concept of the so-called *singleton spotset*: a spotset containing only one element. Dealing with single spots always in the form of a singleton spotset avoids the definition of similar operators both for single spots and for spotsets.

2.3. Basic IMCL predicates

First-order predicate logic forms an integral part of IMCL and several unary and binary predicates are defined. A predicate is either True or False. These are examples of some basic IMCL predicates:

IsAtom(t)	is true, iff (if and only if) t is an atom
IsNumber(t)	is true, iff t is a number
IsCol(t)	is true, iff t is a collection
IsCat(t)	is true, iff t is a catenation
IsNom(t)	is true, iff t is a nomination
IsSpotset(t)	is true, iff t is a spotset
$t_1 = t_2$	is true, iff the entities t_1 and t_2 are identical
$t_1 \in t_2$	is true, iff t_2 is a collection and t_1 is a component t_2
$t \in t_2$	is true, iff t_1 is a singleton spotset and a subset of the spotset t_2
$t_1 \subseteq t_2$	is true, iff t_1 is a subset of t_2 and t_1 and t_2 are collections or spotsets, respectively

Furthermore, since IMCL is based on predicate logic the usual logical quantifiers \forall (for all) and \exists (exists) as well as the logical connectives *not*, *and*, *or*, *iff* (if and only if) and *impl* (implies) are part of the language. For successive quantification the usual notational simplification is used, e.g.

$\forall x(x \in m \text{ and } formula)$
may be abbreviated to
 $\forall x \in m(formula)$

Based on these (and a few more) basic predicates further predicates can be defined according to the specific application, as the following simple example shows:

$\forall t$
(IsPairOfNumbers(t) *iff*
 $\exists l, r$
($t = [→ l → r →]$ *and*
IsNumber(l) *and*
IsNumber(r)))

This example is the specification of a new predicate called IsPairOfNumbers(t), which is True if and only if t is a catenation of two components l and r and both of these components are numbers. Further examples will be shown in Section 3.

2.4. Basic IMCL operators

IMCL includes several basic operators. For instance, when entities are numbers the usual arithmetical operators $+$, $-$, $*$, $/$ (addition, subtraction, multiplication, division) are defined. For collections and spotsets the set theoretical operations \cup , \cap , \setminus (union, intersection, set difference) are defined. Some additional operators are, for example:

- $t_1 // t_2$ If t_1 and t_2 are catenations, $t_1 // t_2$ is the catenation obtained by concatenating t_1 and t_2 .
- \hat{t} If t denotes a construct, \hat{t} denotes the singleton spotset containing t . Loosely speaking, this operator transforms a construct into a singleton spotset and is usually applied before applying further operations such as HEAD, N or C, which are defined only for spotsets.
- $t.$ If t denotes a spotset containing no atom spots (spots which are atoms), then $t.$ (read ' t next inwards') denotes the set of all spots which are immediately inward of the spots of the spotset t . Loosely speaking, this operator 'unpacks' all spots within t and generates a new spotset whose elements are all those returned after this 'unpacking' procedure. This operator is usually applied in the context of decomposing composite structures into their substructures.
- NAMS t If t denotes a nomination, NAMS t (read 'nameset of t ') denotes a collection whose components are the names of the (name,construct)-pairs of t .
- N t If t denotes a singleton spotset immediately inward a nomination spot, N t denotes the name at the spot given by t .
- C t If t denotes a singleton spotset, C t denotes the component construct at the spot given by t .
- CARD t If t denotes a collection or a spotset, CARD t (read 'cardinality of t ') denotes the number of elements within t .
- HEAD t If t is a catenation, HEAD t returns the first component of the catenation.
- $t.$ 'xyz' If t denotes a singleton spotset of a nomination spot, $t.$ 'xyz' denotes the singleton spotset with (name,component)-pair for which the name-part is 'xyz'. Loosely speaking, this notation is used to address a (name,component)-pair for a particular name. Note, that $C^{\hat{t}}.$ 'xyz' denotes the component-part for the name 'xyz'.

The following examples demonstrate the use of these operators.

