INTERNATIONAL STANDARD

ISO 10303-52

First edition
2011-03-01

Industrial automation systems and integration — Product data representation and exchange —

Part 52:
Integrated generic resource: Mesh-based topology

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 52: Ressources génériques intégrées: Topologie fondée sur la maille

Reference number
ISO 10303-52:2011(E)

Contents

Page

Foreword . . . vii

Introduction . . . viii

1 Scope . . . 1

2 Normative references . . . 1

3 Terms, definitions and abbreviated terms . . . 2
3.1 Terms defined in ISO 10303-1 . . . 2
3.2 Terms defined in ISO 10303-110 . . . 2
3.3 Other terms and definitions . . . 3
3.4 Abbreviated terms . . . 3

4 Mesh topology . . . 4
4.1 Fundamental concepts and assumptions . . . 4
4.1.1 Structured mesh . . . 4
4.1.2 Unstructured mesh . . . 6
4.2 mesh_topology_schema type definitions . . . 6
4.2.1 cell_shape . . . 6
4.2.2 cell_shape_0D . . . 6
4.2.3 cell_shape_1D . . . 7
4.2.4 cell_shape_2D . . . 7
4.2.5 cell_shape_3D . . . 8
4.2.6 indices_group . . . 8
4.2.7 mesh_location . . . 9
4.2.8 mesh_maths_space_type . . . 10
4.2.9 structured_mesh_type . . . 10
4.3 mesh_topology_schema entity definitions . . . 13
4.3.1 array_based_unstructured_mesh . . . 13
4.3.2 array_based_unstructured_mesh_and_vertices . . . 14
4.3.3 cell . . . 15
4.3.4 cell_with_explicit_boundary . . . 15
4.3.5 cell_of_structured_mesh . . . 16
4.3.6 explicit_unstructured_mesh . . . 16
4.3.7 extraction_of_structured_submesh . . . 17
4.3.8 extraction_of_submesh . . . 18
4.3.9 extraction_of_submesh_by_cells . . . 18
4.3.10 extraction_of_submesh_by_vertices . . . 19
4.3.11 indices_list . . . 19
4.3.12 indices_range . . . 20
4.3.13 mesh . . . 20
4.3.14 mesh_derived_maths_space . . . 21
4.3.15 product_of_mesh . . . 22

4.3.16 rind . . . 23
4.3.17 structured_mesh . . . 23
4.3.18 structured_mesh_with_rind . . . 25
4.3.19 submesh . . . 26
4.3.20 unstructured_mesh . . . 26
4.3.21 vertex_defined_cell . . . 34
4.4 mesh_topology_schema function definitions . . . 35
4.4.1 all_mesh_vertices . . . 35
4.4.2 cell_counts . . . 36
4.4.3 shorten_array . . . 38
4.4.4 this_schema . . . 39

5 Mesh connectivity . . . 39
5.1 General . . . 40
5.2 Fundamental concepts and assumptions . . . 40
5.3 mesh_connectivity_schema type definitions . . . 42
5.3.1 mismatched_region_type . . . 42
5.4 mesh_connectivity_schema entity definitions . . . 42
5.4.1 matched_mesh_connection . . . 42
5.4.2 mesh_connectivity . . . 44
5.4.3 mesh_overset_hole . . . 45
5.4.4 mismatched_donor_mesh . . . 45
5.4.5 mismatched_mesh_connection . . . 46
5.4.6 mismatched_mesh_region . . . 47
5.4.7 multiple_mesh_block . . . 47
5.4.8 structured_donor_mesh . . . 48
5.4.9 unstructured_donor_mesh . . . 49

6 Mesh function . . . 50
6.1 General . . . 50
6.2 Fundamental concepts and assumptions . . . 50
6.3 mesh_function_schema entity definitions . . . 50
6.3.1 mesh_function . . . 50
6.3.2 mesh_function_basis . . . 53
6.4 mesh_function_schema subtype constraint definitions . . . 55
6.4.1 sc1_application_defined_function . . . 55
6.4.2 sc1_unary_generic_expression . . . 55

Annex A (normative) Short names of entities . . . 56

Annex B (normative) Information object registration . . . 58
B.1 Document identification . . . 58
B.2 Schema identification . . . 58

