Date: 1999-01-19

Number of pages: 53

ISO/TC 211 WG4

Geospatial services

Encoding
Working Draft Version 4.0

Source: David Skogan, Project leader PT18

SINTEF Telecom and Informatics
P.O. Box 124 Blindern
N-0314 OSLO
NORWAY

Telephone: +47 22 06 78 95
Telefax: +47 22 06 73 50

Email: David.Skogan@informatics.sintef.no

Action: Request for comments and indication of whether this document is ready for CD. Please respond before the 29th of January 1999!

Distribution: PT18 and WG4 members

ISO/TC 211 WG4 convenor

Morten Borrebæk
Norwegian Mapping Authority
3500 Hønefoss
Norway

Telephone: +4732118100
Telefax: +4732118101
Email: morten.borrebaek@it.statkart.no

ISO/TC 211/WG 4 N 109

Date: 1998-09-18

/WD

ISO/TC 211/SC /WG 4

Secretariat: NSF

Geographic information — Part 18: Encoding

Information géographique — Partie 18: Encodage

Document type: International Standard
Document subtype: Not applicable
Document stage: (20) Working draft
Document language: E

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

International Standard ISO 15046-18, *Geographic information — Part 18: Encoding*, was prepared by Technical Committee ISO/TC 211 *Geographic information/Geomatics*.

This standard is part 18 of ISO 15046, a multi-part International Standard under the general title *Geographic information* consisting of the following parts:

- *Part 1: Reference model*
- *Part 2: Overview*
- *Part 3: Conceptual schema language*
- *Part 4: Terminology*
- *Part 5: Conformance and testing*
- *Part 6: Profiles*
- *Part 7: Spatial schema*
- *Part 8: Temporal schema*
- *Part 9: Rules for application schema*
- *Part 10: Feature cataloguing methodology*
- *Part 11: Spatial referencing by coordinates*
- *Part 12: Spatial referencing by geographic identifiers*
- *Part 13: Quality principles*
- *Part 14: Quality evaluation procedures*
- *Part 15: Metadata*
- *Part 16: Positioning services*
- *Part 17: Portrayal*
- *Part 18: Encoding*
- *Part 19: Services*

This document contains 5 annexes. Annex A and B are normative, and annexes C, D and E are informative.

This working draft International Standard is being circulated to the Technical Committee ISO/TC 211, *Geographic Information / Geomatics* and to Working Group 4 *Geospatial Services* for review and comments.

Introduction

This part of ISO 15046 specifies the encoding rule that shall be used to enable international data interchange. It allows geographic information defined by application schemas and standardised schemas to be coded into a system independent data structure suitable for transport and storage.

The family of standards is organised as a series of parts, each published separately. References to the different parts are given in parenthesis below. The background, the overall structure of this standard and the fundamental description techniques are defined in the following parts, i.e., Reference Model (1), Overview (2), Conceptual Schema Language (3) and Terminology (4). User of this standard will develop application schemas to capture the semantics of geographic information. An application schema is compiled by integrating elements from a set of standardised conceptual schemas developed in the following parts of this standard: Spatial schema (7), Temporal schema (8), Feature cataloguing methodology (10), Spatial referencing by coordinates (11), Spatial referencing by geographic identifiers (12), Quality principles (13), Metadata (15), and Portrayal (17). How this integration shall take place is described in Rules for Application Schemas (9). The family of standards also defines a set of common services that shall be available when developing geographic information applications. The common services are generally defined in the Services (19) part and will cover access to and processing of geographic information according to the common information model. Three service areas are defined more closely in the Positioning (16), Portrayal (17) and Encoding (18) parts. Implementation issues are covered by: Conformance and Testing (5), Profiles (6), Quality Evaluation Procedures (14) and this part of the standards.

This standard is logically divided into two modules. The first module consist of clauses 6 and 8 which is concerned with fundamental interchange requirements that is independent of any particular encoding format. The second module is dependent on Extensible Markup Language (XML) as the encoding format and consist of clauses 9 and 10.

For readers unfamilliar with XML, please read Annex C. For a short introduction to ISO/IEC 10646 see Annex D. Annex E contains examples of the application of this standard.

Geographic information — Part 18: Encoding

1 Scope

This part of ISO 15046 specifies the encoding rule that shall be used for data interchange purposes. The encoding rule allows geographic information defined in an application schema to be coded into a system independent data structure suitable for transport or storage. The encoding rule specifies the types of data to be coded, the syntax, structure and coding schemes used in the resulting data structure. Then encoding rule defined shall be used to implement encoding services.

The application schemas shall be defined using the standardised conceptual schema language specified in part 3 – Conceptual Schema Language and shall be in conformance with the rules mandated in part 9 – Rules for Application schemas.

The encoding rule specified shall be compatible with the conceptual schema language chosen, i.e., the Universal Modelling Language (UML). Since there are no encoding rules associated with UML this part of ISO 15046 defines an encoding rule based on the Extensible Markup Language (XML). Even though several encoding rules exist as ISO standards, the emerging Extensible Markup Language seems best fit for defining one internationally adopted encoding rule for geographic information. XML is system and computing platform independent, it has a large market push and it is designed to be interoperable with the World Wide Web.

The choice of one international encoding rule does not exclude application domains and national countries to define and use their own encoding rules that can be platform dependent or more effective with regards to data size or processing complexity.

This part of ISO 15046 does not define any digital media or communication formats, neither does it define any transfer protocols.

2 Conformance

To comply with this part of the standard an implementation shall satisfy the requirements specifiecd in the Abstract Test Suite defined in Annex A.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

UML 1.1:1997, *Universal Modelling Language (UML), Object Management Group (OMG)*, *http://www.omg.org/*

XML 1.0:1998, *Extensible Markup Language (XML), W3C Recommendation 10-February-1998 – http://www.w3.org/TR/REC-xml-19980210.*

ISO 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

4 Terms and definitions

For the purposes of this International Standard, the terms and definitions given in ISO 15046-4 and the following apply.

4.1
application schema
conceptual schema for applications with similar data requirements [ISO/TC 211/WG1 N137, harmonized definition Oxford, UK 1997-09-28—29]

NOTE An application schema describes the content, the structure and the constraints applicable to information in a specific application domain.

4.2
character
member of a set of elements that is used for the representation, organisation, or control of data [ISO/IEC 2382-1:1993]

4.3
code
representation of a label according to a specified schema [ISO/TC 211/WG1 N087, harmonized definition Sydney, Australia 1997-01-20—24]

4.4
conceptual schema
schema of a conceptual model [ISO/TC 211/WG1 N137, harmonized definition Oxford, UK 1997-09-28—29]

4.5
conceptual schema language
formal language based on a conceptual formalism for the purpose of representing conceptual schemas [ISO/TC 211/WG1 N137, harmonized definition Oxford, UK 1997-09-28—29]

NOTE - A conceptual schema language may be lexical or graphical.

4.6
data
[data structure]
reinterpretable representation of information in a formalised manner suitable for communication, interpretation, or processing [ISO/IEC 2382-1:1993]

4.7
data communication
transfer of data among functional units according to sets of rules governing data transmission and the coordination of the exchange [ISO/IEC 2382-1:1993]

4.8
data interchange
procedure for delivery, receipt and interpretation of data [ISO/TC 211/WG1 N087, harmonized definition Sydney, Australia 1997-01-20—24]

4.9
data medium
material in or on which data can be recorded and from which data can be retrieved [ISO/IEC 2382-1:1993]

4.10
data transfer
move data from one point to another over a medium [ISO/TC 211/WG1 N087, harmonized definition Sydney, Australia 1997-01-20—24]

NOTE - Transfer of information implies transfer of data.

4.11
data type
identifier that specifies a legal value domain and legal operations on values in this domain

EXAMPLE - Integer, Real, Boolean, String.

4.12
dataset
identifiable collection of related data [ISO/TC 211/WG1 N087, harmonized definition Sydney, Australia 1997-01-20—24]

NOTE - A dataset may be a smaller grouping of data which, though limited by some constraint such as spatial extent or feature type, is located physically within a larger dataset. Theoretically, a dataset may be as small as a single feature or feature attribute contained within a larger dataset.

4.13
database
collection of data organized according to a conceptual structure describing the characteristics of these data and the relationships among their corresponding entities, supporting one or more application areas [ISO/IEC 2382-1:1993]

4.14
encoding
conversion of data into a series of codes [ISO/TC 211/WG1 N087, harmonized definition Sydney, Australia 1997-01-20—24]

4.15
encoding rule
identifiable collection of specifications that defines the encoding for a particular data structure

EXAMPLE - XML, ISO 10303-21, ISO 8211, etc.

NOTE - An encoding rule specifies the types of data to be converted as well as the syntax, structure and codes used in the resulting data structure.

4.16
file
named set of records stored or processed as a unit [ISO/IEC 2382-1:1993]

4.17
information
knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning [ISO/IEC 2382-1:1993]

4.18
interface
shared boundary between to functional units, defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges, and other characteristics, as appropriate [ISO/IEC 2382-1:1993]

4.19
interoperability
capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units [ISO/IEC 2382-1:1993]

4.20
medium
substance or agency for storing or transmitting a data structure

EXAMPLE: Compact disc, internet, radio waves, etc.

4.21
transfer protocol
common set of rules for defining interactions between distributed systems

5 Symbols and abbreviated terms

UML – Unified Modelling language

XML – Extensible Markup Language

URI – Unified Resource Identifier

6 Fundamental concepts and assumptions

The purpose of this family of standards is to enable interoperability between heterogeneous geographic information systems. To achieve interoperability between heterogeneous systems two fundamental issues need to be determined. The first issue is to define the semantics of the content and logical structures of geographic data. This shall be done in an application schema. The second issue is to define a system and platform independent data structure that can represent data corresponding to the application schema.

This clause describes the fundamental concepts of data interchange, i.e., the procedure of defining the application schema, encoding, delivery, receipt and interpretation of geographic data. Clause 6.1 describes an overview of the data interchange process. Clause 6.2 introduces application schemas that allow interpretation of geographic data. Clause 6.3 describes the imprtance of the encoding rule for producing system independent data structures. Clause 6.4 describes a software component for executing the encoding rule, called the encoding service and clause 6.5 describes the procedure for delivery and receipt, i.e., the transfer services.

6.1 Data interchange

An overview of a data interchange is shown in Figure 1. System A wants to send a dataset to system B. To ensure a successful interchange A and B must decide on three things, i.e., a common application schema, which encoding rule to apply and what kind of transfer protocol to use. The application schema *I* is the basis of a successful data transfer and defines the possible content and structure of the transferred data, whereas the encoding rule defines the conversion rules for how to code the data into a system independent data structure.

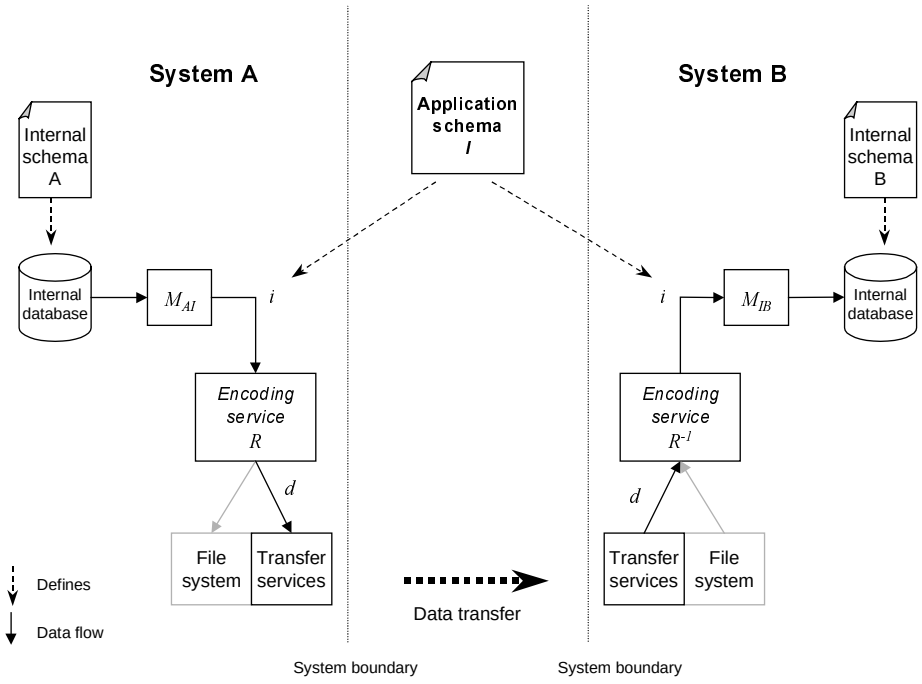


Figure 1 — Overview of Data Interchange

Both systems, A and B, store data in an internal database according to an internal schema, that usually are different, i.e., A ≠ B. The following logical steps must be taken in order to transfer a dataset from A's internal database to B's internal database.

a) The first step for system A is to translate its internal data into a data structure that is according to the application schema. This can be done by defining a mapping from the concepts of the internal schema to the concepts defined in the application schema and by writing appropriate mapping software to translate the data instances. In the figure this mapping is denoted M_{AI}. The result is an application schema specific data structure *i*. This data structure is stored in memory or on a intermediate file and is system dependent and thus not suitable for transfer.

b) The next step is to use encoding services to apply the encoding rule *R* to create a data structure that is system independent and therefore suitable for transfer. This coded dataset is called *d* and may be stored in a file system or transferred using a transfer service.

c) System A then invokes a transfer service to send the coded dataset *d* to system B. The transfer service follows a transfer protocol for how to do packaging and how the actual transportation over an on-line or off-line communication medium should take place. Both parties must agree upon the transfer protocol used.

d) The transfer service on system B receives a transfer dataset, and according to the protocol the dataset is unpacked and stored as a coded dataset *d*, e.g., on an intermediate file.

e) In order to get an application schema specific data structure *i,* system B applies the inverse encoding rule R^{-1} to interpret the coded data.

f) To use the dataset B must translate the application schema specific data structure *i* into its internal database. This by defining a mapping from the application schema into his internal schema and by writing software that does the actual translation. In the figure this mapping is denoted M_{IB}.

This part of ISO 15046 only specifies the encoding rule and the encoding service and not the whole data interchange process. Thus only steps b) and e) will be standardised. Steps c) and d) uses general information technology transfer services and steps a) and f) are the responsibility of system vendors.

6.2 Application schema

An application schema is a conceptual schema for applications with similar data requirements. The application schema is the basis of a successful data interchange and defines the possible content and structure of the transferred data. This family of standards specifies a framework for how to write application schemas. The rules for how to create application schemas are given in ISO 15046-9. The rules include specifications on how to use the standardised schemas to define feature types and how to structure feature instances into datasets.

The application schema shall be written in the UML conceptual schema language. Both a sender and a receiver of data must have access to the application schema. The application schema shall be exchanged before data interchange can take place, and both the sender and receiver shall have prepared their systems by implementing mappings and data structures according to the application schema.

An application schema can either refer to or incorporate standardised schema elements. If the application schema only contains references to standardised schema elements, then the users are responsible for checking whether the references are correct. The references should, for future use, specify which version of the standardised schema elements that are used. If the standardised schema elements are incorporated in the application schema the user still has to check whether the standardised schema elements are defined according to the standard and used correctly.

The conceptual schemas can be represented either lexical, graphical or a combination of lexical or graphical. Implementation of application schema specific data structures can be semi-automated, e.g., by using a compiler for the lexical part or by using a graphical modelling tool with support for code generation.

This standard does not specify how application schemas are to be transferred.

NOTE Object Management Group (OMG) has developed a metadata interchange standard called XML Metadata Interchange (XMI). It is reccomended to use this standard for interchange of UML models.

6.3 Encoding rules

An encoding rule is an identifiable collection of specifications that defines the encoding for a particular data structure. The encoding rule specifies the types of data to be converted as well as the syntax, structure and coding schemes used in the resulting data structure. An encoding rule is applied to an application schema specific data structure to produce a system independent data structure suitable for transport or storage. In order to define an encoding rule three important aspects must be specified, i.e., the input data structure, the output data structure and the conversion rules (or mappings) between the elements of the input and the output data structures.