Let t denote the following catenation:

$[\rightarrow 1 \rightarrow '1' \rightarrow 'abc' \rightarrow]$

Then the following relation holds:

- HEAD(t) = 1, since the first component of the catenation is the number 1.

Now let t denote the following nomination:

$['abc' : 7; 'cde' : 'xyz'; 'def' : [\rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow]]$

Then the following relations hold:

- NAMS(t) = $['abc'; 'cde'; 'def']$, since these are obviously the names appearing in the nomination.
- CARD $\hat{t}.$ = 3, since there are three components within the nomination.
- $C^{\hat{t}}.$ 'abc' = 7, since the construct-part of the (name,construct)-pair with the name atom 'abc' is the number 7.
- $\forall x \in \hat{t}. (N x = 'cde' \text{ impl } C x = 'xyz')$, because the name-part 'cde' has a construct-part equal to 'xyz'.

3. THE FORMAL SPECIFICATIONS OF ODA

3.1. ODA in terms of IMCL

The Formal Specifications of ODA (FODA) is currently an ISO/IEC JTC1/SC 18/WG3 project. FODA at present pursues a *declarative approach* to information structure description by specifying the possible structures that conform to the standard rather than how the structures are obtained as the result of some output process. The formal specification is concerned with the descriptive representations of documents since constituents and their attributes are the basis of the interchange stream and their make-up and relationships the subject of ODA conformance requirements.

FODA originally addressed the specification of *The Document Structures* (ISO 8613-2), since that part plays a central role in supporting the other parts of the standard. The formal specifications of the document structures have been published as Part 10 of ISO 8613 in 1991. The formal specifications of the document profile and the three content architectures of ODA (character text, raster graphics and geometric graphics) started afterwards and have been progressed as Addenda to ISO 8613-10. In the meantime, all these Addenda have passed their final ballots and reached International Standard status. (A further Addendum on the formal specification of the defaulting mechanism for defaultable attributes reached IS status in the autumn of 1992.) Therefore, the FODA project has now reached its initial goal, i.e. to address all Parts of ISO 8613, except for Part 1 which is of an introductory nature, and Part 5 which defines the binary interchange format for ODA documents. These two Parts are not an issue for the formal specification. Of course, the formal specifications will be extended whenever extensions are made to the ODA standard itself. In late 1992 a republication of the ODA standard is expected, including a considerable

number of modifications and extensions which were added to the standard since its first publication in 1989. Accordingly, ISO 8613-10 will be republished in early 1993, reflecting the natural language version of the standard.

A major criterion for the application of a formal description technique in addition to a natural language text was for the provision of a single interpretation, during the *design, implementation and application phases* of the standard. This avoids different views on information structures, functions and processes by ruling out ambiguity in the interpretation of the natural language text. In fact, the FODA project identified a considerable number of ambiguous and even contradictory specifications in the natural language text of the ODA standard which were then corrected accordingly.

By considering FODA, readers gain clearer insights and can reason more thoroughly on the standard than by relying on the natural language text alone. This enhances the probability of producing fully conformant implementations. Due to the mathematical basis of the notation being elementary *set theory* and *predicate logic* most software practitioners with a basic knowledge of computing science should be able to understand the specifications.

FODA is rigorous. This rigor is gained by using the IMCL technique whose foundations stem from mathematics and logic. The technique extends the IMC mathematical and logic basis by providing structures and operations.

The style of the specification is mixed; a formal IMCL notation with semi-formal English text clauses (which do not belong to the formalized definitions). However these clauses do provide an explicit link by referencing between the formal specification and those natural language clauses in the standard that are under definition. This also gives some indirect confirmation about the completeness of the specification.

The formal specification of ODA is a single formula specified in IMCL using *first-order predicate logic*. The formula consists of sub-formulae which are joined by the connective *and*:

$$\text{formula}_1 \text{ and formula}_2 \text{ and } \dots \text{ formula}_n$$

These formulae are also called *definitions*. They define either concepts used in the natural English description of the standard, or subsidiary predicates and functions, which are introduced to enhance readability.

