Annex G
(normative)
Generating longform EXPRESS schemas from shortform schemas

G.1 Introduction

This edition of ISO 10303-11 adds functionality for modularized modelling. An incomplete data specification may be defined in one SCHEMA and may be reused by another SCHEMA to be possibly specialized and extended. A high level SCHEMA may collect several such resources into one top level schema, which may be referred to as the shortform of an application schema, to build a complete data specification. The result of building the complete data specification is called the longform of an application schema.

NOTE The terms shortform and longform are defined in ISO 10303-1:2002.

As long as the implementation methods included in ISO 10303 are not updated to this edition of ISO 10303-11 or implementations are not updated to work with the new implementation methods, it may be useful to convert a data specification that is in accordance to edition 2 of ISO 10303-11 to a data specification that is in accordance to edition 1 of ISO 10303-11. This annex specifies rules for how such a conversion may be done.

The objective of these rules is to build a complete and consistent longform and minimize the loss of the semantics of the original data model.

NOTE Conformance requirements to this specification are stated in clause 4 and annex E.

G.2 Fundamental concepts

Input to the longform generation process is a shortform EXPRESS schema with USE and REFERENCE statements and one or more additional schemas. The process results in a new schema that contains all constructs defined in the shortform, adds all constructs explicitly interfaced with USE and REFERENCE, and then resolves all references for those other constructs according to 11 of this document.

The resulting longform is an almost semantically identical model without the USE and REFERENCE statements; referenced objects are brought directly into the schema. Information describing the originating schema and how that object was interfaced (implicitly or explicitly and by USE or REFERENCE) is discarded. Objects in the interfaced schemas that are not directly referenced through the objects in the shortform are discarded. Pruning and rewriting of some of the constructs of the input schemas takes place to omit the objects that are originally declared, but are not used in the final schema.

Specifically the longform generation has the following impact whether or not this involves the conversion from an edition 1 to an edition 2 schema:

— Loss of knowledge of the schema that each construct originates from.

— Loss of knowledge of how the constructs were interfaced, i.e., their visibility and instantiability are not known. Information describing how the object was interfaced (distinction between USE or REFERENCE), affects its instantiability and visibility; see clause 11 paragraphs 2 and 3. These implications are lost in the resulting longform schema.

— Selectable items of a SELECT type are pruned to remove those items that are not interfaced into the schema. According to 11.4.2, selectable items are not implicitly interfaced as a result of the interfacing of the SELECT type itself. If the selectable items were left in the select list, and the objects were not visible in the schema, the compilation of the longform schema would report syntactic errors.

— SUBTYPE and SUPERTYPE statements in entities are pruned to reflect the pruning of the subtype/supertype graph as described in 11.4.3 and annex C.

— Remarks may be discarded.

 ©ISO 2002 — All rights reserved

— The case of user identifiers may not be preserved.

— SCHEMA names in fully qualified attribute references are replaced with the SCHEMA name of the longform.

The conversion from an edition 2 shortform to an edition 1 longform requires the removal or change of EXPRESS constructs that are not supported by edition 1 of this document. In particular the following actions shall be performed:

— The schema version identifier is converted into a remark.

— EXTENSIBLE SELECT s are resolved into not extensible SELECT s.

— EXTENSIBLE ENUMERATION s are resolved into not extensible ENUMERATION s.

— SUBTYPE_CONSTRAINT s are rewritten if necessary to remove types that would otherwise not appear in the longform schema. They are also converted into SUPERTYPE statements in subtypes.

— TOTAL_OVER constraints are replaced by RULE s.

— ABSTRACT entities and GENERIC_ENTITY are converted into abstract supertype constraints.

— RENAMED constructs are converted to DERIVE attributes.

— Errors are produced for empty selects and for rules with no parameters.

The following sub-clauses describe these actions in more detail.

G.3 Conversion of edition 2 constructs to edition 1 constructs

G.3.1 Conversion of schema version identifier

The STRING that represents a schema version identifier (see 9.3 shall be enclosed within an embedded remark that is located right after the schema name separated by a blank and before the semicolon that completes the SCHEMA declaration. The enclosing quotes for the STRING shall be included in the remark. The STRING shall be preceded by the STRING `schema_version_id =` without the quotes that are used here to distinguish the string from the rest of this sentence.

EXAMPLE 1 The following edition 2 schema:

```
SCHEMA geometry_schema 'version_1';
END_SCHEMA;
```

would become the following edition 1 schema:

```
SCHEMA geometry_schema (* schema_version_id = 'version_1' *);
END_SCHEMA;
```

G.3.2 Conversion of extensible constructed defined data types

For the conversion of extensible select data types and extensible enumerations (see 8.4, a schema shall be chosen as the context schema for the conversion. The completion of the extensions shall be evaluated within the context schema. For each extensible constructed defined data type that is visible in the context schema and that is not defined in the context schema, it is necessary to define the completed enumeration or select list in the schema that the type is defined in. The result of this process will be that a context schema-specific schema is created for each schema that is visible in the context schema.