6.3.1 Input data structure

The input data structure is an application schema specific data structure. The data structure can be thought of as a set of data instances, i.e., $\boldsymbol{i} = \{ i_1, ..., i_p \}$. Each data instance i_k is an instance of a concept I_l defined in an application schema. The application schema defines a set of concepts $\boldsymbol{I} = \{ I_1, ..., I_m \}$.

The application schema is a conceptual schema $\boldsymbol{c}$ written in a conceptual schema language $\boldsymbol{C}$. The conceptual schema defines a set of concepts $\boldsymbol{c} = \{ c_1, ..., c_m \}$ by instantiating the concepts of the conceptual schema language $C = \{ C_1, ..., C_r \}$. Since the application schema is a conceptual schema we have that $\boldsymbol{c} = \boldsymbol{I}$.

6.3.2 Output data structure

The output data structure is defined by a schema $\boldsymbol{D} = \{ D_1, ..., D_s \}$. The output data structure can be thought of as a set of data instances, i.e., $\boldsymbol{d} = \{ d_1, ..., d_q \}$ where each data instance d_k is an instance of a concept D_l.

The schema D defines the syntax and coding schemes of the output data structure.

6.3.3 Conversion rules

A conversion rule specifies how a data instance in the input data structure shall be converted to null, one or more instances in the output data structure. The conversion rules are defined based on the concepts of the conceptual schema language C and on the concepts of the output data structure schema D. We need to specify a conversion rule R_i for each of the concepts and legal combination concepts in the conceptual schema language. The set of

conversion rules are $\boldsymbol{R}$ = { R_1, ..., R_n }, where R_i is the *i*-th conversion rule and C_i is the *i*-th combination of instances from the schema language that has a well defined meaning. A conversion table for all possible C_i can be set up, where each C_i maps to a production of instances in the output data structure *D*.

$$\forall i : R_i(C_i) = \begin{cases} \varnothing \\ D_x \\ \{D_x, \ldots, D_y\} \end{cases}$$

NOTE The encoding rules are defined based on the two schema languages and not on any particular application schema. This is a generic approach that allows developers to write application schema independent encoding services, which can be used for different application schemas as long as the schemas are defined in the same conceptual schema language.

6.4 Encoding service

An encoding service is a software component that has implemented the encoding rule and which provides an interface to encoding and decoding functionality.

Figure 2 presents a diagram of an encoding service with relationships to important specification schemas. The encoding service shall be able to read the input data structure and convert the instances to an output data structure and vice versa. It shall also be able to read the application schema declarations and write the corresponding output data structure schema. The input data structure is defined by an application schema. The application schema is defined using concepts of the conceptual schema language. The output data structure is also described with a schema, called the data structure schema, which defines the possible content, structure and coding schemes of the output data structure. The data structure schema is described with a schema language. The encoding rule specifies conversion rules at two levels, the first is at the schema level and the second is at the instance level. Thus, at the schema level the conversion rules defines a mapping for each of the concepts defined in the application schema to corresponding concepts in the data structure schema. At the instance level the conversion rules defines a mapping for each of the instances in the input data structure to corresponding instances in the output data structure.

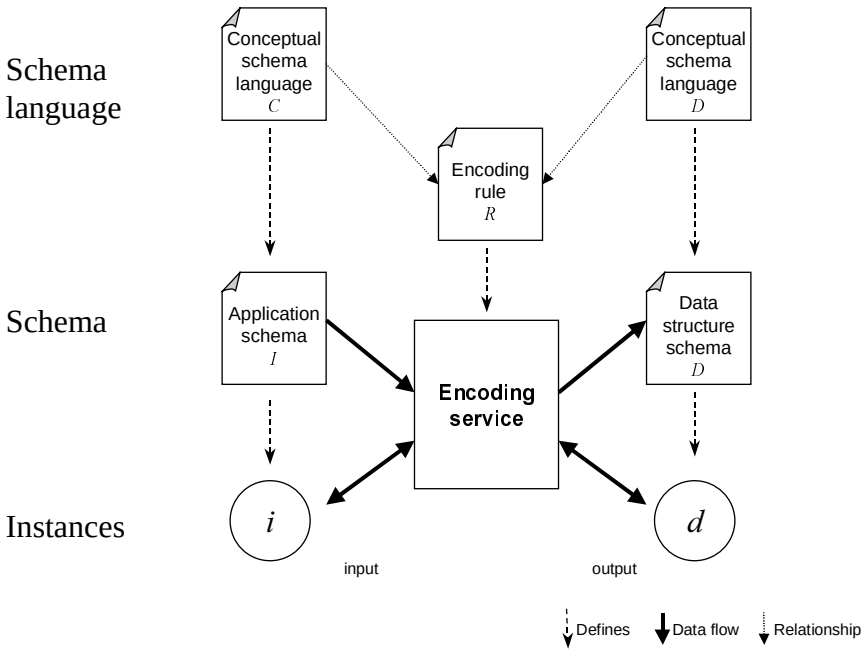


Figure 2 — Schematic overview of the encoding process

An encoding service shall at least provide interfaces for encoding and decoding functionality. Examples of such interfaces are for encoding ***d = encode (i, I)***, and for decoding: ***i = decode (d, I)***. Here, ***i*** is a reference to an application schema specific data structure, ***I*** is a reference to the application schema and ***d*** is a reference to the system independent data structure.

An encoding service component can be implemented in several ways. One way is to parse the application schema and automatically generate software that can be compiled to an encoding service component. The generated software shall consist of two parts. The first part is code for representing the application specific data structure and the second part is code for doing the actual encoding. The encoding service component is of course system and platform dependent, and may be written in any programming language, e.g., Java, Visual Basic or C++.

6.5 Transfer services

A transfer service is a software component that has implemented one or more transfer protocols, which allows data transfer between distributed information systems over off-line or on-line communication medium. To successfully transfer data between two systems the sender and receiver needs to agree on the transfer protocol to be used.

Different transfer protocols can be defined. One example is off-line transfer protocols where data is stored on optical or magnetic medium and delivered using postal services or other dedicated delivery services. An other example is on-line transfer protocols where data is compressed and included as an email attachment, delivered using a file transfer protocol or transferred using other distributed information technology services which rely on an underlying network service.

This standard does not prescribe any preferred transfer protocols.

7 Character repertoire

This standard adopts the international character set standards ISO/IEC 10646-1 and ISO/IEC 10646-2. These standards define an international recognised repertoire of characters and their encoding. There exist several encodings for the two adopted international standards. The encodings that shall be used within this family of standard are the Universal Transfer Format (UTF) UTF-8, UTF-16, and the Universal Character Set (UCS) UCS-2, and UCS-4. Two are of fixed size, i.e., UCS-2 and UCS-4 and the other two are of variable size UTF-8 and UTF-16. These character coding schemes shall be used in international profiles of this standard. However within national profiles and system implementation different character coding mechanisms may be used. For a closer description of these standards see Annex D. The fixed size character encodings are often used in database implementations and the variable size are often used for data interchange purposes. Encoding service implementations that comply to this standard shall be able to support all of these standards.

The character set standards adopted only specify the repertoire of characters and give no indication on which language that is actually used. In cases where it is important to distinguish between different languages there shall be an mechanisms for tagging text strings to indicate the language used.

ISO/IEC 10646 defines mechanisms for creating composite characters. Composite characters are characters that are composed by a base character that is modified by superimposing it with one or more additional characters. ISO/IEC 10646 also defines a set of precomposed characters and their defined decomposition. For example ö has the defined decomposition o¨. Since mixing composite characters with their precomposed equivalents may lead to interpretation problems it is deprecated to use a composite character if a precomposed character exist. That is the precomposed character shall always be used.

8 Application schema specific data structure

8.1 General

This clause defines two abstract models that shall be used to represent application schemas and data corresponding to any application schema written in UML. The models are abstract in the sense that they are not dependent of any programming language, database or application schema and that they may not have to be implemented in order to realise encoding services. They are tools for reasoning with application schemas and data, for expressing examples, and for defining the conversion rules stated in clause 9. The models are based on the

metamodel of UML and on object-oriented principles. The models are called schema model and instance model, respectively. The schema model shall be used to represent application schemas and standardised schemas. The instance model shall be used to represent data corresponding to an application schema.

An application schema shall be written in UML and consist of application defined concepts expressed as packages, classes and associations. Clause 8.2 describes the repertoire of data types that are allowed in a class declaration. Clause 8.3 describes in general associations. Clause 8.4 defines the schema model and clause 8.4.5 defines the instance model and thus which concepts in the Unified Modelling Language that are mapped to the instance model and which are not.

8.2 Attributes and data types

8.2.1 Categories

This clause describes the repertoire of data types that shall be used in the UML-models of the different parts of ISO 15046 and in the application schemas developed by the users of this family of standard. The fundamental building blocks out of which all forms of data are composed are primitive data elements such as numbers, coordinates, text strings, dates, etc. A data type defines the legal value domain and the operations on values of that domain. The different data types can be grouped into four categories:

a) Basic data types: Fundamental types for representing values, examples are String, Integer, ..., Binary, Boolean, Date, Time, etc.

b) Collection data types: Template types for representing multiple occurrences of other types, examples are Set, Bag and Sequence.

c) Enumerated data types: A mechanism for defined a list of legal values, where each value is a mnemonic word with well defined semantics.

d) Model types: Classes defined in application schemas or standardised schemas, examples are GM_Point, Building, etc.

The repertoire of data types are described in the following subclauses. Each data type is described and encodings for both character data and binary data are considered. In addition to the description of the type a simple example of character encoding is given.

NOTE 1 This standard does not put any restrictions on how to represent values of these data types in implementations, only on their names and on their encoding in the output data structure.

NOTE 2 Encoding standards define how these data types are delimited and identified. Various ISO coding standards define how these data elements are coded; that is, how bit patterns are assigned to represent the allowed values of each of the data elements. For example, ISO 8601 defines how to represent date and time.

8.2.2 Basic data types

8.2.2.1 Integer

A signed integer number, the length of an integer is encapsulation and usage dependent. Character encoding may make use of the characters "1234567890" and " - " and binary encoding may be in two’s complement integer form including a null state for fixed field encodings to represent the absence of a number. Where required, the largest negative number in two’s complement binary encoding can serve as an indication that the number field contains no number.

Example: 29, -65547

8.2.2.2 Real

A signed real (floating point) number consisting of a mantissa and an exponent, the length of a real is encapsulation and usage dependent. Character encoding may make use of the characters "1234567890 - ." and "

E " and binary encoding may be in IEEE 754-1986 (Standard for Binary Floating Point Arithmetic). IEEE 754 contains an indication for a null state.

Example: 23.501, -1.234E-4, -23.0

8.2.2.3 Binary

A set of bits representing a single bit coded value such as a pixel value or a complete image. Bit value encodings may be a concatenation of integer, real or fraction numbers, or be a bit level interleaving. Numerous ISO standards exist for bit level encoding of pixel data [see ISO JTC1 SC29 and TC 130]. Character Encoding may use hexadecimal numbers.

Example: FE10A0

8.2.2.4 String

A string is an arbitrary-length sequence of characters including accents and special characters from repertoire of one of the adopted character sets:

— ISO/IEC 10646-1: Universal Character Set (UCS) repertoire implementation level 2 also called the Base Multilingual plane of ISO 10646 (i.e. Including Latin, alphabet, Greek, Cyrillic, Arabic, Chinese, Japanese etc.)

— ISO/IEC 10646-2: Universal Character Set repertoire UCS-4, the full repertoire of ISO 10646.

Example: "Ærlige Kåre så snø for første gang."

The maximum length of a string is encapsulation and usage dependent. There shall be mechanisms for identification of the language of string values, this is called a language tag. There are two alternative ways of doing this:

1) As a language tag outside the string, following the internet language tagging mechanism as used in XML. Example xml:lang="EN-us".

2) As language tags inside the string, according to ISO/IEC JTC 1/SC 2's ongoing work on using plane 14 of ISO/IEC 10646 to identify language tags within text strings.

Both language tag mechanisms shall be supported by implementations that comply to this standard.

NOTE 1 The language tagging mechanism on string values are meant as a supplement to the default language specification in Metadata.

NOTE 2 Alternative 2 is not yet reached a official status as an amendment to ISO/IEC 10646, but as soon as it is available it should be implemented.

8.2.2.5 Date

A date gives values for year, month and day. Character encoding of a date is a string which shall follow the format for date specified by ISO 8601. Binary encoding may be a combination of integer values.

Example: 1998-09-18

8.2.2.6 Time

A time is given by a hour, minute and second. Character encoding of a time is a string which follows the ISO 8601 format, where optional elements are the time zone according to GMT. Binary encoding may be a combination of integer values.

Example: 18:30:59 or 18:30:59+01:00

8.2.2.7 DateTime

A DateTime is a combination of a date and a time type. Character encoding of a DateTime shall follow ISO 8601. Binary encoding may be a combination of integer values.

Example: 1998-09-18T18:30:01

8.2.2.8 Boolean

A value specifying TRUE or FALSE. Character Encoding may make use of the characters "0" and " 1 ", "T" or "F", or "true" or "false" and binary encoding are equivalent to integer encoding with a bit assigned to the meaning of true or false.

Example: true or false

8.2.2.9 DirectPosition

An ordered set of numbers called coordinates, that represent a position in a coordinate system. The coordinates may be in any number space of any dimension. Encodings may be a concatenation of integer, real or fraction numbers, or be a bit level interleaving.

Example: (123, 514, 150)

NOTE This data type is expected to be defined in part 11 – Spatial referencing by coordinates.

8.2.2.10 Summary of basic data types

Table 1 gives a summary of the basic data types.

Table 1 — Summary of basic data types

Data type	Description
Integer	An integer number.
Real	A floating point real number.
Binary	A sequence of octets.
String	A string of characters.
Date	A string which follows the ISO 8601 formats for time.
Time	A string which follows the ISO 8601 formats for date.
DateTime	A string which follows the combined date / time format of ISO 8601.
Boolean	A quantity that takes the values TRUE or FALSE.
DirectPosition	A set of number representing a coordinate in a coordinate system.

8.2.3 Collection types

A collection type is a template type that indicates multiple occurences of instances of a specific type. There are three different types of collections: set, bag and sequence. They have different semantics with regard to the ordering of the elements and the possible operations allowed on the collection. An collection type usually do not have an upper bound, i.e., a limit on the number of elements. The collection type is a template type because it

takes a type as an argument. The argument is the type of the elements of the collection. The maximum number of elements shall be specified as a constraint.

8.2.3.1 Set

A set shall not contain any duplicated instances. There is not specified any ordering of the elements of the set.

Type: Set (T), where T is the data type of the legal elements of the bet.

Example: Set (GM_Point)

8.2.3.2 Bag

A bag may contain duplicated instances. As for the set there is no specified ordering among the elements of a bag.

Type: Bag (T), where T is the data type of the elements of the bag.

Example: Bag (Integer)

8.2.3.3 Sequence

A sequence is a collection that has specified an sequential ordering between the elements. A synonym to sequence is List.

Type: Sequence (T), where T is the data type of the elements of the sequence.

Example: Sequence (String)

8.2.4 Enumerated types

An enumerated type declaration defines a list of valid identifiers of mnemonic words. Attributes of an enumerated type can only take values from this list. The order of which the identifiers are named in the specification defines the relative order of the identifiers.

Example:attr1 : BuildingType, where BuildingType is defined as Enum BuildingType { Public, Private, Turist };

NOTE It is difficult to give a enumerated type a specific type name in UML. Usually users want to define a enumerated type, give it a name and then use it as types of attributes of more than one class. In UML the default way of stating an enumerated type is to give the list of allowed identifiers, e.g., attr1 : { public, private, turist }.

8.2.5 Model types

The model types are all classes defined in the UML model. They can be used as data types in attribute declarations or as target types in association ends. When a model type is used in an attribute declaration it shall be interpreted similar to a composite association.