The definitions are grouped into several sets. Slightly simplifying, the first set of definitions specifies the overall structure of ODA documents. For instance, these definitions specify the rules for which constituents are required or permitted in different classes of ODA documents. They specify structural requirements (e.g. the objects in the specific structures must form a tree structure) and cross-relations between attributes and their values in different constituents (e.g. if an attribute in one constituent has a particular value, the value of an attribute

in another constituent must satisfy some specified constraints).

The second group of definitions specifies the structure of the individual constituents of an ODA document such as composite logical objects, pages or blocks. In particular, these definitions specify the required or permitted attributes of a particular constituent and constraints on their attribute values.

The third group of definitions specifies the permitted values or value ranges of attributes described in ISO 8613-2. Since many attribute values are substructured into parameters, sub-parameters or even sub-sub-parameters, often with cross-relations between the values of these parameters and sub-parameters, these definitions are sometimes rather complex.

The Addenda to ISO 8613-10 mentioned above contain further sets of definitions, namely those specifying attributes and attribute values of the document profile (described in ISO 8613-4) and of the different content architectures (described in ISO 8613-6, -7, -8). The last set of definitions in the Addendum on the formal specification of the defaulting mechanism specifies the rules for deriving the value for each defaultable attribute.

Currently ISO 8613-10 and its Addenda contains over 600 so-called definitions (or formulae), each one defining one or more of the technical specifications of the ODA standard in IMCL notation, which sums up to well over 300 printed pages of text and formulae.

3.2. Examples

A simple example shall show how IMCL can be used to specify 'rules' about structures in ODA documents. Consider a so-called *composite logical object* which might look as follows:

```
['object type' : 'composite logical object';
 'object identifier' : [  $\rightarrow 3 \rightarrow 7 \rightarrow$  ];
 'subordinates' : [  $\rightarrow 0 \rightarrow 2 \rightarrow$  ];
 'protection' : 'unprotected']
```

The composite logical object is modelled in IMCL as a nomination where the names are 'object type', 'object identifier', 'subordinates' and 'protection'. For the name 'object type' the associated value is 'composite logical object', for the name 'object identifier' the value is a catenation of components 3 and 7, etc.

Instead of specifying a specific composite logical object we now want to specify the rules which hold for such an object. According to the ODA standard, these rules are:

4. A composite logical object is a set of attributes, each attribute having a name and an associated value.
5. The attribute names are 'object type', 'object identifier', 'subordinates' and 'protection'.
6. For the attribute name 'object type' the value is 'composite logical object'.
7. For the attribute name 'object identifier' the value is a logical object identifier, which is a sequence of numbers where the first number is 3.

8. For the attribute name 'subordinates' the value is a sequence of numbers.
9. For the attribute name 'protection' the value is either 'protected' or 'unprotected'.

The actual rules as specified in the ODA standard are more complicated but these six rules shall be sufficient for the example.

Modelling these rules in IMCL requires the following steps:

1. We define a predicate 'IsCompositeLogicalObject(*obj*)' which has the value True if and only if *obj* satisfies these rules.
2. In IMCL a set of attributes is modelled by a nomination, i.e. as a first condition *obj* must be a nomination.
3. The name set of this nomination must be specified according to the ODA specifications, i.e. a predicate for specifying the name set of the nomination is needed.
4. For each of the four attribute names a predicate for specifying its value range must be given.

This can be accomplished by defining the following predicate:

```

∀ obj
IsCompositeLogicalObject(obj) iff
(IsNom(obj) and
  NAMS(obj) = ['object type'; 'object identifier';
               'subordinates'; 'protection'] and
  ∀ a ∈ obj.
  ((N a = 'object type' impl
    C a = 'composite logical object') and
   (N a = 'object identifier' impl
    IsLogicalObjectId(C a) and
   (N a = 'subordinates' impl
    IsSequenceOfNumbers(C a) and
   (N a = 'protection' impl
    C a ∈ ['protected'; 'unprotected'])))

```

As can be seen, the specification of the value ranges for the attributes 'object type' and 'subordinates' has been 'factorized' into two further predicates called 'IsLogicalObjectId' and 'IsSequenceOfNumbers'. This kind of factorization approach is common practice in IMCL and provides for a clearer, cleaner and more structured specification of objects in a given application.