EXAMPLE 1 If the context schema was the shortform AIM of an Application Protocol and if it and the integrated resources used by this shortform contained extensible constructed defined types, the result

©ISO 2002 — All rights reserved

of this process would be a set of supporting schemas that were derived from the interfaced integrated resource schemas and that could only be used in the context of this AP.

For each extensible or extending data type in any schema that is visible to the context schema, the following conversion shall be performed:

— each extensible or extending data type shall be replaced with a non-extensible data type of the same name that is not an extension of any data type;

— the list items specified in each constructed data type shall be the set of items within the domain of the data type as evaluated with respect to the context schema;

— in the case that a defined data type in the schema that is visible in the context schema has an extensible or extending data type as its underlying data type, add a WHERE clause to that defined data type; this shall eliminate all items that result from the conversion and that are not valid list items in the set of schemas that are visible in the context schema;

— to maintain the dependencies among the extensible and extending data types their target constructs shall be related; depending, i.e., the extending data types shall be created as defined data types that use the extensible data type as underlying type; the elements of the underlying data type that are not valid in the defined data type shall be constrained using local rules; see below for examples.

The following sub-clauses give some details on how to implement these rules for ENUMERATION and SELECT data types.

G.3.2.1 Extensible enumeration

Extensible enumerations (see 8.4.1) shall be converted according to the procedure above to enumeration data types that are not extensible. The name of the source construct shall be maintained in the target.

An enumeration data type that is based on an extensible enumeration data type shall be converted to a defined data type that has as underlying data type the target enumeration data type of the extensible enumeration that it is based on. The name of the defined data type in the target model shall be the name of the corresponding enumeration data type in the source. The defined data type shall exclude invalid enumeration items, as they are specified for its underlying enumeration, from instantiation in the context schema by local rules.

EXAMPLE 1 Source schema according to EXPRESS edition 2:

```
SCHEMA S1;

TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
END_TYPE;

END_SCHEMA;
```

Target in longform edition 1 for `S1` as context schema:

```
TYPE general_approval = ENUMERATION OF (approved, rejected);
END_TYPE;
```

Source schema according to EXPRESS edition 2:

```
SCHEMA S2;

USE FROM S1 (general_approval);

TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH
(pending);
END_TYPE;
```

 ©ISO 2002 — All rights reserved

```
END_SCHEMA;
```

Target in longform edition 1 for S2 as context schema:

```
TYPE general_approval = ENUMERATION OF (approved, rejected, pending);
END_TYPE;

TYPE domain2_approval = general_approval;
END_TYPE;
```

Source schema according to EXPRESS edition 2:

```
SCHEMA S3;

USE FROM S1 (general_approval);

TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH
(cancelled);
END_TYPE;

END_SCHEMA;
```

Target in longform edition 1 for S3 as context schema:

```
TYPE general_approval = ENUMERATION OF (approved, rejected, cancelled);
END_TYPE;

TYPE domain3_approval = general_approval;
END_TYPE;
```

Source schema according to EXPRESS edition 2:

```
SCHEMA S4;

USE FROM S2 (domain2_approval);

REFERENCE FROM S3 (domain3_approval);

TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
END_TYPE;

END_SCHEMA;
```

Target in longform edition 1 for S4 as context schema:

```
TYPE general_approval = ENUMERATION OF (approved, rejected, pending,
cancelled, rework);
END_TYPE;

TYPE domain2_approval = general_approval;
WHERE
  WR1: SELF <> cancelled;
END_TYPE;

CONTINUE

TYPE domain3_approval = general_approval;
WHERE
```

©ISO 2002 — All rights reserved

```
  WR1: SELF <> pending;
  WR2: SELF <> rework;
END_TYPE;

TYPE specific_approval = general_approval;
WHERE
  WR1: SELF <> cancelled;
END_TYPE;
```

EXAMPLE 2 Source schema according to EXPRESS edition 2:

```
SCHEMA EXPORT;

TYPE colour = EXTENSIBLE ENUMERATION;
END_TYPE;

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

END_SCHEMA;

SCHEMA IMPORT;

USE FROM EXPORT (stop_light);

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

END_SCHEMA;
```

Target enumerations in longform edition 1 for `IMPORT` as context schema:

```
TYPE colour = ENUMERATION OF (red, white, yellow, green);
END_TYPE;

TYPE canadian_flag = colour;
END_TYPE;

TYPE stop_light = colour;
WHERE
  WR1: SELF <> white;
END_TYPE;
```

Special attention needs to be given to the fact that enumeration data types according to edition 1 of this document imply an order of the enumeration items. This concept of order is not present in edition 2.

G.3.2.2 Extensible select

Extensible selects (see 8.4.2) shall be converted according to the procedure above to select data types that are not extensible. The name of the source construct shall be maintained in the target.

A select data type that is based on an extensible select data type shall be converted to a defined data type that has as underlying data type the target select data type of the extensible select that it is based on. The name of the defined data type in the target model shall be the name of the corresponding select data type in the source. The defined data type shall exclude invalid select items, as they are specified for its underlying select, from instantiation in the context schema by local rules.