8.3 Associations and association ends

An association is used to describe a relationship between two or more classes. UML defines three different types of associations called association, aggregation and composition. The three assoiation types have different semantics. An ordinary association shall be used when users want to represent general relationship between two classes. Whereas the aggregation and composition associations shall be used to create part-whole relationships between two classes.

A binary association has a name and two association-ends. An association-end has a role name, a multiplicity statement, an optional aggregation symbol and shall always be connected to a class.

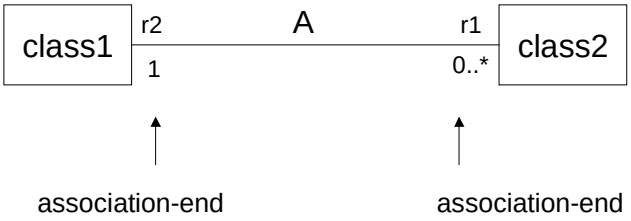


Figure 3 — Association

Figure 3 shows an association named "A" with its two respective association-ends. The role name is used to identify the end of an association, the role name **r1** identifies the association-end which is connected to the class named **class2**. The multiplicity of the association-end can be one of exactly-one (1), zero-or-one (0..1), one-or-more (1..*), zero-or-more (0..*) or an interval (n..m). Viewed from the class the role name of the opposite association-end identifies the role of the target class. We say that **class2** has an association to **class1** that is identified by the role **r2** and which as a multiplicity of exactly-one. The other way around we can say that **class1** has an association to **class2** that is identified by the role name **r1** with multiplicity of zero-or-more. In the instance model we say that **class1** objects reffers to zero-or-more **class2** objects and that **class2** objects have a reference to exactly one **class1** object.

An aggregation association is a relationship between two classes, where one of the classes plays the role of container and the other plays the role of a containee. Figure 4 shows an example of an aggregation, notice the aggregation symbol at the association-end close to **class1**. Here **class1** is an aggregation of **class3**. We say that **class3** is a part of **class1**. In the instance model we have that **class1** objects contain one-or-more **class3** objects. The aggregation association shall be used when the objects representing the parts of a container object can exist without the container object.

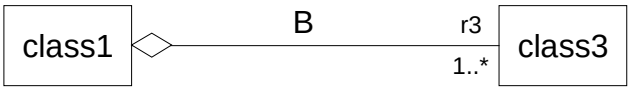


Figure 4 — Aggregation

A composition association is a strong aggregation, in the sense that if a container object is deleted all its containee objects are deleted as well. The composition association shall be used when the objects representing the parts of a container object, cannot exist without the container object. Figure 5 shows a composition association, notice that the composition symbol is filled. Here **class1** objects consist of one-or-more **class4** objects, and the **class4** objects cannot exist without the existence of a **class1** object.

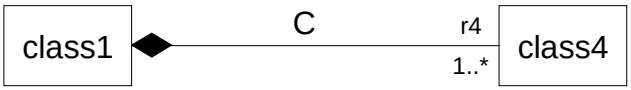


Figure 5 — Composition

8.4 Schema model

8.4.1 General

The purpose of the schema model is to represent an application schema with associated standardised schemas and their elements. This model consist of the following three main concepts: Package, Class and Association. Figure 6 shows the schema model. The model is based on UML's metamodel but simplified.

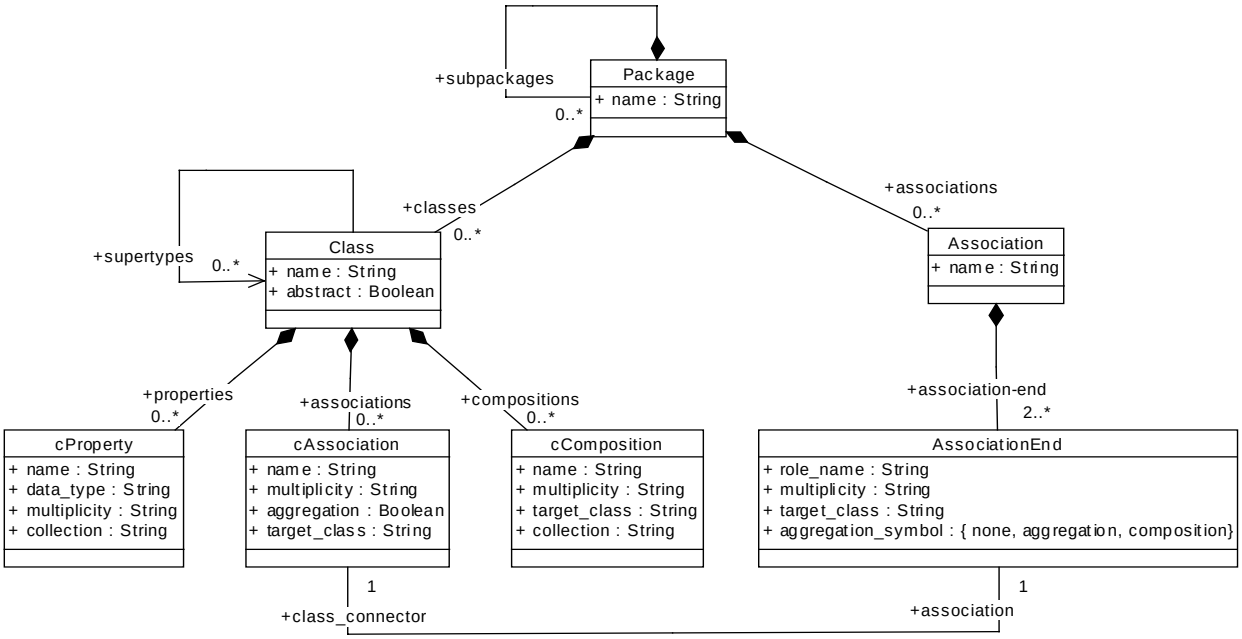


Figure 6 — Schema model

8.4.2 Package

A UML package is a container that is used to group declarations of subpackages, classes and their associations. The package structure in UML enables a hierarchical structure of subpackages, class declarations and assocations. A package shall be used to represent a schema. The structure of packages shall be as follows: At the top level (level 0) there shall be one package representing the complete schema. At the next level (level 1) there shall be a package for the application schema and a package for each of the standardised schemas used in the application schema. At the next level (level 2) there shall be one package for each subpackage and so on.

Level 0 – Complete schema
Level 1 – Application schema package, name "RoadMap"
Class and Association declarations
Level 2 - Subpackages
Level 1 – Standardised schema package, name "Spatial"
Class and Association declarations
Level 2 - Subpackages
Level 1 – Standardised schema package, name "Quality"
Class and Association declarations
Level 2 - Subpackages

Figure 7 — Example package structure

8.4.3 Class

A UML class has a name, attributes, operations, constraints and participates in associations. All these aspects of the class can be represented as attributes in the schema model. Only aspects important to capture the state of a class shall be represented, thus operations and constraints are not represented in the schema model.

A class in the schema model has a name and three categories of attributes represented by the three classes: cProperties, cAssociations and cCompositions. All attributes defined in the UML class that is of basic types, enumerated types or collection type of basic or enumerated types shall be represented a cProperties object. All attributes of model type shall be represented as a cComposition object. Composite associations shall also be represented as a cComposition object. The other associations of which the class participates in shall be represented as a cAssociaton object, with the corresponding role names as the name. A class shall have an association to its supertype if any and a indication whether the class is abstract or not.

A cProperty class has a name, a data type and a multiplicity statement and a collection statement. The collection statement indicates the type, if any, of collection semantics that shall be applied on multiple occurences of the data type.

A cAssociation class has a name, a multiplicity statement, aggregation type, the name of the target class, and a reference to the corresponding association-end. The name of the attribute corresponds to the opposite association-end's role name and the multiplicity of the attribute corresponds to the multiplicity of the association-end.

A cComposition class has a name, a multiplicity, a reference to the target class, and a collection statement.

UML allow users to define stereotypes that can be used to model classes with specific semantics. The schema model handles all classes the same way. This means that stereotype classes shall be mapped to ordinary classes.

In UML all attributes are per default mandatory. By using the stereotype mechanism of UML users may specify either <<optional>> or <<conditional>> at the start of an attribute declaration. Where <<optional>> means this attribute may have a value or it can have been omitted. The <<conditional>> statement means that this attribute has a value (is present) if some conditions are fulfilled, e.g., that a previous attribute has a specific value. The condition shall be expressed as an OCL constraint in connection with the class declaration. This mean that a null value must be represented in the instance model, e.g., a place holder element or a null value. Attributes with optional or conditional stereotypes shall be mapped to its corresponding cProperty or cComposition with a multiplicity of (0..1).

8.4.4 Association

A Association class has a name and a aggregation association to two or more association-ends. Each association-end has a role name, multiplicity statement, target class and optional aggregation symbol.

8.4.5 Qualified names

The packages, classes and attributes in the schema model shall be identified by a qualified name. The form of the qualified names is *name1.name2.name3*, where *name1* is the name of the outermost package, *name2* is a name which appears within the namespace of *name1* and *name3* is a name that appears within the namespace of *name2*. The "." character shall be used as a name separator. There is no limit of the depth of this namespace hierarchy.

Example: In the Spatial schema there is a subpackage named Geometry, which defines a class named GM_Object, which again has an association with role name SRS. The fully qualified name for this association is: *Spatial.Geometry.GM_Object.SRS*.

8.5 Instance model

8.5.1 General

The purpose of the instance model is to represent data corresponding to an application schema. This models follows the schema model, but does not represent the associations directly, but as attributes of object reference type. The model is given in Figure 8. The instance model consist of instances of the classes found in the schema model. These instances are called objects. The instance model is a flat model in that all objects in the model are accessible at the same level through a unique object-identifier. Thus this object-identifier is the key for accessing objects in the instance model. The instance model has in addition a simple indexing mechanism that is based on the package structure of the schema model. This indexing mechanism is realised with the iPackage class, and it enables the user to ask for all objects of a particular schema.

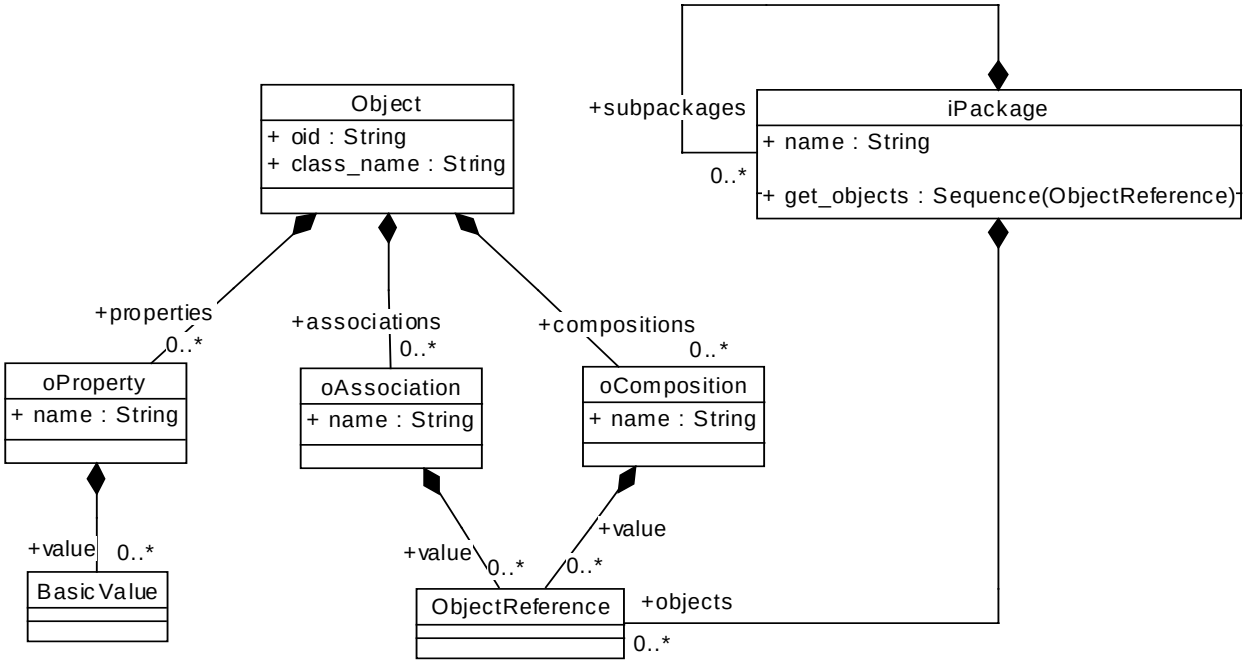


Figure 8 — Instance model

8.5.2 Package

A package shall be mapped to an instance package (iPackage) in the instance model. An iPackage is a simple index that shall contain references to all objects whose class is defined in the corresponding UML package. The instance packages are arranged in a hierarchical manner according to their structure in the UML model, that is subpackages shall be contained in their parent package. At the top level (level 0) there shall be one package representing the whole dataset. At the next level there shall be a package for the application schema and a package for each of the standardised schemas used in the application schema. At the next level (level 1) there shall be one package for each subpackage and so on. An instance package shall be able to return all object references held by itself and by its underlying packages. That means that the subpackage hierarchy below level 1 may be collapsed, see Figure 9.

Level 0 – Top package
Level 1 – Application schema package, name "RoadMap"
<object references to RoadMap objects, ...>
Level 1 – Standardised schema package, name "Spatial"
<object references to spatial objects, ... >
Level 1 – Standardised schema package, name "Quality"
<object references to quality objects, ... >

Figure 9 — Example of instance package structure

8.5.3 Object

An instance of a class is called an object. Every object shall have a unique identifier within an instance model and a reference to its class name. The state of the object is represented by a list of propertie values, a list of association values and a list of compositions values corresponding to the declaration in the schema model.

The properties list shall contain instances of oProperty for each of the corresponding instances of cProperty in the schema model. A oProperty has a name and a value. The value is a list of BasicValue objects. This instance model defines the BasicValue to be a text string which can encode basic data types and enumerated data types. The values shall be instantiated with values according to the value domain of their data type and further restricted to the value domains stated in the constraint statements of the class.

The association list shall contain instances of oAssociation for each of the corresponding instances of cAssociation in the schema model. A cAssociation has a name and a value. The value is a list of ObjectReference objects. This instance model defines the ObjectReference to be of a string data type. An ObjectReference is an object that contains an object-identifier that uniquely identifies the associated object within the dataset.

The composition list shall contain instances of oComposition for each of the corresponding instances of cComposition in the schema model. A cComposition has a name and a value. The value is a list of ObjectReference objects.

An abstract class shall not be instantiated. But in cases where abstract classes are used as the data types of attribute or association declarations special care must be taken.

Objects based on classes that have supertypes shall contain all the properties, associations and compositions of its class and of its supertypes. Thus, all attributes and associations shall be copied from the supertypes and is considered to be a part of the object. Attribute and association names shall be the way of accessing the values of the attributes and they shall therefore be unique within the class.

Operations and constraints shall not be mapped to the instance model.

NOTE 1 Attribute and association name clashes can cause problems when using inheritance. A simple way avoid this is that all attributes and associations shall be prefixed with their appropriate class name, alternatively it is left to the user.

NOTE 2 Attribute and association redeclaration can also cause problems when using inheritance. Redeclaration happens when an attribute or association declared in a supertype gets redeclared in a subtype with a new or restricted type. Many object-oriented programming language cannot handle redeclarations and therefore they shall be deprecated.

9 Output data structure

9.1 General

This clause specifies the encoding rule that shall apply to this family of standards. It consist of a set of conversion rules between UML concepts and XML concepts. There are two types of conversion rules. The first one is the schema conversion rules which define rules for how to convert the UML schema declarations to the XML Document Type Definition (DTD). The second category is the instance conversion rules which define rules for how to convert instances of the instance model into instances of the resulting data structure. Figure 10 depicts the two types of conversion rules. Notice that the schema conversion rules are a one way mapping, because of the limmitations of XML's DTD syntax. The instance conversion rules shall be two-ways.