These two predicates could be defined as follows:

```

∀ v
IsSequenceOfNumbers(v) iff
(IsCat(v) and
  ∀ a ∈ v. (IsNumber(C a)))

∀ v
IsLogicalObjectId(v) iff
(IsSequenceOfNumbers(v) and HEAD(v) = 3)

```

A sequence of numbers is defined as a catenation whose components are numbers, and a logical object identifier is defined as a sequence of numbers with the first

component being 3, as is specified in the ODA standard. Figure 2 shows another example of a FODA definition taken from ISO 8613-10. This definition specifies the value of the ODA attribute 'separation' in IMCL. The semiformal description associated with this definition reads as follows:

'The value of the attribute "separation" is a nomination, i.e. a set of pairs (*name*, *value*). The *name* is an element of the set ["leading edge"; "trailing edge"; "centre separation"]; the *value* is a non-negative integer. The parameters are independently defaultable.'

In this case, an attribute value has a substructure, i.e. the value of the attribute 'separation' is structured into the three parameters 'leading edge', 'trailing edge' and 'centre separation'. In IMCL such a substructure is modelled as a nomination.

Furthermore, the parameter values are defaultable; this is modelled in IMCL by using the special predicate 'IsPlaceholder'.

Additionally, it can be seen that each definition has a unique number in FODA. The definition of 'IsSeparationValue' has the number 2.117 and references other definitions such as 'IsNeNom' with the number 1.2 and 'IsNnInt' with the number 1.4.

The previous examples have been used for clarity. The actual FODA definitions can be more complex as can be seen in the following definition which also shows the equivalent Prolog and C++ translations respectively (see Figures 3–5).

Here it is shown how some quite complex inter-relationships of ODA can be handled by IMCL in an 'unambiguous' way. An English translation of an IMCL definition specifying such a concept is given for ease of understanding (see below, also see Figure 1 for details of constituent terminology). The example in Figure 3 shows the possible tests that can be performed on several independent ODA objects which interrelate to form a possible ODA document.

The IMCL definition could be read as:

'For all entities called *doc* this entity is a document description if and only if there exists an entity called *prof* which is a valid document profile as stated in ISO 8613-2 (this is formally defined in formula 2.20) and that the *doc* could be either a collection of a single *prof* entity or if the *doc* is not equal to a collection of a single *prof* entity it is an entity which is processable (this is formally defined in formula 2.4), formatted processable (this is formally defined in formula 2.5) or formatted (this is formally defined in formula 2.6) depending on the value of the document profile attribute "document architecture class", and with a "resource document" (which is not physically part of the document in question and is thus external in nature) specified in the profile if any "resource" constituent is specified in the document.'

3.3. Survey of FODA applications

As previously stated, ISO used IMCL to specify the *Document Structures* and the *ODA Defaulting Mechanism* (ISO 8613-2), the *Document Profile* (ISO 8613-4) and the three *content architectures* (ISO 8613-6, -7 and -8, respectively) of the ODA standard. In addition, the *Formal Specifications* of ODA have been used in the development of conformance testing software. Three

```

1   $\forall v$ 
2  (0 IsSeparation Value( $v$ ) iff
3    IsNeNom1.2( $v$ ) and
4    NAMS1.15( $v$ ) = ['leading edge'; 'trailing edge'; 'centre separation'] and
5     $\forall a \in \hat{v}$ .
6    (1 IsPlaceholder1.16( $C a$ ) or IsNnInt1.4( $C a$ )1)0)

```

FIGURE 2. Example of a FODA definition.

```

1   $\forall doc$ 
2  (0 IsDocumentDescription( $doc$ ) iff
3     $\exists prof$ 
4    (1 IsDocumentProfilePart22.20( $prof$ ) and
5    (2  $doc = [prof]$  or IsProcessable2.4( $doc$ ) or
6    IsFormattedProcessable2.5( $doc$ ) or IsFormatted2.6( $doc$ )2) and
7     $doc \neq [prof]$  impl
8    (3 ( $C \hat{prof}$ . 'document architecture class' = 'processable' iff
9    IsProcessable2.4( $doc$ )4) and
10   (5  $C \hat{prof}$ . 'document architecture class' = 'formatted processable' iff
11   IsFormattedProcessable2.5( $doc$ )5) and
12   (6  $C \hat{prof}$ . 'document architecture class' = 'formatted' iff
13   isFormatted2.6( $doc$ )6) and
14    $\forall cst \in doc$ 
15   (7 'resource'  $\in$  NAMS1.15( $cst$ ) impl 'resource document'  $\in$  NAMS1.15( $prof$ )7)3)1)0)