EXAMPLE 1 Source schema according to EXPRESS edition 2:

 ©ISO 2002 — All rights reserved

```
SCHEMA EXPORT;

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

ENTITY nail;
END_ENTITY;

ENTITY screw;
END_ENTITY;

END_SCHEMA;

SCHEMA IMPORT;

USE FROM EXPORT (attachment_method);

TYPE permanent_attachment = SELECT BASED ON attachment_method WITH (glue, weld);
END_TYPE;

TYPE simple_attachment = SELECT BASED ON attachment_method WITH (needle, tape);
END_TYPE;

...;

END_SCHEMA;
```

Target selects in longform edition 1 for IMPORT as context schema:

```
TYPE attachment_method = SELECT (nail, screw, glue, weld, needle, tape);
END_TYPE;

TYPE permanent_attachment = attachment_method;
WHERE
  WR1: NOT (('IMPORT.NEEDLE') IN TYPEOF(SELF));
  WR2: NOT (('IMPORT.TAPE') IN TYPEOF(SELF));
END_TYPE;

TYPE simple_attachment = attachment_method;
WHERE
  WR1: NOT (('IMPORT.GLUE') IN TYPEOF(SELF));
  WR2: NOT (('IMPORT.WELD') IN TYPEOF(SELF));
END_TYPE;
```

EXAMPLE 2 Source schema according to EXPRESS edition 2:

```
SCHEMA SELECT_TREE_EXPORT;

TYPE t1 = EXTENSIBLE SELECT (tata);
END_TYPE;

TYPE t2 = EXTENSIBLE SELECT BASED_ON t1 WITH (titi);
END_TYPE;

TYPE t3 = EXTENSIBLE SELECT BASED_ON t2;
END_TYPE;

ENTITY tata;
```

©ISO 2002 — All rights reserved

```
END_ENTITY;

ENTITY titi;
END_ENTITY;

END_SCHEMA;

SCHEMA SELECT_TREE_IMPORT;

USE FROM EXPORT (t1, tata, t2, titi, t3);

TYPE t4 = SELECT BASED_ON t1 WITH (toto);
END_TYPE;

ENTITY toto;
END_ENTITY;

END_SCHEMA;
```

Target selects in longform edition 1 for SELECT_TREE_IMPORT as context schema:

```
TYPE t1 = SELECT (tata, titi, toto);
END_TYPE;

TYPE t2 = t1;
WHERE
  WR1: NOT (('SELECT_TREE_IMPORT.TOTO') IN TYPEOF(SELF));
END_TYPE;

TYPE t3 = t2;
END_TYPE;

TYPE t4 = t1;
WHERE
  WR1: NOT (('SELECT_TREE_IMPORT.TITI') IN TYPEOF(SELF));
END_TYPE;
```

********** to be continued **********

G.3.3 Conversion of subtype constraints

The constraints on the subtype/supertype graph in the context of the context schema are added to all the schemas visible to the context schema. The subtype constraint is deleted from all schemas visible to the context schema. For each total over constraint a global rule is added to the schema in which the subtype constraint is specified as follows:

the rule name shall be total_over_<subtype_constraint_name and the rule shall be over the list of entity types in the total over and the supertype entity;

the where rule in the global rule shall be of the form <supertype = <subtype 1 + <subtype 2 + ... + <subtype n where <supertype is the name of the supertype and <subtype x is the name of the xth subtype in the total over.

JH - For any entity constrained by a subtype constraint containing a supertype constraint if the supertype constraint only contains entity types defined in the same schema as the supertype, the supertype constraint is enclosed in parentheses and added to a supertype constraint specified in the entity;

each constraint resulting from a subtype constraint is combined via ANDOR with any other constraints

 ©ISO 2002 — All rights reserved

added from different subtype constraint;

if any supertype constraint contains entity types not defined in the same schema as the supertype, a global rule is added to the context schema and each such supertype constraints is converted to an expression in a where clause of the global rule.

G.3.4 Conversion of abstract entity and generalized types used in abstract entities

Map the abstract entity declaration into an abstract supertype constraint on the entity in the class 1 schema.

For attributes in the abstract entity type that have a generalized type as their domain the following mapping applies:

if all subtypes redeclare the attribute domain to be of the same type, migrate that type to be the type of the attribute in the supertype in the class 1 schema;

in all other cases, create a select type in the class 1 schema adding all named types that are the type of the attribute in any redeclarations in any subtype of the supertype;

(JH: last bullet deleted)

A note will be added stating that this causes problems in Part 21 but there is nothing we can do about that.

G.3.5 Conversion of attributes renamed in a redeclaration

For each attribute that is renamed in a redeclaration in a subtype the following mapping applies:

a derived attribute with the new name is created in the subtype where the derivation of the attribute value is `SELF\<supertype name.<old attribute name`

A note will be added stating that in the case that a the new attr name has anything assigned to it in a function, this causes failures in those functions. Manual fixup of schema would be required in this rare case.

©ISO 2002 — All rights reserved