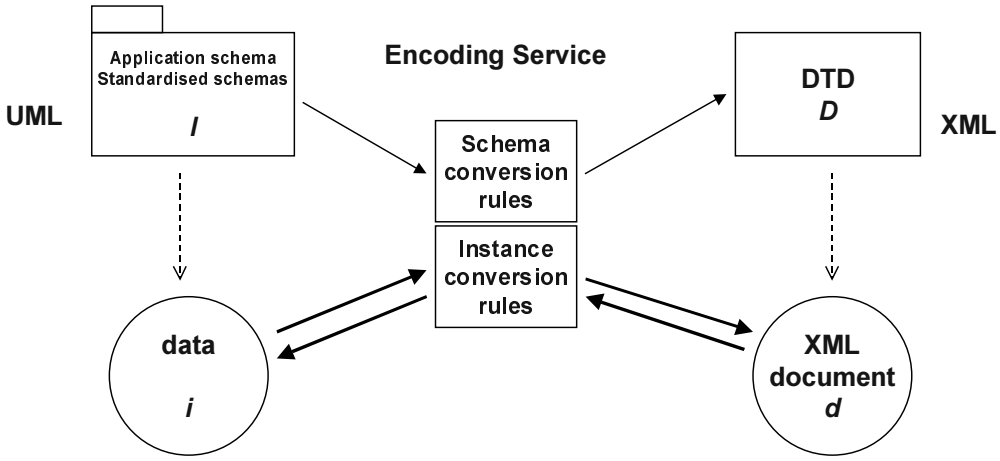


Figure 10 — Conversion rules

The schema conversion rules are described in clause 9.2 and the instance conversion rules are described in clause 9.3.

NOTE For a description of XML and XMI see Annex C.

9.2 XML DTD production

9.2.1 General

This clause describes the schema conversion rules. It describes how to map concepts in the schema model into concepts of XML DTDs. The result of an schema conversion is an XML DTD file. This file shall contain the following DTD declarations:

- An XML version processing instruction and optional encoding declaration. Example: <?xml version="1.0" encoding="UCS-2" ?>
- The required DTD declarations.
- Declarations for the application schema.
- Declarations for any of the standardised schemas used by the application schema.

The conversion rules are based on the idea that all packages, classes, attributes and associations in a UML model can be mapped to XML elements.

Subclause 9.2.2 define the required DTD declarations and subclause 9.2.2.9 defines production rules for the schema model.

NOTE The DTD production is inspired by the XMI Hierarchical Entity DTD (Rule Set 3) and it uses XML's parameter entity substitution technique extensively.

9.2.2 Required DTD declarations

The following subclauses define the required DTD declarations that always shall be part of an XML DTD file. They are common to all generated DTDs. The elements are also listed in Annex B.

9.2.2.1 Element identification attributes

Three XML attributes are defined to identify XML elements in an XML docuement and so that XML elements can be association with each other. The purpose of these attributes is to allow XML elements to refer to other XML elements using XML's IDREF, XLink, and XPointer mechanisms. The attributes are declared in an XML entity called **GI.element.att**. Its declaration is as follows:

```
<!ENTITY % GI.element.att
            ' gi.id          ID        #IMPLIED
             gi.label        CDATA     #IMPLIED
             gi.uuid         CDATA     #IMPLIED ' >
```

The **gi.id** attribute values shall be unique within an XML document, but do not have to be globally unique. The **gi.label** may be used to provide a string label identifying a particular XML element. Users may put any value in this attribute. The purpose of the **gi.uuid** attribute is to provide a globally unique identifier for an XML element. The values of this attribute shall be globally unique strings prefixed by the type of identifier. The form of the Universally Unique Identifier (UUID) may be taken from the standard defined by the Open Group or defined by the application community.

9.2.2.2 Linking attributes

Several XML attributes are required to enable XML elements to refer to other XML elements. The purpose of these attributes are to allow XML elements to act as simple XLinks or XPointers, or to hold a reference to an XML element in the same document using the XML IDREF mechanism. The entity shall be declared as follows:

```
<!ENTITY % GI.link.att
            ' xml:link       CDATA     #IMPLIED
            href             CDATA     #IMPLIED
            gi.idref         IDREF     #IMPLIED
            gi.uuidref       CDATA     #IMPLIED' >
```

The first two attributes declared **xml:link** and **href** enable an XML element to act as a linking element according to the XLink and XPointer specifications. The declaration and use of those attributes are defined in the XLink and XPointer specifications. The use of the simple XLinks shall be to set the **xml:link** attribute to "simple" and give **href** values that can be used to reference XML elements with **gi.id** or **gi.label** attributes set to particular values. The **gi.id** attribute value can be specified using a special URI form of XPointers defined in the XPointer specifications. Only the vertical-bar connector type "|" shall be allowed in a XML document following this standard.

The **gi.idref** attribute allows an XML element to refer to another XML element within the same document using the XML IDREF mechanism. The value of this attribute shall always correspond to a XML element with the **gi.id** attribute set. The **gi.uuidref** attribute shall be used to refer to an XML element that have a corresponding **gi.uuid** attribute.

9.2.2.3 Document element (GI / ISO15046)

The document element or root element of each XML document that comply to this standard shall be the **GI** element. Its declaration is:

```
<!ELEMENT GI (GI.header, GI.content?, GI.difference*) >
<!ATTLIST GI
            gi.version       CDATA     #FIXED "1.0"
```

```
            timestamp       CDATA       #IMPLIED
            verified        (true | false)      #IMPLIED
            containment (flat | nested | mixed) #IMPLIED
>
```

The **gi.version** attribute is required to be set to "1.0". This indicates that the geodata conforms to this version of the standard. Revised versions of this standard will have another number associated with them. The **timestamp** attribute indicates the date and time that the data was written. The format of the timestamp shall follow ISO 8601. The **verified** attribute indicates whether the data has been verified. If it is set to "true", verification of the model was performed by the document creator and it is an indication of that the data correspond to the application schema. In that case, XML validation should find errors only in encoding or transmission. The **containment** attribute indicates whether the structure of the XML document is flat, nested or mixed. If the value is "flat" all composite attributes are given as references to the objects which are stored under its package extent. If the value is "nested" all composite attributes are included in the application schema instances. If the value is "mixed" a combination of a flat and mixed structure can occur. The default value is "mixed".

9.2.2.4 GI.header

The GI.header element contains XML elements which identify the application schema and some metadata about the dataset. The declaration is:

```
<!ELEMENT GI.header ( GI.documentation?, GI.applicationSchema+, GI.lookup.table? ) >
```

9.2.2.5 GI.documentation

This XML element shall contain a restricted set of metadata about the XML document. The elements are the owner of the data, a contact person for the data, long and short descriptions of the data , the exporter tool which created the data, the version of the tool, and copyright or other legal notices regarding the data. In addition other information can be included as text within this element, since its content model is mixed. The declaration is:

```
<!ELEMENT GI.documentation (#PCDATA |
            GI.owner | GI.contact |
            GI.longDescription |
            GI.shortDescription | GI.exporter |
            GI.exporterVersion | GI.notice)* >

<!ELEMENT GI.owner ANY>
<!ELEMENT GI.contact ANY>
<!ELEMENT GI.longDescription ANY>
<!ELEMENT GI.shortDescription ANY>
<!ELEMENT GI.exporter ANY>
<!ELEMENT GI.exporterVersion ANY>
<!ELEMENT GI.notice ANY>
```

NOTE This must be harmonized with Metadata!

9.2.2.6 GI.applicationSchema

This XML element identifies the application schema to which the instance data being transferred conforms. There may be multiple models, if the model to which the instance data being transferred conforms to more than one model. This element is expected to be a simple XLink pointing to a URI of the application schema. The declaration is:

```
<!ELEMENT GI.applicationSchema ANY>
<!ATTLIST GI.applicationSchema
            %GI.link.att;
            gi.name         CDATA       #REQUIRED
            gi.version      CDATA       #REQUIRED
>
```

The **gi.name** and **gi.version** attributes are the name and version of the application schema respectively. The **href** attribute may contain a URI that refers to a resource that contains the application schema, e.g., encoded according to the XMI specification. Since the content is ANY, additional documentation is possible.

9.2.2.7 GI.content

The **GI.content** XML element contains the actual data being transferred. Its declaration is:

<!ELEMENT Gi.content ANY>

9.2.2.8 GI.lookup.table

This XML element holds a lookup table of short-name long-name pairs. A long-name is a valid XML name which corresponds to a qualified name in the schema model. A short-name is a valid XML name which corresponds to a long-name. The encoding service may produce XML documents with only long-names or alternatively only short-names. The rationale for this is to reduce the size of the markup in the XML document, but also to operate with language independent XML tags. The short-names must be unique within the schema. Since the applicatoin schema only uses qualified names the matching short-names must be included in the DTD file. The **GI.lookup.table** is defined as follows:

<!ELEMENT GI.lookup.table (#PCDATA) >

The content of this element shall be a list of short-name long-name pairs separated by whitespace.

9.2.2.9 Data type entities

XML does not currently support more than three data types: text, tokenized and enumerated types. A property is mapped to a corresponding XML element. The properties attributes of the schema model that are of basic types such as Integer, Real, Binary, String, Date, DateTime and Time, and shall all be declared to have #PCDATA as content. Their values shall be coded according to character encoding described in clause 8. To make the DTD easier to read the following entities shall be defined and used for defining XML elements for attributes. Here entities for content and attributes are defined. The **GI.Boolean.cont** entity shall be used as content for elements together with the **GI.Boolean.att** attributes to code an attribute of type Boolean.

<!ENTITY % GI.Integer.cont '(#PCDATA)'>

<!ENTITY % GI.Real.cont '(#PCDATA)'>

<!ENTITY % GI.Binary.cont '(#PCDATA)'>
<!ENTITY % GI.Binary.att 'length CDATA #REQUIRED'>

<!ENTITY % GI.String.cont '(#PCDATA)'>
<!ENTITY % GI.String.att
'xml:lang CDATA #IMPLIED
length CDATA #IMPLIED'>

<!ENTITY % GI.Date.cont '(#PCDATA)'>
<!ENTITY % GI.Timecont '(#PCDATA)'>
<!ENTITY % GI.DateTime.cont '(#PCDATA)'>

<!ENTITY % GI.Boolean.cont 'EMPTY'>
<!ENTITY % GI.Boolean.att ' value (true | false) #REQUIRED'>

The property attributes of the schema model that are of enumerated types shall be declared with an the **GI.Enum.cont** entity (an empty content model) and with XML attributes corresponding to the enumerated list of valid identifiers.

<!ENTITY % GI.Enum.cont 'EMPTY'>

Attributes of DirectPosition type shall be declared with a **GI.DirectPosition.cont** entity as the content model.

<!ENTITY % GI.DirectPosition.cont '(#PCDATA)'>

9.2.3 Schema DTD production

9.2.3.1 Package

A package is mapped to an XML element with the same name as the qualified package name. The content model of a package is the choice of subpackages and classes that are declared within the package. An example of a package declaration is:

<!ELEMENT p (SubPackage | Class1 | Class2 ...)*>

9.2.3.2 Class

A class is mapped to an XML element with the same name as the qualified name of the class. A class is decomposed into three parts according to the schema model, i.e., properties, associations and compositions. Three entities are declared for each class. The entities are declared for every schema model class. The name of the entity has the name of the class as a prefix and the suffix is "Properties", "Associations", or "Compositions". The properties entity contains a list of the XML elements which correspond to the schem model properties. The associations entity contains the XML elements representing the association attributes. The compositions entity contains the XML elements which represents the composition attributes.

The three entities shall be included in the content model of the XML element corresponding to the class, and the attribute list of the XML element shall have the identification attributes and linking attributes defined in the **GI.element.att** and **GI.link.att** entities:

```
<!ENTITY % cProperties 'propertiesForC'>
<!ENTITY % cAssociations 'associationsForC'>
<!ENTITY % cCompositions 'compositionsForC'>

<!ELEMENT c (%cProperties;, %cAssociations;, %cCompositions;)?>
<!ATTLIST c
    %GI.element.att;
    %GI.link.att;
>
```

Only the entities that are not empty shall be included in the content model of element c to maintain valid XML syntax.

XML does currently not have a built-in mechanism to represent inheritance. Therefore inheritance shall be represented by copying down the properties, associations and compositions of the superclass(es) using the substitution capability of XML entities. Since XML requires entities to be declared in a DTD before being used, this method of representing inheritance requires that the entities of superclasses precede the declarations of entities and elements of their subclasses.

Multiple inheritance is treated in such a way that the properties, associations, and compositions of classes that occur more than once in the inheritance hierarchy are only include once in their subclasses.

If a class **c1** has a direct superclass **c** then the declaration of the required entities for class **c1** is as follows:

```
<!ENTITY % c1Properties ' %cProperties;, properties for c1 if any...'>
<!ENTITY % c1Associations ' %cAssociations;, associations for c1 if any...'>
<!ENTITY % c1Compositions ' %cCompositions;, compositions for c1 if any...'>
```

Should there be a class, **c2**, derived from **c1**, then the entity declarations for **c2** shall be:

```
<!ENTITY % c2Properties ' %c1Properties;, properties for c1 if any...'>
<!ENTITY % c2Associations ' %c1Associations;, associations for c1 if any...'>
<!ENTITY % c2Compositions ' %c1Compositions;, compositions for c1 if any...'>
```

9.2.3.3 Properties

An property attribute is mapped to a corresponding XML element. The fully qualified attribute name becomes the element name. In the following we use the schema defined in Figure 11 as example.

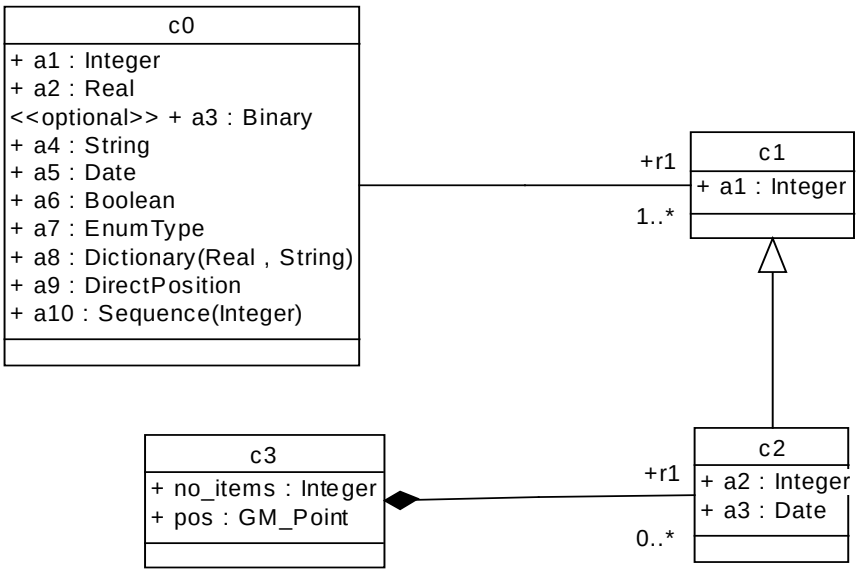


Figure 11 — Example classes

For attributes of type *Integer* the following declaration shall apply:

<!ELEMENT a1 %GI.Integer.cont;>

For attributes of type *Real* with name **a2** the following declaration shall apply:

<!ELEMENT a2 %GI.Real.cont;>

Attributes of type *Binary* shall be declared as follows:

<!ELEMENT a3 %GI.Binary.cont;>
<!ATTLIST a3 %GI.Binary.att;>

The **length** attribute shall indicate the number of octets encoded in hexadecimal.

Attributes of type *String* shall be declared:

<!ELEMENT a4 %GI.String.cont;>
<!ATTLIST a4 %GI.String.att; >

Attributes of type *Date, Time* and *DateTime* shall be declared with #PCDATA as content model and where values follows the formats specified in ISO 8601. The declaration is as follows:

<!ELEMENT a5 %GI.Date.cont;>

Attributes of type *Boolean* and user defined enumerations shall be declared as empty XML elements with a value attribute which shall have values taken from the enumerated list.