```

FIGURE 3. IMCL version of the FODA definition IsDocumentDescription.

```

IsDocumentDescription(DOC) iff                                % 2
exists(PROF,(                                                % 3
  isDocumentProfilePart2(PROF) and                            % 4
  (DOC = s@[PROF] or isProcessable(DOC) or                    % 5
  isFormattedProcessable(DOC) or isFormatted(DOC)) and        % 6
  (DOC \ = s@[PROF]; impl (                                     % 7
    >>(c ('PROF.!'document architecture class')) = 'processable') iff % 8
    isProcessable(DOC) and                                     % 9
    >>(c ('PROF.!'document architecture class')) = 'formatted processable') iff % 10
    isFormattedProcessable(DOC) and                            % 11
    >>(c ('PROF.!'document architecture class')) = 'formatted') iff % 12
    isFormatted(DOC) and                                       % 13
    forall(CST e DOC,(                                         % 14
      >>('resource' e nams(CST)) impl >>('resource document' e nams(PROF)) % 15
    ))
  )
)).

```

FIGURE 4. Prolog translation of the FODA IsDocumentDescription.

such software projects based on and using ISO 8613-10 are briefly described below.

Around early 1988 a project funded by the Canadian Department of Communications (DOC) was carried out at Carleton University, Ottawa [6]. The project resulted in a subset of IMCL (predefined functions and predicates, data types and operators) being implemented in Prolog. These were then combined with one-to-one FODA definition translations in Prolog—since both FODA and Prolog are based on predicate calculus this was an easily performed task, and has the possibility of being mechanically performed in the future. What resulted was basically a prototype FODA Conformance Analyzer (FCA). The input to this prototype analyzer is a Prolog definition of an ODA document (at present the FCA cannot handle the binary encoding of ODA documents

directly but rather clear text representations of the encoding). The FCA checks for consistency of the input ODA document against its FODA-to-Prolog translated definitions and reports any errors, if present. The only drawback of this system is the large amount of processing time required to analyze a single average ODA document containing a large number of constituents.

During late 1989 the NCC completed the first 'phase of its ODA conformance testing system—TODAC (Testing ODA Conformance) [3] a collaborative project between NCC and the Canadian DOC and from 1990 to 1992 produced a *comprehensive* ODA Implementation testing system ODACT (ODA implementation Conformance Tester) with INTAP (the Japanese Interoperability Technology Association for Information Processing) as collaborators. A component-module in

```

boolean  Foda::IsDocumentDescription_2_3(Document* doc)
{
  INITIALISE( IsDocumentDescription_2_3, 2, 3, doc )
  OB(0){
  LINE(3)  THERE EXISTS_prof(doc)
  OB(1){
  LINE(4)   IsDocumentProfilePart2_2_1(prof) AND
  OB(2){
  LINE(5)   (doc->No_Of_Constitits() == doc->No_Of_Constitits(CO_DOCUMENT_PROFILE) == 1) OR
  LINE(5)   IsProcessable_2_4(doc) OR
  LINE(6)   IsFormattedProcessable_2_5(doc) OR
  LINE(6)   IsFormatted_2_6(doc) CB(2)} AND
  LINE(7)   (doc->No_Of_Constitits() > doc->No_Of_Constitits(CO_DOCUMENT_PROFILE)) IMPL
  OB(3){ OB(4)
  LINE(8)   (E_VALUE_OF(prof, DOCUMENT_ARCHITECTURE_CLASS) == PROCESSABLE) IFF
  LINE(9)   IsProcessable_2_4(doc) END_IFF CB(4)} AND
  OB(5){
  LINE(10)  (E_VALUE_OF(prof, DOCUMENT_ARCHITECTURE_CLASS) == FORMATTED_PROCESSABLE) IFF
  LINE(11)  IsFormattedProcessable_2_5(doc) END_IFF CB(5)} AND
  OB(6){
  LINE(12)  (E_VALUE_OF(prof, DOCUMENT_ARCHITECTURE_CLASS) == FORMATTED) IFF
  LINE(13)  IsFormatted_2_6(doc) END_IFF CB(6)} AND
  LINE(14)  FOR ALL_CS(cst, doc)
  OB(7){
  LINE(15)  E_NAMS(cst, RESOURCE) IMPL
  LINE(15)  E_NAMS(prof, RESOURCE_DOCUMENT) END_IMPL
  CB(7)} END_FOR CB(3)} END_IMPL CB(1)} CB(0)}
  REPORT( IsDocumentDescription_2_3, 2, 3, doc )}