Annex C (informative) Computer-interpretable listings . . . 59

Annex D (informative) EXPRESS-G diagrams . . . 60

Annex E (informative) Additional information 73

Bibliography 74

Index 75

Figures

Figure 1 Schema relationships ix
Figure 2 Example convention for a 2-D cell centre 5
Figure 3 Example mesh with rind vertices 5
Figure 4 A 1-D rectangular_mesh or pentahedral_mesh or pyramidal_mesh or tetrahedral_-mesh (with $i = 5$) 11
Figure 5 A 2-D rectangular_mesh (with $i = 5$, $j = 4$) 11
Figure 6 A 3-D rectangular_mesh (with $i = 5$, $j = 4$, $k = 3$) 11
Figure 7 A 2-D pentahedral_mesh or pyramidal_mesh or tetrahedral_mesh (with $i = 5$, $j = 4$) 12
Figure 8 A 3-D pentahedral_mesh (with $i = 5$, $j = 4$, $k = 3$) 12
Figure 9 A 3-D pyramidal_mesh (with $i = 5$, $j = 4$, $k = 3$) 13
Figure 10 A 3-D tetrahedral_mesh (with $i = 5$, $j = 4$, $k = 3$) 13
Figure 11 Parametric coordinate system for a 1-D structured mesh 24
Figure 12 Parametric coordinate system for a 2-D structured mesh 24
Figure 13 Parametric coordinate system for a 3-D structured mesh 24
Figure 14 Linear, quadratic and cubic line cells 27
Figure 15 Linear, quadratic and cubic triangle cells 28
Figure 16 Linear, quadratic and cubic quadrilateral cells 29
Figure 17 Linear, quadratic and cubic hexahedron cells 30
Figure 18 Linear, quadratic and cubic wedge cells 31
Figure 19 Linear, quadratic and cubic tetrahedron cells 32
Figure 20 Linear, quadratic and cubic pyramid cells 33
Figure 21 A 1-to-1 abutting interface 40
Figure 22 A mismatched abutting interface 41
Figure 23 An overset interface 41
Figure D.1 Entity level diagram of mesh_topology_schema schema (page 1 of 10) 60
Figure D.2 Entity level diagram of mesh_topology_schema schema (page 2 of 10) 61
Figure D.3 Entity level diagram of mesh_topology_schema schema (page 3 of 10) 62
Figure D.4 Entity level diagram of mesh_topology_schema schema (page 4 of 10) 63
Figure D.5 Entity level diagram of mesh_topology_schema schema (page 5 of 10) 64
Figure D.6 Entity level diagram of mesh_topology_schema schema (page 6 of 10) 65
Figure D.7 Entity level diagram of mesh_topology_schema schema (page 7 of 10) 65
Figure D.8 Entity level diagram of mesh_topology_schema schema (page 8 of 10) 66
Figure D.9 Entity level diagram of mesh_topology_schema schema (page 9 of 10) 67
Figure D.10 Entity level diagram of mesh_topology_schema schema (page 10 of 10) 68
Figure D.11 Entity level diagram of mesh_connectivity_schema schema (page 1 of 3) 69
Figure D.12 Entity level diagram of mesh_connectivity_schema schema (page 2 of 3) 70
Figure D.13 Entity level diagram of mesh_connectivity_schema schema (page 3 of 3) 71
Figure D.14 Entity level diagram of mesh_function_schema schema (page 1 of 1) 72

Tables

Table 1	Number of vertices in a structured_mesh	25
Table 2	Edges of triangle, quadrilateral and polygon cells	27
Table 3	Edges of hexahedron, wedge, tetrahedron and pyramid cells	27
Table 4	Faces of hexahedron, wedge, tetrahedron and pyramid cells	32
Table 5	Domain of the control values table for a **mesh_function**	52
Table A.1	Short names of entities	57
Table E.1	Elements of **mesh_topology_schema** used by other schemas	73

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 2.

The main task of technical committees is to prepare International Standards. 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.

Attention is drawn to the possibility that some of the elements of this part of ISO 10303 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 10303-52 was prepared by Technical Committee ISO TC184/SC4. *Automation systems and integration,* Subcommittee SC4 *Industrial data.*

ISO 10303 is organised as a series of parts, each published separately. The structure of ISO 10303 is described in ISO 10303-1.

Each part of ISO 10303 is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the integrated generic resource series.

The integrated generic resources and the integrated application resources specify a single conceptual product data model.

A complete list of parts of ISO 10303 is available from Internet:

`<http://www.tc184-sc4.org/titles/STEP_titles.rtf>`

Should further parts of ISO 10303 be published, they will follow the same numbering pattern.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing products throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and as a basis for archiving.

This part of ISO 10303 is a member of the integrated resources series. Major subdivisions of this part of ISO 10303 are:

— **mesh_topology_schema**;

— **mesh_connectivity_schema**.