An attribute of a *Boolean* type shall be declared as:

<!ELEMENT a6 %GI.Boolean.cont;>
<!ATTLIST a6 %GI.Boolean.att;>

An attribute of an enumerated type shall be declared as:

<!ELEMENT a7 %GI.Enum.cont;>
<!ATTLIST a7
value (enum1 | enum2 | enum3) #REQUIRED>

The legal enumerated values in this example are **enum1**, **enum2** or **enum3**.

An attribute of type Dictionary where the types of the elements of the dictionary are of simple types shall be mapped to an XML element corresponding to the two content elements.

<!ELEMENT a8 (GI.Integer, GI.String)* >

An attribute of type DirectPosition shall be mapped to an XML element with empty content an the attributes specified by the entity **GI.DirectPosition.att**.

<!ELEMENT a9 %GI.DirectPosition.cont;>
<!ATTLIST a9 %GI.DirectPosition.att;>

Alternatively an attribute of the type DirectPosition shall have a GI.DirectPosition.cont entity as the content model. This corresponds to #PCDATA and thus requires that the coordinate values must be separated by whitespace.

<!ELMENT a9 %GI.DirectPosition.Alt.cont; >

Attributes of collection types such as Collection, Set, Bag, Array and Sequence shall be mapped to corresponding XML element of the parameter type of the collection. The appropriate multiplicity operator shall be indicated in the property entity of the class. For our example this will look as follows:

<!ELEMENT a10 %GI.Integer.cont; >

All XML elements that corresponds to the properties of the class shall be listed in the class' property entity, as specified in clause 9.2.3.2. The multiplicity operator following the element name in the list governs whether the element may occur one or more (+), zero or more(*), or zero or more times (?). The absence of such an operator means that the element must appear exactly once.

An optional or conditional attribute shall have the multiplicity operator (?). A collection attribute shall have the multiplicity operator (+) or (*). And all other attribute types shall not have any multiplicity operators. For our example this will look as follows:

<!ENTITY % c0Properties 'a1, a2, a3?, a4, a5, a6, a7, a8, a9, a10*'>

Here attribute **a3** and **a10** have multiplicity operators.

9.2.4 Associations

Each association attribute of the schema model is represented by an XML entity and by an XML element. The multiplicity of the role must be translated to the XML multiplicities of XML. The representation of an association attribute named **r** with multiplicity "m" for a class **c** shall be declared as:

<!ELEMENT r (*content***) >**

<!ENTITY % cAssociations 'r'

The *content* is a list of XML elements that correspond to the target class and any of the class' concrete subclasses.

For our example we have that **c0** has a assocation to class **c1** with role name **r1** and multiplicity (1..*), and we have that **c2** is a concrete subclass of **c1**.

<!ELEMENT r1 (c1 | c2) >

The multiplicity of the association attribute shall be translated to XML multiplicity and stated in the entity declaration of the association:

<!ENTITY % c0Association 'r1+'

9.2.5 Compositions

Each containment role is as the associations represented by an XML entity and an XML element. The content model of the XML element representing the composition attribute of the schema model is the XML element corresponding to the class and the XML elements corresponding to each of the concrete subclasses of the class.

<!ENTITY r (c1 | c2)>

9.3 XML document production

9.3.1 General

This clause describes the instance conversion rules. It describes how to map the objects defined in the instance model into XML documents. The result of an instance conversion is an XML document that shall be a valid XML document. This file shall contain the following:

— An XML version processing instruction and optional encoding declaration. Example: <?xml version="1.0" encoding="UCS-2" ?>

— A mandatory reference to an external DTD subset containing the DTD declarations for the application schema and for the standardised schemas used by the application schema. Example: <!DOCTYPE GI SYSTEM "http://www.ex.org/roadmap.dtd">

— Any XML element that conform to the external DTD declaration and to the application schema.

Subclause XX defines the package production rules and subclause XX defines the object production rules.

By package extent, Optional object (containment)

9.3.2 Package

The objects in the instance model shall be structured according to the index package structure. That is the first XML element shall correspond to the package representing the application schema. The content of this package contains the subpackages and objects contained within the application schema package. The next package shall correspond to the first standardised schema, and so on.

9.3.3 Object

An object shall be mapped to an XML element according to the corresponding DTDs element declaration. The content of the element shall be its property elements, association elements and its composition elements. Each object element shall as a minimum have values for their **gi.id** attributes.

The content of association elements shall contain an XML element corresponding to the target class with the empty content and with one of the following alternative attribute combinations:

1) As a linking element. The attribute **xml:link** shall have the value "simple" and the **href** value shall have contain an optional URI, a vertical-bar connector "|" and a valid XPointer fragment value.

2) As a IDREF element. The attribute **gi.idref** is used to refer to an XML element with corresponding **gi.id** attribute.

3) As a uuid reference. The attribute **gi.uuidref** is used to refer to an XML element with corresponding **gi.uuid** value.

The content of composition elements shall contain an XML element corresponding to the target class. In the case of a nested containment model the target XML element shall be given directly, thus creating a nested structure. In case of a mixed containment model the target XML element may be given directly or be a linking element pointing to the data instance. If the containment model is flat the target XML element shall be empty and only have attributes similar to the association attributes pointing to the data instance.

9.4 Character coding

DTD files and XML documents shall follow the character coding specified by this standard. The legal encodings are "UTF-8", "UTF-16", "ISO-10646-UCS-2" and "ISO-10646-UCS-4". The character encoding shall be defined in the first processing statement of the XML file. If no encoding is given the default is according to XML "UTF-8" or "UTF-16". Example:

<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>

10 Encoding service

An encoding is a software component that has implemneted the encoding rule and which provides an interface to its functionality. This clause describe the main interfaces of an encoding service that shall be implemented by system vendors. An encoding service shall at least have functionality for generate a DTD based on the application schema, encode data according to XML production rules and decode data by applying the inverse XML production rules.

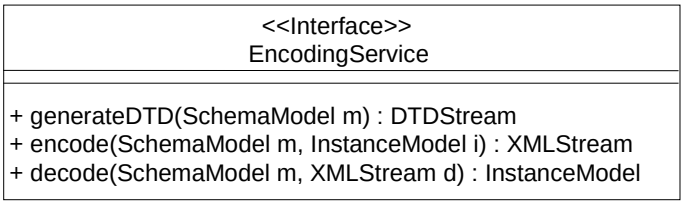


Figure 12 — Encoding service interface

The interface of the encoding service is shown in Figure 12. It contains three operations: generateDTD, encode and decode.

The generateDTD operation shall be used to generate an XML DTD file. It takes a schema model as input parameter and produces a DTDStream object as result.

The encode operation shall be used to generate an XML document. It takes a schema model and an instance model as input paramters and returns a XMLStream object.

The decode operation shall be used to interpret an XML document. It takes a schema model and an XMLStream object as input and returns an instance model.

Annex A
(normative)

Abstract Test Suite

TO BE COMPLETED!

This Annex specifies the basic conformance specification of this part of ISO 15046. The main test purpose is to check whether implementations of encoding services are in conformance to the encoding rule mandated by this standard.

The test purposes will be decomposing into sub-test purposes forming a hierarchical structure. Each test purpose forms an abstract test suite, which specifies a requirement that must be satisfied for implementations to be in conformance. The abstract test suites should include a test purpose and a test method.

If the hierarchical structure of the abstract test suites is not clear enough, an extended conformance clause (to clarify if the hierarchical structure) is needed.

There may also be necessary to define conformance classes and conformance levels.

NOTE This clause is based on ISO/TC 211 N419 – Guidelines on Abstract Test Suites and Conformance Clauses. Text must be added when the content of the standard has been stabilised.

Well-formed, Valid XML

External DTD,

Character encoding

Annex B
(normative)

Required DTD elements

This annex contains the required DTD declarations that all DTD files produced by encoding services shall include.

```
<!-- Required DTD declarations -->


<!-- Element identification attributes -->

<!ENTITY % GI.element.att
                ' gi.id                 ID          #IMPLIED
                 gi.label           CDATA       #IMPLIED
                 gi.uuid            CDATA       #IMPLIED ' >


<!-- Element linking attributes -->

<!ENTITY % GI.link.att
                ' xml:link          CDATA       #IMPLIED
                href                CDATA       #IMPLIED
                gi.idref            IDREF       #IMPLIED
                gi.uuidref          CDATA       #IMPLIED' >


<!-- Document element -->

<!ELEMENT GI (GI.header, GI.content?, GI.difference*) >
<!ATTLIST GI
                gi.version          CDATA       #FIXED "1.0"
                timestamp           CDATA       #IMPLIED
                verified          (true | false)    #IMPLIED
                containment (flat | nested | mixed)       #IMPLIED
>

<!-- GI.header element -->

<!ELEMENT GI.header ( GI.documentation?, GI.applicationSchema+, GI.lookup.table? ) >


<!-- GI.documentation elements -->

<!ELEMENT GI.documentation (#PCDATA |
                GI.owner | GI.contact |
                GI.longDescription |
                GI.shortDescription | GI.exporter |
                GI.exporterVersion | GI.notice)* >

<!ELEMENT GI.owner ANY>
<!ELEMENT GI.contact ANY>
<!ELEMENT GI.longDescription ANY>
<!ELEMENT GI.shortDescription ANY>
<!ELEMENT GI.exporter ANY>
<!ELEMENT GI.exporterVersion ANY>
```

```
<!ELEMENT GI.notice ANY>


<!-- GI.application schema element -->

<!ELEMENT GI.applicationSchema ANY>
<!ATTLIST GI.applicationSchema
                %GI.link.att;
                gi.name         CDATA       #REQUIRED
                gi.version      CDATA       #REQUIRED
>

<!-- GI.content element -->

<!ELEMENT Gi.content ANY>


<!-- GI.lookup.table element -->

<!ELEMENT GI.lookup.table (#PCDATA) >


<!-- Data type entities -->

<!ENTITY % GI.Integer.cont '(#PCDATA)'>

<!ENTITY % GI.Real.cont '(#PCDATA)'>

<!ENTITY % GI.Binary.cont '(#PCDATA)'>
<!ENTITY % GI.Binary.att 'length CDATA #REQUIRED'>

<!ENTITY % GI.String.cont '(#PCDATA)'>
<!ENTITY % GI.String.att
                'xml:lang       CDATA       #IMPLIED
                length          CDATA       #IMPLIED'>

<!ENTITY % GI.Date.cont '(#PCDATA)'>
<!ENTITY % GI.Timecont '(#PCDATA)'>
<!ENTITY % GI.DateTime.cont '(#PCDATA)'>

<!ENTITY % GI.Boolean.cont 'EMPTY'>
<!ENTITY % GI.Boolean.att ' value ( true | false ) #REQUIRED'>

<!ENTITY % GI.Enum.cont 'EMPTY'>

<!ENTITY % GI.DirectPosition.cont '(#PCDATA)'>
```

Annex C
(informative)

Extensible Markup Language (XML)

C.1 Introduction

The purpose of this Annex is to give a short introduction to XML and the rationale behind the design of this standard. This standard uses XML in a way inspired by OMG's XML Metadata Interchange (XMI) specification.

XML is an open, platform independent and vendor independent standard. It supports the international character set standards of ISO 10646 and Unicode. The XML standard is programming language neutral and API-neutral. A range of XML APIs are available, giving the programmer a choice of access methods to create, view, and integrate XML information.The cost of entry for XML information providers is low. XML's tag structure and textual syntax make it as easy to read as HTML, and it is clearly superior for conveying structured information. The cost of entry for automatic XML document producers and consumers is also low. A growing set of tools is available for XML development.

XMI is an XML based exchange standard for exchange of object-oriented metadata models. The purpose of XMI is to allow exchange UML models between modelling tools in a vendor neutral way. It is based on OMG's Meta Object Facility and on CORBA data types. XMI can in theory be used to exchange data based on UML models, but are not primarely designed for this purpose. This standard is therefore designed based on the princples of XMI, but simplified and adapted to suit the needs of this family of standards. And thus more specialised to allow exchange of data based on UML directly.

Clause C.2 and C.3 give introductions to XML and XMI, respectively. Clause C.4 outlines some of the differences between XMI and this standard and clause C.6 gives some references for further reading.

C.2 Extensible Markup Language

C.2.1 General

The Extensible Markup Lanugage [XML] is a subset of SGML ISO 8879:1996. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. It is a new format designed to bring structured information to the Web. It is in effect a Web based language for electronic data interchange. XML is an open technology of the World Wide Web Consortium [W3C].

XML defines a class of data objects called **XML documents**. A software module called an **XML processor** is used to read XML documents and provide access to their content and structure. XML documents contain structures of matched tag pairs (aslo called markup) containing nested tags and data. In combination with its advanced linking capabilities, XML can encode a wide variety of information structures. The rules which specify how the tags are structured are called a Document Type Declaration or DTD. XML is in that respect similar to UML and has a declarative model (DTD) and an instance model (XML document).

XML documents comes in two flavors, i.e., **well-formed** XML documents and **valid** XML documents. An XML document is well-formed if it conforms to the XML standard, and if it contains exactly one root element and any number of content elements where the elements, delimited by start- and end-tags, nest properly within each other. A XML document is valid if it is well-formed and if it conforms to its DTD. An XML processor can be *non-validating* or *validating.* That means that a *non-validating* XML processor only checks if the XML doucment is well-formed. A *validating* XML processor checks whether the XML document conforms with its DTD as well.

An XML document can consist of one or more storage units which makes out the document's physical structure. These storage units are called entities. Each **XML entity** has a name and a content. The general idea is that if you quote the name of the entity in the logical structure of the XML document you receive the corresponding content.

This quoting mechanism is called an **entity reference**. There are two types of entities: text and binary. A text entity contains text data that is considered to form part of the XML document. An XML document is considered to be a text entity. A binary entity is basically anything that isn't to be treated as though it is XML-encoded. Each binary entity needs to have a notation associated with it to indicate the type of binary encoding used, for example gif, plain-text or PDF.

C.2.2 XML element

The logical structure of an XML document comprises of properly nested XML elements and entity references. An **XML element** may have attributes and a content. An element always has a start tag which may include the element's attributes, a content and an end tag. The content of an element is called a content model. This can be empty, a sequence of elements, one of an alternative list of elements, repetitions of elements, plain text or mixed elements and text data. An example of a simple XML element called **Road** with no attributes and plain text as content is:

<Road>Route 66</Road>

Here we see that "<Road>" is the start tag, "Route 66" is the content of the element, and "</Road>" is the closing tag of the element. A more complex XML element is an element with attributes and two elements as content:

```
<Person id="convenor1">
  <FirstName>Olaf</FirstName>
  <LastName>&Oslash;stensen</LastName>
</Person>
```

Here we have an XML element called **Person** with an attribute **id** with value "convenor1". The content of the **Person** element is two XML elements called **FirstName** and **LastName** which has plain text as their content model. The entity reference "Ø" is refering to an XML text entity which content is the Latin 1 character "Ø".

An XML element can point to another element within the XML document or to external resources by using special purpose XML attributes. These XML elements are called linking elements. Linking elements point to a target resource through a Universal Resource Identificator (URI) reference. Declaring XML elements with these special purpose XML attributes indicate their behavior for the XML processor. See the next clause for an overview of some of these special purpose XML attributes. Here is an example of an XML element that points to another XML document:

<Reference xml:link="simple" href="http://www.ex.org/AnnexA.xml">See Annex A</Reference>

Here the special purpose attribute **xml:link** indicates that this is a linking element and value of the attribute **href** is a URI pointing to another XML document.

C.2.3 Document Type Declaration (DTD)

A Document Type Declaration (**DTD)** declares the valid XML elements, their structure, and the XML entities that can be used by a class of XML documents. An XML document can have an external DTD subset defined in a separate file and/or an internal DTD subset defined as a part of the header information of the XML document. If an XML document contains both an external subset and an internal subset, the internal subset is considered to occur before the external subset. In the following DTD declarations are given with examples on how instances in the XML document will look like.