```

FIGURE 5. C++ translation of the FODA definition IsDocumentDescription.

this system is the ODA *Structure Analyser*, based on the formal specifications of the ISO 8613-2. However, unlike the Canadian FCA the NCC's chosen implementation language for both collaborations was C++. This choice of language significantly decreased the processing time of analyzing the ODA document structure and hence made the *Structure Analyser* more viable as a component-module in the complete conformance testing system in contrast with a prospective prolog *FCA-like* component-module. In addition the *Structure Analyser* was implemented to analyze documents which had been automatically decoded and checked from their ODIF encodings (ASN.1 data streams) and which had been represented in the TODAC/ODACT systems own internal structure representation of that document.

Furthermore, a conformance testing project has been carried out by DANET in Germany recently. Their approach is rather similar to the one taken by NCC. The main difference seems to be that they used C as the implementation language (private communication to the authors, no written report is available yet).

4. CONCLUSIONS

There exists a need for a formal description technique which has a rigorous mathematical basis for specifying information structures (in particular when creating standards for document processing systems). This applies especially to complex structures such as can be found in ODA documents and is demonstrated in natural lan-

guage specifications, where due to lack of precision, ambiguities and even inconsistencies can be permanent features.

It has proved worthwhile, even although a natural English text following ISO regulations is still the final form of a standard, to make the effort at formally describing it. This has *payoffs* which improve the standard and can provide definitive reference documents (as in ODA) to answer most of the questions arising from attempted interpretations of the English text version. A formal specification in addition to creating a rigorous foundation for a standard can also be used in the development of useful *standards conformance test tools* amongst other things, as described in this paper.

For a contemporary review of different techniques, methodologies, case studies and best practice in the area of formal methods in standardization see [8].

ACKNOWLEDGEMENTS

The authors would like to thank Gernot Richter of GMD (Bonn, Germany) for his invaluable help, advice and comments on this paper. Thanks are also extended to Gerald Karam of Carleton University (Ottawa, Canada) for generously supplying code of the FODA definition used in Figure 4.

REFERENCES

- [1] W. Appelt, *Document Architecture in Open Systems: The ODA Standard*. Springer Verlag, New York (1991).

- [2] W. Appelt, R. Carr and G. Richter, The formal specification of document structures of the ODA standard. In: *Proceedings EP88 Conference*. Electronic Publishing (1988).
- [3] R. Carr and F. Dawson, Conformance testing of Office Document Architecture (ODA). *Computer Communications*, pp. 00–00 (1989).
- [4] R. Durchholz and G. Richter, *Compositional Data Objects. The IMC/IMCL Reference Manual*. John Wiley, New York (1992).
- [5] ISO 8613, Office Document Architecture (ODA) and Interchange Format International Standard ISO 8613. ISO, Geneva (1989). To be republished as *Open Document Architecture (ODA) and Interchange Format International Standard ISO 8613* in 1993
- [6] G. Karam, *The FODA Conformance Analyzer: Prototyping with Prolog*. Technical Report, Carleton University, Ottawa (1988).
- [7] G. Richter and R. Durchholz, IML-inscribed high-level Petri Nets. In: *Information Systems Design Methodologies: A Comparative Review*. T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart (eds.), pp. 333–368. North-Holland, Amsterdam (1982).
- [8] C. L. N. Ruggles (ed.), *Formal Methods in Standards: A report from the BCS Working Group*. Springer Verlag, New York (1990).