— **mesh_function_schema**.

The relationships of the schemas in this part of ISO 10303 to other schemas that define the integrated resources of this International Standard are illustrated in Figure 1 using the EXPRESS-G notation. EXPRESS-G is defined in ISO 10303-11. The schemas identified in the bold boxes are specified in this part of ISO 10303. The **support_resource_schema** is specified in ISO 10303-41. The **topology_-schema** is specified in ISO 10303-42. The **mathematical_constructs_schema** and the **mathematical_-functions_schema** are specified in ISO 10303-50. The **mathematical_description_of_distribution_-schema** is specified in ISO 10303-51. The **structural_response_representation_schema** is specified in ISO 10303-104. The **ISO13584_generic_expressions_schema** is specified in ISO 13584-20. Except for **ISO13584_generic_expressions_schema**, the schemas illustrated in Figure 1 are components of the integrated resources.

There are many applications that have to deal with massive amounts of data, which is normally numerical in nature. The quantity of data may be measured in gigabytes and in some cases terabytes. Examples include computational fluid dynamics, dynamic simulation of vehicle behaviour, and experimental data of many kinds ranging from high energy physics to global weather measurements.

A major concern in dealing with such data is to optimise the data representation and structure with respect to data transmission and storage. As part of the optimisation, the data tends to be maintained in large arrays where any particular data element can be referenced by a simple index into the array. When the data is part of a computer simulation the data is usually associated with a mesh of some kind — either structured or unstructured. The data can be bound to the vertices of the mesh or to the cells of the mesh. In any case, it is also possible to represent the simpler kinds of meshes by an indexing scheme. Within this part illustrative examples have been principally taken from the field of computational fluid dynamics.

This part of ISO 10303 provides general, application independent, means of representing indexible data and meshes.