An XML element is declared using the special DTD start tag "<!ELEMENT":

<!ELEMENT Road (#PCDATA)>

<Road>Route 66</Road>

Here we define an XML element named **Road** with text as the content model. This is indicated by the XML reserved keyword "#PCDATA", which is short for parsed character data. The content model can be empty, a

sequence or a choice of specific child elements, any combination of elements, or a mixture of text and specific child elements. A multiplicity operator can be used to specify the allowed occurences of the child elements. The multiplicity operators are (?) for zero-or-one, (+) for one-or-more and (*) for zero-or-more occurences. Absence of the multiplicity operator means that the child element must appear exactly once. Here is an example on the use of multiplicity operators:

```
<!ELEMENT parent ( c1, c2?, c3+, c4* )>

    <parent>
      <c1> ... </c1>
      <c3> ... </c3>
      <c3> ... </c3>
      <c4> ... </c4>
    </parent>
```

The parent XML element shall contain the four child XML elements c1, c2, c3 and c4, in that particular order. The element c1 must always occur, c2 is optional, there can be one-or-more c3 elements and zero-or-more c4 elements. The example shows one c1, no c2, two c3 and one c4 elements.

An XML element with a choice of child elements is declared as follows:

```
<!ELEMENT car_part (door | wheel | engine)>

    <car_part>
      <wheel> ... </wheel>
    </car_part>
```

Here a car_part element can contain either a door, a wheel or an engine, but not more than one child element.

A mixed content model allows a mixture of text and special child elements:

```
<!ELEMENT mp (#PCDATA | Road)*>

    <mp>Here we can mix text and Road elements. <Road>E6</Road> See!</mp>
```

The ANY keyword specifies that the content model of an XML element can be anything:

```
<!ELEMENT p ANY>

    <p>Anything goes <Road>E6</Road>here<tag/>!</p>
```

If an XML element is declared to be empty it shall not have a closing tag in the XML documents. This is indicated by a slash at the end of the start tag.

```
<!ELEMENT tag EMTPY>

    <tag/>
```

An XML element can have attributes. The attributes are declared in an XML attribute list statement. XML has three groups of attribute types: string types, tokenised types and enumeration types. An attribute can either be mandatory or optional indicated by the #REQUIRED and #IMPLIED statements in the attribute list declaration. There is also possible to give default values for attributes. An attribute is declared in an attribute list construct:

```
<!ELEMENT point EMPTY>
<!ATTLIST point
          id                ID              #REQUIRED
          dim       (oneD | twoD | threeD) #IMPLIED "twoD"
          x                 CDATA           #REQUIRED
          y                 CDATA           #IMPLIED
          z                 CDATA           #IMPLIED>
```

Here we have defined an XML element named point. The point has an attribute list of five attributes. The attribute called **id** has a tokenised type ID and it is required. The **dim** attribute has an enumerated type that takes one out of three valid values. This attribute is implied, but it has a default value of "twoD". The **x**, **y** and **z** attributes all are of string type "CDATA", which is short for character data. Since XML does not have any data types for numeric values the coordinate values must be converted to strings. Notice that only **x** is required. Two instance of this XML element are:

<point id="i01" x="12300" y="234"/>
<point id="i02" dim="threeD" x="123456" y="-234567" z="50"/>

For the first instance with **id** equal "i01" we can use the implied default value of **dim** to indicate that this point is two dimensional thus do not have to give this explicit. The second instance has all attributes filled in.

The tokenised attribute types specified in XML are:

XML Type	*Semantics*
ID	An identifier for the element that, if specified, must be unique within the XML document. The value of the identifier must always start with a letter, '_' or ':'. An XML element can only have one attribute of type ID.
IDREF	A reference to an XML element in the XML document. The value must correspond to an attribute value of type ID in an existing XML element.
IDREFS	A reference to one or more XML elements. The values must be separated by spaces and must correspond to existing XML element ID's.
ENTITY	A reference to an external entity. The value must be a legal entity name.
ENTITIES	A reference to any number of entity names, where the entity names is separated by spaces.
NMTOKEN	A NMTOKEN (Name Token) is any mixture of characters.
NMTOKENS	Any number of NMTOKENs separated by spaces.

Some attributes are specific for XML and have reserved names and well-defined semantics. The **xml:lang** attribute can be used to indicate language used in XML elements and **xml:space** can be used to indicate how to handle whitespace within elements.

Attribute name	*XML Type*	*Semantics*
xml:lang	NMTOKEN	A special attribute that may be inserted in documents to specify the language used in the contents and attribute values of any element in a XML document. But it must be defined in the attribute list specification of the actual element.
xml:space	(default | preserve)	A special attribute that signals an intention that in that element, white space should be preserved by applications.

C.2.4 Linking element

Linking elements are recognised based on the use of a designated attribute named xml:link and a set of accompanying attributes. This is described in the XML XLink specification [XLink]. A **link** is an explicit relationship between two or more data objects or portions of data objects. The content of the linking element is called the **local resource** and the target of the link is called the **remote resource**. A remote resource is identified by a text string called a **locator**. A locator value may contain either a Uniform Resource Identifier (URI) or a fragment identifier, or both. The syntax of a locator is first the URI, followed by a **connector** ("#" or "|") and a fragment identifier. The URI describes a remote resource, and the fragment identifier describes a sub-resource within that resource. A **fragment identifier** pointing in to a XML document must be an XPointer [XPointer]. If the connector is "#", this signals that the remote resource is to be fetched as a whole, and that the XPointer processing to extract the sub-resource is to be performed on the client. If the connector is "|", no intent is signalled as to what processing model is to be used for accessing the designated resource.

The following information can be associated with a link and its resources: One or more locators to identify the remote resources participating in the link (a locator is required for each remote resource), the semantics of the link, the semantics of the remote resources and the semantics of the local resource. Example of an linking element is:

<link xml:link="simple" href="http://www.ex.org/data.xml|id(i005)" >This is the local resource</link>

Here we have a linking element called **link**. It has two attributes **xml:link** and **href**. **xml:link** states that this is a linking element, whereas **href** holds the locator identifying the remote resource. The locator consist of an URI which is "http://www.ex.org/data.xml" and a fragment identifier "id(i005)" in XPointer syntax. We could also have used "i005" directly, it is defined as a shortcut in the XPointer specification. The content of the linking element is the local resource. Other combinations can be:

<link xml:link="simple" href="|id500">This link points to a element within the document with an attribute of type ID with value "id500"</link>

<link xml:link="simple" href="xdata.xml">This link points to an XML document</link>

The XLink specification defines the following attributes:

Attribute name	*XML Type*	*Semantics*
xml:link	CDATA	This is a special reserved attribute that indicates that the element shall act as a linking element. Legal values are: simple, extended, locator, group, document. We will only use simple links!
href	CDATA	The value of the href attribute in linking elements shall always contains a locator which identifies a resource, e.g., by an URI-reference or by an XPointer specification.
inline	(true | false)	The inline attribute specifies the first part of a link's semantics. A link is either inline or out-of-line. Since we will only use simple links this attribute shall always be true.
role	CDATA	The role attribute also specifies a part of the link's semantics. The value of this attribute identifies to the application software the meaning of the link. This allows the application to show different symbols for the different kinds of links.
title	CDATA	This is the title shown to the user for the remote resource.
show	(embed | replace | new)	This attribute indicates the behavior policies to use when the link is traversed for the purpose of display or processing. The *embed* value indicates that the designated resource should be embedded in the body of the resource and at the location where the traversal started. The *replace* value indicates that the designated resource

		should replace the resource where the traversal started. The *new* value indicates that the designated resource should be displayed or processed in a new context.
actuate	(auto | user)	The actuate attribute is used to express a policy as to when the traversal of a link should occur. The *auto* value indicates that the resource

C.2.5 XML entity

XML entities are divided into text and binary entities. There is also a distinction between internal entities and external entities. An internal entity has a value associated with it as part of the entity declaration. An external entity associates a name with a physical storage unit (file name). In the following we define an XML entity named **XML** which is an internal text entity:

```
<!ENTITY XML 'Extensible Markup Lanuage'>
<!ELEMENT p (#PCDATA) >

    <p>This is a written in XML (&XML;).</p>
```

Only text entities that can be parsed as XML can be referenced directly in the XML document. Binary entities do not contain valid XML and must therefore be referenced in an XML element's attribute of the ENTITY type. Here we define an external binary entity named "my.sign" and refer to it in the **src** attribute of an **image** element.

```
<!ENTITY my.sign SYSTEM "image/signature.gif" NDATA GIF>
<!ELEMENT pi ANY>
<!ELEMENT image EMPTY>
<!ATTLIST image
        src             ENTITY      #REQUIRED >

    <pi>This is my signature: <image src="my.sign"/></pi>
```

There is a special construct that only can be used in the DTD called a parameter entity. A **parameter entity** can be used as a shortcut for commonly occurring structures. A parameter entity declaration consist an entity declaration with a "%" sign before the parameter entity name. The string within the apostrophes will be substituted by the XML processor when it reads the DTD. An example of a declaration of a parameter entity called "GI.Boolean.att" and its usage is as follows:

```
<!ENTITY % GI.Boolean.att '( true | false )' >

<!ELEMENT my_Boolean_element EMPTY>
<!ATTLIST my_Boolean_element %GI.Boolean.att; >
```

C.2.6 Character coding

Each XML text entity must declare which character encoding scheme that it uses internally. External parsed entities in an XML document may use different encoding schemes for its characters than used in the root document. All XML processors must accept the UTF-8 and UTF-16 encodings of ISO 10646 as a minimum. Parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with an encoding declaration in the document entity:

```
 <?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
```

According to the XML recommendation the following values are allowed:

- For ISO/IEC 10646 and UNICODE based encodings: "UTF-8", "UTF-16", "ISO-10646-UCS-2" and "ISO-10646-UCS-4" shall be used. If an XML text entity is encoded in UCS-2, it must start with an appropriate encoding signature, the Byte Order Mark, which is the character with hexadecimal value FEFF.

- For ISO/IEC 8859: "ISO-8859-1", "ISO-8859-2", ... , "ISO-8859-10" can be used.

- For various encodings of JIS X-0208-1997: "ISO-2022-JP", "Shift_JIS" and "EUC-JP" can be used.

For ISO 15046 we will restrict this to "UTF-8", "UTF-16", "ISO-10646-UCS-2" and "ISO-10646-UCS-4".

Alternatively one can reference any character in ISO/IEC 10646 by quoting its character number in a character reference regardless of which encoding scheme used. Or one can declare a text entity that represents the character in question. There are two ways of referring characters directly, by decimal representation or by hexadecimal reference. The hexadecimal reference for the less-than-sign '<' is:

<

And the decimal representation of the same sign is:

<

C.2.7 XML document header

All XML documents must start with a processing instruction that specifies that this is a XML document and which version of the XML standard used. Information about the character encoding, if it is not "UTF-8" or "UTF-16", can be included in the header also. An example is as follows:

<?xml version="1.0" encoding="UTF-8" ?>.

The next element shall be the document type declaration element:

```
<!DOCTYPE top SYSTEM "root.dtd" [
    <!ELEMENT bot EMPTY> ] >
```

Here we see a document type element declaration that has both an external subset and an internal subset DTD. The root XML element is called **top** and it must be defined in the external DTD subset declared in the "root.dtd" file. The internal DTD subset is declaring a single XML element called **bot**.

C.2.8 Miscellaneous

An XML comment may appear anywhere in the document, but outside other markup. Comments are not part of the document's character data. A comment starts with the string '<!--' , followed by any characters except the string '--' and ends with the string '-->'. An example of a comment is:

<!-- This is a comment. -->

All of an XML document is case sensitive, both markup and text. This is different from SGML and HTML and it was introduced to allow markup in non-Latin alphabet characters and to avoid the problem with case folding. This means that element type names, entity names and attribute names all are case sensitive. The following elements are different and thus allowed in an XML document:

```
<road>This is a road element</road>
<Road>This is a Road with a capital R</Road>
```

C.2.9 Other XML standards

The base XML standard [XML] is associated with a number of supporting standards. The most relevant standards are shortly described in this clause.

A **link** is an explicit relationship between two or more data objects or portions of data objects. In addition to the IDREF mechanism in XML documents a link can be constructed with attributes on XML elements using a combination of the XLink and XPointer specification. The **XML Linking Language** [XLink] and the **XML Pointer Language** [XPointer] specifies constructs to describe links between both external and internal objects of XML documents. XLink is used to create simple unidirectional hyperlinks between objects in separate XML documents as well as more sophisticated links. Whereas XPointer is used for addressing internal structures within XML documents, such as references to elements, character strings and other parts of XML documents, whether or not they have an explicit ID attribute.

The **Namespace in XML** specification [Namespace] provide a simple method for qualifying element and attribute names used in XML documents by associating them with a namespace identified by a URI references. An **XML namespace** is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. This allows XML documents to mix XML elements and attributes from more than one DTD that may have identical names and different semantics. The mechanism for achieving this is to use qualified names for both XML elements and attributes. A **qualified name** consist of a namespace prefix, a single colon as separator, followed by the local name. Here is an example of the use of the XML namespace mechanism:

```
<x xmlns:gis="http://www.ex.org/schema/spatial.dtd">
  <gis:point>234 445 150</gis:point>
  <point>23 55</point>
</x>
```

Here we see **x** contain two elements named point. The local **point** element is defined in the DTD of the XML document, but the **gis:point** element is defined in a DTD located at URI "http://www.ex.org/shema/spatial.dtd". The special purpose attribute **xmlns:gis** in element **x** defines **gis** as a namespace prefix of the declarations defined in the target DTD.

The **Resource Description Framework** (RDF) is a result of the W3C Metadata Activity. RDF is the foundation for processing and exchanging machine-understandable metadata on the Web using XML as the exchange format. It can be used in a variety of application areas: in resource discovery, in cataloguing for describing the content and content relationships available at a particular Web site, in content rating, in describing collections of pages that represent a single logical document, for describing intellectual property rights of Web pages, and for expressing the privacy preferences of a user as well as the privacy policies of a Web site. The **RDF Model and Syntax Specification** [RDF] document defines a data model for representing named properties and property values. This data model consist of three object types: Resource, Property and Statement. Instances of the RDF data model is called a RDF Schema. RDF resources are things of interest, e.g., an entire Web page, an XML element or an entire Web site. An RDF property is a specific aspect, characteristic, attribute or relation used to describe a resource. Whereas an RDF statement is a specific RDF resource in combination with a named RDF property for that resource. RDF can be used to express a wide variety of data models, e.g., Entity-Relationship models. The **RDF Schema Specification** [RDFSchema] document defines a schema specification language that provides a basic type system for use in RDF models. It defines resources and properties such as class and sub-class-of constructs that can be used in application specific schemas. RDF Schemas can be compared with XML DTDs, but unlike an XML DTD, which gives specific constraints on the structure of an XML document, an RDF Schema provides information about the interpretation of the statements given in an RDF data model.

The **Document Content Description** [DCD] document is a note to W3C which proposes a structural schema facility for XML, which may be used in the same way as the current XML DTD mechanism. DCD is designed for describing constraints to be applied to the structure and content of XML documents. It provides mechanism for defining XML elements, attributes and their logical structure as well as document constraints and basic datatypes. DCD is a RDF vocabulary, that means that it is based on the above mentioned RDF specifications.

The **Schema for Object-oriented XML** [SOX] is a note to W3C which proposes an alternative to XML DTDs. SOX provides basic data types, extensible data typeing mechanism, content model and attribute interface inheritance. A SOX document, or schema, is a valid XML document that represent a complete XML DTD-like structure.

The **Vector Markup Language** [VML] is a note to W3C which defines a format for encoding of vector information together with additional markup to describe how that information may be displayed and edited. The VML defines XML elements for vector graphic information and uses a stylesheet mechanism to determine the layout of the

vector graphics. VML is supported by Microsoft, Autodesc, Visio and Hewlet-Packard and is implemented in Internet Explorer 5.0 beta 3.

The **Extensible Stylesheet Language** [XSL] is intended to control the appearance of XML documents. XSL is a language for expressing stylesheets. A stylesheet expresses rules for presenting a class of XML documents. Thus a stylesheet contains descriptions on how XML elements can be rendered by an XML browser. XSL views XML documents as a tree and uses a two stage presentation process. First, the result tree is constructed from the source tree. Second, the result tree is interpreted to produce formatted output on display, on paper, in speech or onto other media.