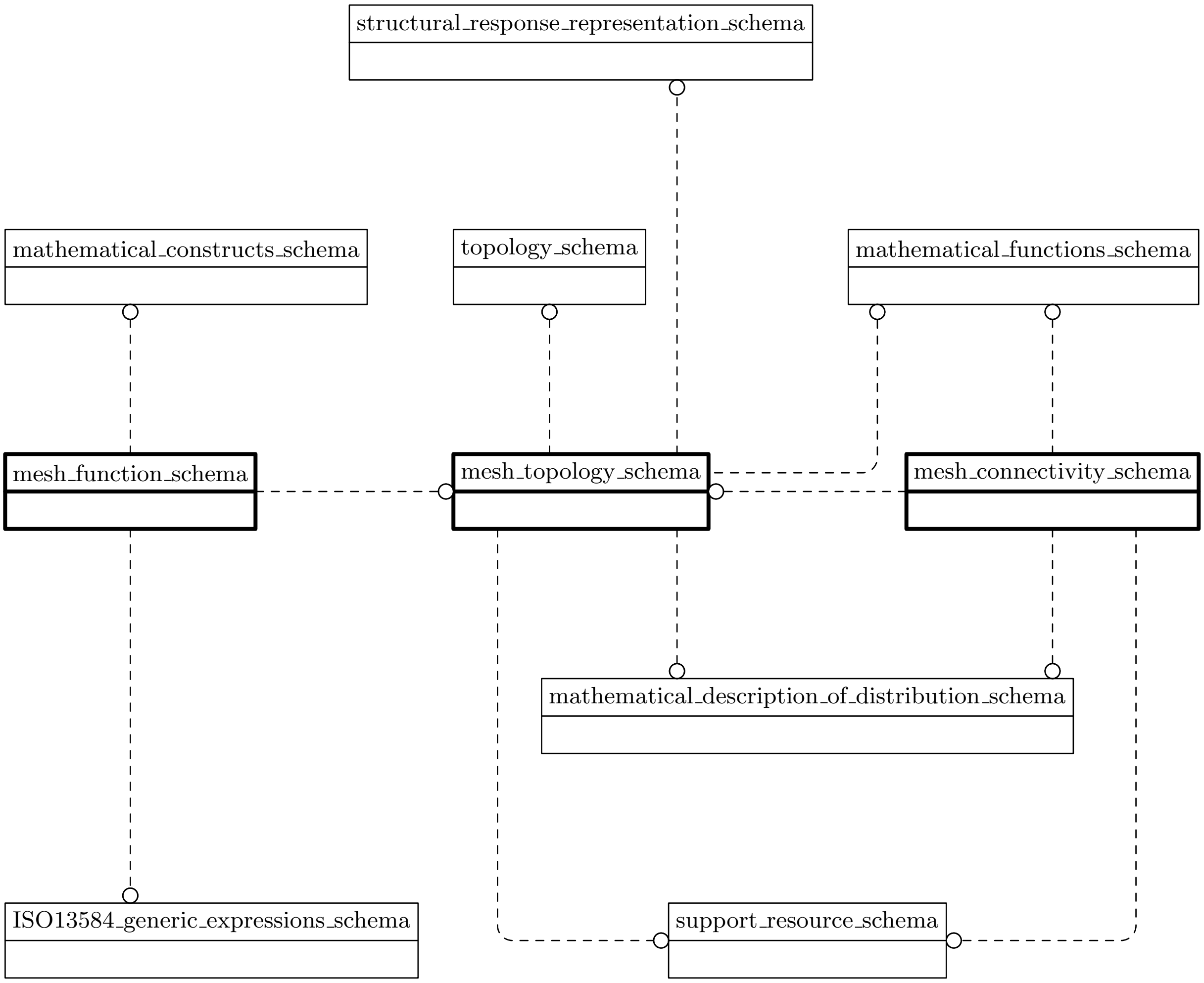


Figure 1 – Schema relationships

Industrial automation systems and integration — Product data representation and exchange —

Part 52:
Integrated generic resource: Mesh-based topology

1 Scope

This part of ISO 10303 provides general and application-independent means of representing structured and unstructured meshes, and mathematical functions and numeric data defined over such meshes. The schemas in this document are specified in the EXPRESS language; EXPRESS is defined in ISO 10303-11.

The following are within the scope of this part of ISO 10303:

— mesh-based topologies;

— cell connectivity and multiblock mesh interfaces;

— mathematical functions defined over meshes;

— the association of numeric data with the cells, faces, edges, and vertices of a mesh.

The following are outside the scope of this part of ISO 10303:

— applications of mesh topologies;

— applications of mesh interfaces;

— the semantics of data associated with a mesh.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For updated references, the latest edition of the referenced document (including any amendments) applies.

ISO 10303-1, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.*

ISO 10303-11, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description method: The EXPRESS language reference manual.*

ISO 10303-41, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support.*

ISO 10303-42, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation.*

ISO 10303-50, *Industrial automation systems and integration — Product data representation and exchange — Part 50: Integrated generic resource: Mathematical constructs.*

ISO 10303-51, *Industrial automation systems and integration — Product data representation and exchange — Part 51: Integrated generic resource: Mathematical description.*

ISO 10303-104, *Industrial automation systems and integration — Product data representation and exchange — Part 104: Integrated application resource: Finite element analysis.*

ISO 10303-110, *Industrial automation systems and integration — Product data representation and exchange — Part 110: Integrated application resource: Mesh-based computational fluid dynamics.*

ISO 13584-20, *Industrial automation systems and integration — Parts library — Part 20: Logical resource: Logical model of expressions.*

3 Terms, definitions and abbreviated terms

3.1 Terms defined in ISO 10303-1

For the purposes of this document, the following terms defined in ISO 10303-1 apply.

— application protocol (AP)

— integrated resource

— product

3.2 Terms defined in ISO 10303-110

For the purposes of this document, the following term defined in ISO 10303-110 applies.

— rind

3.3 Other terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.3.1
cell
manifold of dimensionality one or higher that is a part of, or the whole of, a mesh

3.3.2
cell edge
one-dimensional manifold that is on the boundary of a cell and that joins two cell vertices

3.3.3
cell face
two-dimensional manifold that is on the boundary of a cell and that is enclosed by one or more cell edges

3.3.4
cell vertex
vertex that is at the end of one or more cell edges

3.3.5
mesh
arrangement of cells with connectivity between the cells defined by the possession of common cell faces or cell edges

3.3.6
topological region
point set with a single topological dimension

3.3.7
vertex
point within, or on the boundary of, a cell

NOTE 1 A vertex can, but need not, be a cell vertex.

NOTE 2 In some applications, particularly in finite element analysis, the word node is used as an equivalent term to vertex.