The **Document Object Model** [DOM] specification defines a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of XML documents. The DOM provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. This means that vendors can support the DOM as an interface to their proprietary XML processors. The document defines three language bindings: IDL, Java and ECMA Script Language.

C.3 XML Metadata Interchange (XMI)

C.3.1 General

The Object Management Group [OMG] has created a XML based specification for interchange of metadata, called XML Metadata Interchange [XMI]. The main purpose of XMI is to enable interchange of metadata between UML based modelling tools and MOF based metadata repositories. The specification is based on OMG's Meta Object Facility (MOF), which is an OMG metamodelling and metadata repository standard, and on OMG's Unified Modeling Language (UML), which is an OMG modelling standard. The XMI specification defines the design principles for generating XMI based DTDs and XML documents. It consist of a set of **XML DTD production rules** for transforming MOF based metamodels into XML DTDs, a set of **XML document production rules** for encoding and decoding MOF based metadata, and concrete DTDs for UML and MOF. The concrete DTDs are generated based on the XML DTD and XML production rules.

The Meta Object Facility [MOF] is OMG's adopted technology for defining metadata and representing it as CORBA objects. In OMG's terminology *metadata* is a general term for data that in some sense describes information. The term *model* is generally used to denote a description of something, typically something in the real world and in the MOF context, the term model is any collection of metadata that is related. Metadata that describes metadata is called *meta-metadata*, and a model that consist of meta-metadata is called a metamodel.

The **MOF Model** is the MOF's built-in meta-metamodel. It is defined as the top layer of a four-layer metamodelling architecture, where each layer defines the language for specifying models at the layer below, see table C.1. The top layer of this architecture is called the meta-metamodel. The next layer is called the metamodel and the UML metamodel is an example of a model at this layer. The metamodel layer defines the language for the next layer, which is called the model layer. An example of a model at the model layer is any UML model, e.g., the RoadMap application schema in Annex E and the Spatial schema. The model layer defines the language for the lowest layer called the user object layer. Examples of models at this level are datasets which conforms to the models at the next higher layer.

Table C.1 — OMG's four-layer metamodelling architecture

Meta-level	*MOF terms*	*Examples*
M3	meta-metamodel	MOF Model
M2	metamodel	UML metamodel
M1	model	UML model (RoadMap)
M0	user objects (data)	Instances

The MOF Model and the **UML metamodel** are closely aligned in their modelling concepts. The main metadata modelling constructs are: **Class**, **Association** and **Package**.

— Classes can have Attributes and Operations at both object and class level. Attributes are used to represent metadata. Operations are provided to support metamodel specific functions on the metadata. Both Attributes and Operations Parameters may be defined as ordered, or as having structural constraints on their cardinality and uniqueness. Classes may multiply inherit from other Classes.

— Associations support binary links between Class instances. Each Association has two AssociationEnds that specify ordering or aggregation semantics, and structural constraints on cardinality or uniqueness. When a Class is the type of an AssociationEnd, the Class may contain a reference that allows navigability of the Association's links from a Class instance.

— Packages are collections of related Classes and Associations. Packages can be composed by importing other Packages or by inheriting from them. Packages can also be nested.

Other significant MOF Model constructs are **DataType** and **Constraint**. DataTypes allow the use of non-object types for Parameters or Attributes. In the OMG MOF specification, these must be data types or interface types expressible in CORBA IDL. Constraints are used to associate semantic restrictions with other elements in a MOF metamodel. This defines the well-formedness rules for the metadata described by a metamodel. Any language may be used to express Constraints, though there are obvious advantages in using a formal language like OCL.

C.3.2 DTD and XML document production

The XMI specification defines DTD and XML production rules that can be used to transfer any models described by a MOF metamodel, i.e., any metamodel that is defined in the abstract language of the MOF Model. This is illustrated in table C.2. A M2 level metamodel such as the UML metamodel can be encoded against the XML DTD for the M3 level MOF Model. And a M1 level model, a UML model, can be encoded against the XML DTD for the M2 level UML metamodel.

Table C.2 — XMI and OMG's metadata architecture

Meta-level	*Models*	*XML DTDs*	*XML documents*
M3	MOF Model	XMI based MOF DTD	
M2	UML metamodel	XMI based UML DTD	XMI based MOF metamodel documents
M1	UML models		XMI based UML model documents
M0	Instances		

XMI defines a number of XML elements that must be included in the DTDs generated. Some of these XML elements contain metadata about the metadata to be transferred, for example, the identity of the metamodel associated with the metadata, the time the metadata was generated, the tool that generated the metadata, whether the metadata has been verified, etc. All XML elements defined have the prefix "XMI." to avoid name conflicts with XML elements that would be a part of a metamodel. Thus, XMI does not make use of XML Namespace, because this is not an W3C recommendation yet, but states that it may be possible to place all required XML elements in a namespace. Every XML element that corresponds to a metamodel class must have attributes that enable the XML element to act as a proxy for a local or remote XML element. These attributes are used to associate an XML element with another XML element. Most of the XML attributes defined have the prefix "xmi.".

Every metamodel class is represented in the DTD by an XML element whose name is the class name. The content model of the element lists the attributes of the class; references to association ends relating to the class; and the

classes that this class contains, either explicitly or through composition associations. Every attribute of a metamodel is represented in the DTD by an XML element whose name is the attribute name. The attributes are listed in the content model of the XML element corresponding to the metamodel class in the order they are declared in the metamodel. Each association (both with and without containment) between metamodel classes is represented by two XML elements that represents the roles of the association ends. The multiplicities of the association ends are translated to the XML multiplicities that are valid for specifying the content models of XML elements. The content model of the XML element that represent the container class has an XML element with the name of the role at the association end, with the multiplicity defined for its association end. The XML element representing the role has a content model that allows XML elements representing the associated class and any of its subclasses to be included.

XMI defines a mechanism for extending a metamodel class. Any number of **extension** elements are included in the content model of any class. These XMI extension elements have a content model of ANY, allowing considerable freedom in the nature of the extensions.

XMI provides mechanisms for specifying **differences** between documents so that an entire document does not need to be transmitted each time. The XMI specification does not specify an algorithm for computing the differences, just a form for transmitting them. Thus only the model changes that occur can be transmitted. XMI defines the following three types of differences, and the changes they represent:

- Delete: The delete operation refers to a particular element of the old model and specifies a deep removal of the referenced element and all of its contents.
- Add: The add operation refers to a particular element of the old model and specifies a deep addition. The element and its contents are added. The contents of the new element are added at the optional position specified, the default being as the last element of the contents.
- Replace: This operation deletes the old element, but not its contents. The new element and its contents are added at the position of the old element. The original contents of the old element are then added to the contents of the new element at the optional position specified, the default being at the end.

XMI also can be used to transmit incomplete models or model fragments. An incomplete model is a model which may be missing some information, while maintaining the same structure required for valid models.

Every XMI based DTD consist of the following declarations:

- An XML version processing instruction and optional encoding declaration. Example: <?xml version="1.0" encoding="UCS-2" ?>
- Any other valid XML processing instructions
- The required XMI declarations
- Declarations for a specific metamodel
- Declarations for differences
- Declarations for extensions

Every XMI based XML document consist of the following:

- An XML version processing instruction with an optional encoding declaration.
- Any other valid XML processing instruction
- An optional external DTD declaration with an optional internal DTD declaration. Example: <!DOCTYPE XMI SYSTEM "http://www.xmi.org/xmi.dtd">
- Any XML elements that conform to the XMI based DTD.

C.4 Design differences

TO BE COMPLETED!

Differences from the XMI approach, Why?

Metadata vs. data

Data types vs. CORBA IDL

meta-metamodel vs. UML model

extension?

differences?

XMI.reference?

Names, fully qualified names, shortname lookuptable.

Incomplete models?

C.5 References

[FAQ] Frequently Asked Questions about the Extensible Markup Language, Version 1.41 (6 October 1998), http://www.ucc.ie/xml

[XML] Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998. http://www.w3.org/TR/REC-xml

[W3C] The World Wide Web Consortium (W3C). The standards group responsible for maintaining and advancing HTML and other Web related standards. http://www.w3.org/

[XLink] XML Linking Lanugage (XLink), W3C Working Draft 03-March-1998. http://www.w3.org/TR/WD-xlink

[XPointer] XML Pointer Language (XPointer), W3C Working Draft 03-March-1998, http://www.w3.org/TR/WD-xptr

[Namespace] Namespaces in XML, W3C Working Draft, 17-November-1998, http://www.w3.org/TR/PR-xml-names

[RDF] Resource Description Framework (RDF) Model and Syntax Specification, W3C Proposed Recommendation, 05-January-1999, http://www.w3.org/TR/PR-rdf-syntax

[RDFSchema] Resource Description Framework (RDF) Schema Specification, W3C Working Draft 30-October-1998, http://www.w3.org/TR/WD-rdf-schema

[DOM] Document Object Model (DOM) Level 1 Specification, Version 1.0, W3C Recommendation 01-October-1998, http://www.w3.org/TR/REC-DOM-Level-1

[XSL] Extensible Stylesheet Language (XSL), Version 1.0, W3C Working Draft 16-December-1998, http://www.w3.org/TR/WD-xsl

[DCD] Document Content Description for XML, W3C NOTE, 31-July-1998, http://www.w3c.org/TR/NOTE-dcd

[SOX] Schema for Object-oriented XML (SOX), W3C NOTE, 30-September-1998, http://www.w3c.org/TR/NOTE-SOX

[VML] Vector Markup Language (VML), W3C NOTE, 13-May-1998, http://www.w3c.org/TR/NOTE-VML

[OMG] The Object Management Group, http://www.omg.org/

[XMI] XML Metadata Interchange (XMI), Proposal to the OMG Object Analysis & Design Task Force RFP 3: Stream-based Model Interchange Format (SMIF), Joint Submission, OMG Document ad/98-10-05, October 20, 1998, http://www.omg.org/techprocess/meetings/schedule/Stream-based_Model_Interchange.html

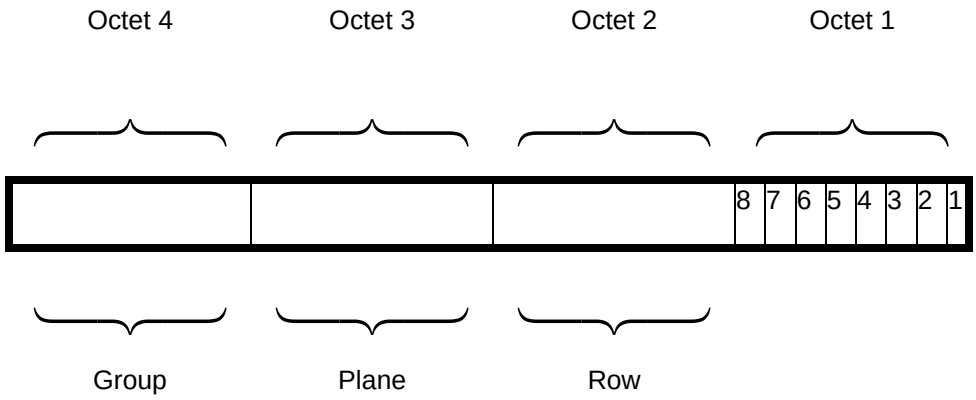
[MOF] OMG's Meta Object Facility (MOF) Specification, ad/97-08-14, http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#MOF_Specification

Annex D
(informative)

Character Repertoire (ISO/IEC 10646)

All text is defined in terms of character set code tables. The most comprehensive international character set is the Universal Multiple-Octet Character Set (UCS) in its 4 octets version (ISO/IEC 10646, UCS-4 version[1]). The ISO/IEC 10646 standard defines a "Universal Character Set" for virtually all languages in the world.

ISO/IEC 10646 organise its code tables in 256 groups of 256 planes of 65 536 characters. Each plane is itself arranged in 256 rows of 256 characters. The first edition of ISO/IEC 10646 (1993) defined plane 0, which can be represented with 2 octets only, in its UCS-2 version[2] (the table of which correspond with the UNICODE industry standard).



ISO/IEC 8859-1 (Latin-1) : row 0

ISO/IEC 10646-1 and UNICODE : plane 0

Others : to be determined

Figure D.1 — UCS-4 structure

---

[1] UCS-4 can virtually handle all languages covered by UCS-2, plus the **extended** Han script (roughly 70 000 ideographs). Every ancient as well as modern writing system is specified. UCS-4 is required for such character sets as ancient Egyptian hieroglyphics.

[2] UCS-2 corresponds to the Basic Multilingual plane (Plane 0) of ISO/IEC 10646 which can virtually handle all Latin alphabet languages, Greek, Hebrew, Cyrillic, Arabic, **part of** the Han script (roughly 30 000 ideographs), Korean Hangul, Japanese Katakana, etc.

The first row of 256 characters is based the Latin alphabet number 1 (ISO/IEC 8859-1[3]) which was historically based on a structure of codes defined in terms of row and columns, where the row was represented by the first 3 or 4 bits and the column by the last 4 bits of each bit combination of 7-bit or 8-bit character sets (forming pages of 94 or 96 graphic character sets). 32 bit combinations were reserved per page in this schema for control character allocation. In the UCS world, only the first row of 256 characters reserves 64 bit combinations for control characters.

Some of the control characters are reserved for specialised use, such as transmission control in an asynchronous communications system or application level delimiting such as is used by ISO/IEC 8211. These characters are not to be used as part of text data. The only format effecting control characters required as part of the primitive data element text coding in TC211 115046-18 Encoding are: Carriage Return (CR), Line Feed (LF), Back Space (BS), Horizontal Tab (HT), Vertical Tab (VT) and Form Feed (FF).

ISO/IEC 10646 defines mechanisms for creating composite characters. Composite characters are characters that are composed by a base character that is modified by superimposing it with one or more additional characters. ISO/IEC 10646 also defines a set of precomposed characters and their defined decomposition. For example ö has the defined decomposition o¨. Since mixing composite characters with their precomposed equivalents may lead to interpretation problems it is deprecated to use a composite character if a precomposed character exist. That is the precomposed character shall always be used.

ISO/IEC 10646 defines mechanisms for creating composite characters. Composite characters are characters that are composed by a base character that is modified by superimposing it with one or more additional characters. ISO/IEC 10646 also defines a set of precomposed characters and their defined decomposition. For example ö has the defined decomposition o¨. Since mixing composite characters with their precomposed equivalents may lead to interpretation problems it is deprecated to use a composite character if a precomposed character exist. That is the precomposed character shall always be used.

In the simple octet character set realm, the use of localised character sets requires complex code extension techniques described in ISO/IEC 2022. In the context of ISO 15046, the basic character set in use is UCS-4. As a fallback, UCS-2 can be used. Only in the limited context of a part of Western Europe and a part of the Americas should the ISO/IEC 8859-1 character set be used. It is recommended that UCS-2 should be the minimal character set. Provisions shall be taken so that any application be able to exchange UCS-4 character data.

[3] ISO 8859-1 corresponds to the Latin alphabet number 1 (including accented letters for a limited support of Western Latin alphabet based languages [French, one of the three official languages of ISO, is partly supported, for example]).

Annex E
(informative)

Examples

E.1 Introduction

TO BE COMPLETED

Introduction to the Examples Annex. The number of examples.

Some pointers for XML software at XMLSOFTWARE.COM -> http://www.xmlsoftware.com/

Some pointers to international examples are found at XMLINFO.com: Chinese, Korean, Japanese, German, French ... http://www.xmlinfo.com/international/

E.2 A simple RoadMap example

E.2.1 Application schema

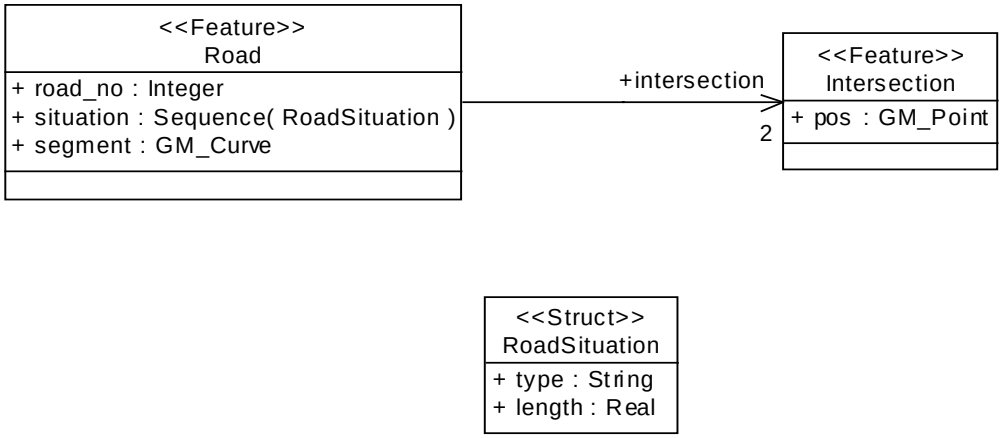


Figure E.1 — RoadMap application schema

E.2.2 Example data

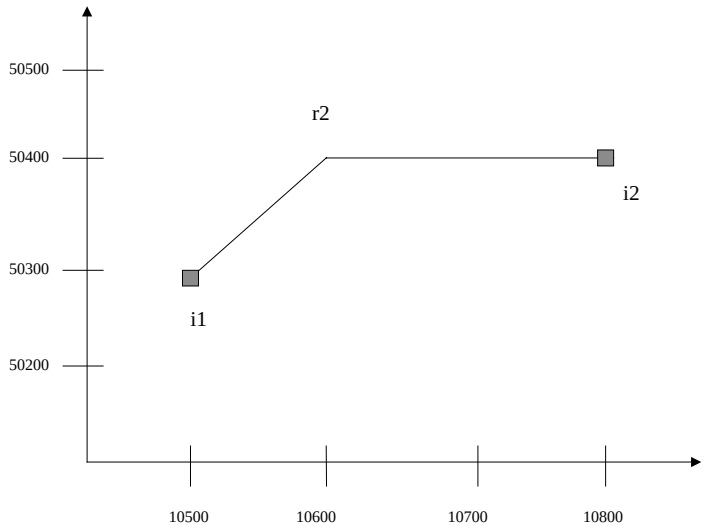


Figure E.2 — Simple map based on the RoadMap application schema

Here we have some simple data: One road and two intersections. The data is organised according to the instance model described in clause 8.

RoadMap
i01: Road
properties:
road_no: "66"
associations:
intersection: { i05, i06 }
compositions:
situation: { i02, i03, i04 }
segment: i09

i02: RoadSituation
properties:
type: "gravel"
length: "159.5"

i03: RoadSituation
properties:
type: "asphalt"
length: "30.5"

i04: RoadSituation
properties:
type: "concrete"
length: "160.0"

i05: Intersection
compositions:
pos: i07

i06: Intersection
compositions:
pos: i08

Spatial
i07: GM_Point
properties:
position: "50400 10800"
associations:
SRS: i11

i08: GM_Point
properties:
position: "50300 10500"
associations:
SRS: i11

i09: GM_Curve
associations:
SRS : i11
compositions:
segment : { i10 }

i10: GM_CurveSegment
properties:
controlPoint: { "50300 10500", "50400 10600", "50400 10800" }
controlParameter: { "1.0", "1.0", "1.0" }
interpolationMethod: "Linear"

Spatial reference by coordinates
i11: SpatialReferenceSystem
properties:
type: "UTM"

E.2.3 Document Type Definition (roadmap.dtd)

```
<!-- DTD file: roadmap.dtd -->

<!-- Import fixed elements from file called "iso15046/fixed.dtd" -->

<!ENTITY % GI.fixed.elements SYSTEM "iso15046/fixed.dtd" >
%GI.fixed.elements;

<!-- Root element -->

<!ELEMENT GI.data (RoadMap, Spatial) >


<!-- PACKAGE RoadMap -->

<!ELEMENT RoadMap (RoadMap.Road | RoadMap.Intersection)* >


<!-- CLASS Road -->

<!ELEMENT RoadMap.Road.road_no %GI.Integer.cont; >
<!ENTITY RoadMap.RoadProperties ' RoadMap.Road.road_no ' >

<!ELEMENT RoadMap.Road.intersection ( RoadMap.Intersection) >
<!ENTITY % RoadMap.RoadAssociations 'RoadMap.Road.intersection+' >
```

```
<!ELEMENT RoadMap.Road.situation ( RoadSituation ) >
<!ELEMENT RoadMap.Road.segment ( Spatial.GM_Curve ) >
<!ENTITY % RoadMap.RoadCompositions 'RoadSituation*, RoadMap.Road.segment' >

<!ELEMENT RoadMap.Road ( %RoadMap.RoadProperties;,
                %RoadMap.RoadAssociations;,
                %RoadMap.RoadCompositions; )? >
<!ATTLIST RoadMap.Road
                %GI.element.att;
                %GI.link.att; >


<!-- CLASS Intersection -->

<!ELEMENT RoadMap.Intersection.pos ( Spatial.GM_Point ) >
<!ENTITY RoadMap.IntersectionCompostions ' RoadMap.Intersection.pos' >

<!ELEMENT RoadMap.Intersection (%RoadMap.IntersectionCompositions; ) >
<!ATTLIST RoadMap.Intersection
                %GI.element.att;
                %GI.link.att; >


<-- CLASS RoadSituation STEREOTYPE Struct -- >

<!ELEMENT RoadMap.RoadSituation.type %GI.String.cont; >
<!ATTLIST RoadMap.RoadSituation.type %GI.String.att; >

<!ELEMENT RoadMap.RoadSituation.length %GI.Real.cont; >

<!ENTITY % RoadMap.RoadSituationProperties 'RoadMap.RoadSituation.type,
                RoadMap.RoadSituation.length' >

<!ELEMENT RoadMap.RoadSituation ( %RoadMap.RoadSituationProperties; ) >


<!-- PACKAGE Spatial -->

<!ELEMENT Spatial (GM_Curve | GM_CurveSegmetn | GM_Point)* >


<!-- CLASS GM_Object -->

<!ELEMENT Spatial.GM_Object.SRS (Positioning.SpatialReferenceSystem)>
<!ENTITY % Spatial.GM_ObjectAssociations 'Spatial.GM_Object.SRS'>


<!-- CLASS GM_Curve SUBTYPE OF GM_Object -->

<!ELEMENT Spatial.GM_Curve.coincidentPoint ( Spatial.GM_Point ) >
<!ENTITY % Spatial.GM_CurveAssociations
                '%Spatial.GM_ObjectAssociations;,
                Spatial.GM_Cuve.coincidentPoint*' >

<!ELEMENT Spatial.GM_Curve.segment ( Spatial.GM_CurveSegment ) >
<!ENTITY % Spatial.GM_CurveCompositions 'Spatial.GM_Curve.segment*' >

<!ELEMENT Spatial.GM_Curve (
                %Spatial.GM_CurveAssociations;,
```

```
                %Spatial.GM_CurveCompositions; )? >
<!ATTLIST Spatial.GM_Curve
                %GI.element.att;
                %GI.link.att; >


<!-- CLASS GM_CurveSegment -->

<!ELEMENT Spatial.GM_CurveSegment.controlPoint %GI.DirectPosition.cont;>

<!ELEMENT Spatial.GM_CurveSegment.controlParameter ( GI.Real, GI.String )>
<!ELEMENT Spatial.GM_CurveSegment.interpolation %GI.String.cont; >
<!ATTLIST Spatial.GM_CurveSegment.interpolation %GI.String.att; >

<!ENTITY % Spatial.GM_CurveSegmentProperties
                ' Spatial.GM_CurveSegment.controlPoint+,
                Spatial.GM_CurveSegment.controlParameter*,
                Spatial.GM_CurveSegment.interpolation' >

<!ELEMENT GM_CurveSegment ( %Spatial.GM_CurveSegmentProperties; )? >
<!ATTLIST GM_CurveSegment
                %GI.element.att;
                %GI.link.att; >


<!-- CLASS GM_Point SUBTYPE of GM_Object-->

<!ELEMENT Spatial.GM_Point.position %GI.DirectPosition.cont;>
<!ATTLIST Spatial.GM_Point.position %GI.DirectPosition.att; >

<!ENTITY % Spatial.GM_PointProperties 'GM_Point.position' >
<!ENTITY % Spatial.GM_PointAssociations '%Spatial.GM_ObjectAssociations;' >
<!ELEMENT Spatial.GM_Point (
                %Spatial.GM_PointProperties;,
                %Spatial.GM_PointAssociations; )? >
<!ATTLIST Spatial.GM_Point
                %GI.element.att;
                %GI.link.att; >
```

E.2.4 XML document

```
<?xml version="1.0" ?>
<!DOCTYPE GI SYSTEM "roadmap.dtd" >
<GI gi.version="1.0" timestamp="1999-01-06 12:00" verified="false" containment="nested">
    <GI.header>
      <GI.applicationSchema xml:link="simple" href="http://www.as.org/RoadMap/"
        gi.name="RoadMap" gi.version="1.2">Additional text</GI.applicationSchema>
    </GI.header>
    <GI.content>
      <RoadMap.Road gi.id="i01">
        <RoadMap.Road.road_no>66</RoadMap.Road.road_no>
        <RoadMap.Road.intersection>
          <RoadMap.Road.Intersection gi.idref="i05">
        </RoadMap.Road.intersection>
        <RoadMap.Road.intersection>
          <RoadMap.Road.Intersection gi.idref="i06">
```

```
    </RoadMap.Road.intersection>
    <RoadMap.Road.situation>
      <RoadMap.RoadSituation gi.id="i02">
        <RoadMap.RoadSituation.type>gravel</RoadMap.RoadSituation.type>
        <RoadMap.RoadSituation.length>159.5</RoadMap.RoadSituation.length>
      </RoadMap.RoadSituation>
    </RoadMap.Road.situation>
    <RoadMap.Road.situation>
      <RoadMap.RoadSituation gi.id="i03">
        <RoadMap.RoadSituation.type>asphalt</RoadMap.RoadSituation.type>
        <RoadMap.RoadSituation.length>30.5</RoadMap.RoadSituation.length>
      </RoadMap.RoadSituation>
    </RoadMap.Road.situation>
    <RoadMap.Road.situation>
      <RoadMap.RoadSituation gi.id="i04">
        <RoadMap.RoadSituation.type>concrete</RoadMap.RoadSituation.type>
        <RoadMap.RoadSituation.length>160</RoadMap.RoadSituation.length>
      </RoadMap.RoadSituation>
    </RoadMap.Road.situation>
    <RoadMap.Road.segment>
      <Spatial.GM_Curve gi.id="i09">
        <Spatial.GM_Object.SRS>
          <Position.SpatialReferenceSystem gi.idref="i11"></Position.SpatialReferenceSystem>
        </Spatial.GM_Object.SRS>
        <Spatial.GM_Curve.segment>
          <Spatial.GM_CurveSegment gi.id="i10">
            <Spatial.GM_CurveSegment.controlPoint>50300 10500
            </Spatial.GM_CurveSegment.controlPoint>
            <Spatial.GM_CurveSegment.controlPoint>50400 10600
            </Spatial.GM_CurveSegment.controlPoint>
            <Spatial.GM_CurveSegment.controlPoint>50400 10800
            </Spatial.GM_CurveSegment.controlPoint>
            <Spatial.GM_CurveSegment.controlParameter>
              <GI.Real>10.5</GI.Real><GI.String>Add to c1</GI.String>
            </Spatial.GM_CurveSegment.controlParameter>
            <Spatial.GM_CurveSegment.interpolation>linear
            </Spatial.GM_CurveSegment.interpolation>
          </Spatial.GM_CurveSegment>
        </Spatial.GM_Curve.segment>
      </Spatial.GM_Curve>
    </RoadMap.Road.segment>
  </RoadMap.Road>

  <RoadMap.Intersection gi.id="i05">
    <RoadMap.Intersection.pos>
      <Spatial.GM_Point gi.id="i07">
        <Spatial.GM_Object.SRS>
          <Position.SpatialReferenceSystem gi.idref="i11"></Position.SpatialReferenceSystem>
        </Spatial.GM_Object.SRS>
        <Spatial.GM_Point.position>50300 10500</Spatial.GM_Point.position>
      </Spatial.GM_Point>
    </RoadMap.Intersection.pos>
  </RoadMap.Intersection>

  <RoadMap.Intersection gi.id="i06">
    <RoadMap.Intersection.pos>
      <Spatial.GM_Point gi.id="i08">
        <Spatial.GM_Object.SRS>
          <Position.SpatialReferenceSystem gi.idref="i11"></Position.SpatialReferenceSystem>
        </Spatial.GM_Object.SRS>
```

```
        <Spatial.GM_Point.position>50400 10800</Spatial.GM_Point.position>
      </Spatial.GM_Point>
    </RoadMap.Intersection.pos>
</RoadMap.Intersection>

<Position.SpatialReferenceSystem gi.id="i11">UTM zone 36</Position.SpatialReferenceSystem>
```

E.3 A Property-Building-Loan example (english)

E.3.1 Application schema: PBL

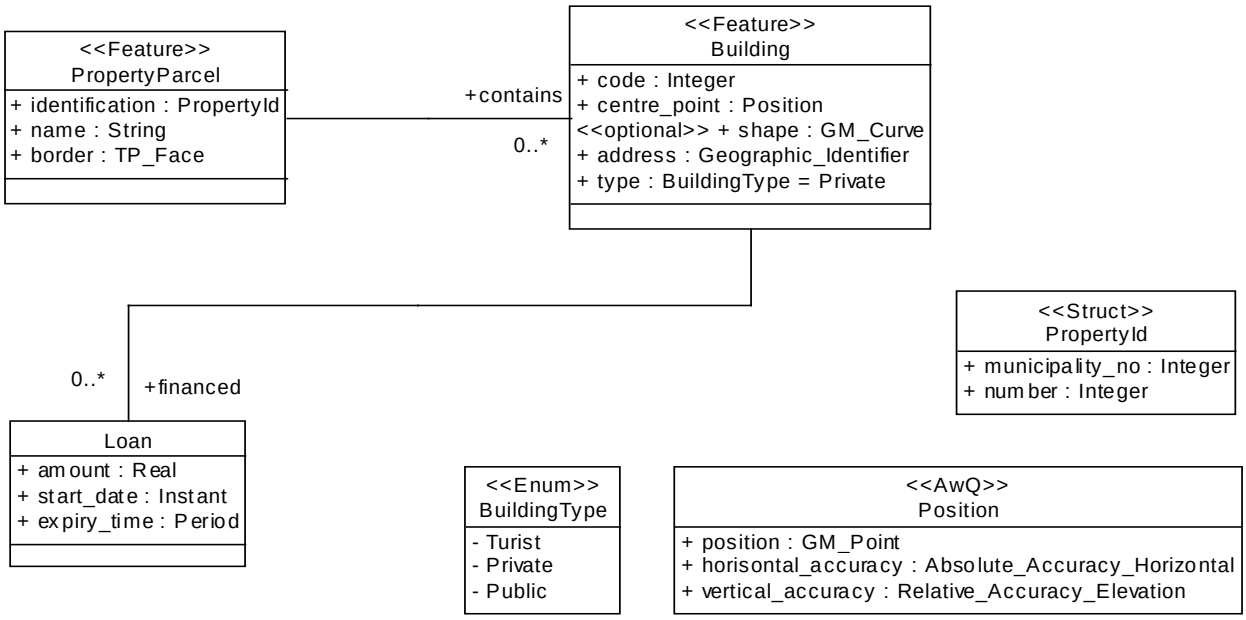


Figure E.3 — PBL Application schema

E.3.2 Example data

Map – Graphical view

Data tables for: Building, Property Parcel, Loan, GM_Point, GM_Surface and TP_Face

E.3.3 Document Type Declaration (pbl.dtd)

E.3.4 XML document

E.4 An example with national characters

E.5 Evaluation

Size considerations

Processing considerations

Compression