3.4 Abbreviated terms

CFD	computational fluid dynamics
URL	Universal Resource Locator

4 Mesh topology

The following EXPRESS declaration begins the **mesh_topology_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA mesh_topology_schema;
  REFERENCE FROM mathematical_description_of_distribution_schema  -- ISO 10303-51
    (property_distribution_description);
  REFERENCE FROM mathematical_functions_schema                    -- ISO 10303-50
    (maths_space);
  REFERENCE FROM structural_response_representation_schema        -- ISO 10303-104
    (element_order,
     element_representation,
     fea_model);
  REFERENCE FROM support_resource_schema                          -- ISO 10303-41
    (identifier,
     label,
     text);
  REFERENCE FROM topology_schema                                  -- ISO 10303-42
    (topological_representation_item,
     vertex, vertex_point);
(*
```

NOTE The schemas referenced above can be found in the following parts of ISO 10303:

mathematical_description_of_distribution_schema	ISO 10303-51
mathematical_functions_schema	ISO 10303-50
structural_response_representation_schema	ISO 10303-104
support_resource_schema	ISO 10303-41
topology_schema	ISO 10303-42

4.1 Fundamental concepts and assumptions

A mesh is defined by its vertices and the connections between the vertices. A mesh is a connected graph.

4.1.1 Structured mesh

In a structured mesh the cells are arranged in a regular pattern and their shapes are implied by the particular kind of mesh.

A 3–D rectangular mesh is a mesh of hexahedral cells. Each cell is a dimensionality 3 hexahedral region defined by eight vertices forming the corners of the hexahedron. Each cell is bounded by six faces, where each face is the quadrilateral defined by four vertices. A face is limited by the four edges that connect the four vertices.

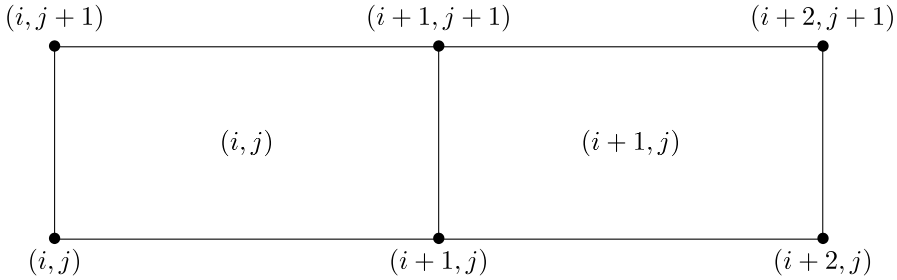


Figure 2 – Example convention for a 2-D cell centre

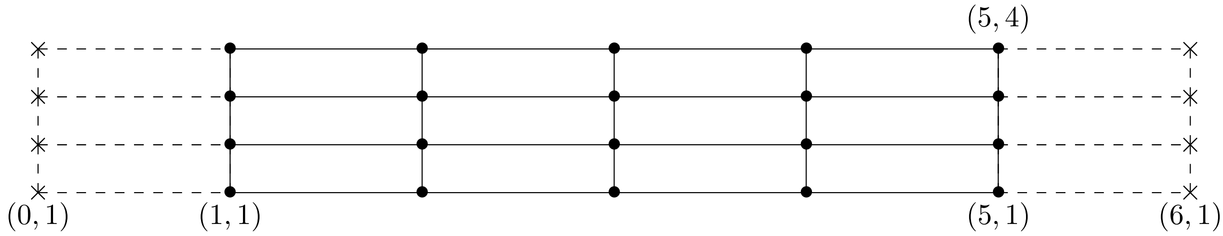


Figure 3 – Example mesh with rind vertices

A 2–D rectangular mesh is a mesh of quadrilaterals. Each cell is a dimensionality two quadrilateral region defined by four vertices forming the corners of the quadrilateral. Each cell is limited by the four edges that connect the four vertices.

A 1–D mesh is of linear form. Each cell is a dimensionality one linear region bounded by two vertices.

Indices describing a structured mesh are ordered: for 3–D (i, j, k); (i, j) is used for 2–D; and (i) for 1–D.

Cell centres, face centres, and edge centres are indexed by the minimum of the connecting vertices.

EXAMPLE 1 For example a 2-D cell center (or face centre on a 3-D mesh) would have the conventions shown in Figure 2.

In addition, the default beginning vertex for a regular mesh is $(1, 1, 1)$; this means the default beginning cell centre of a regular mesh is also $(1, 1, 1)$.

There may be locations outside the mesh itself. These are referred to as ‘rind’ or ghost points and may be associated with fictitious vertices or cell centres. They are distinguished from the vertices and cells making up the mesh (including its boundary vertices), which are referred to as ‘core’ points.

EXAMPLE 2 Figure 3 shows a 2–D mesh with a single row of ‘rind’ vertices at the minimum and maximum i-faces. The mesh size (i.e., the number of ‘core’ vertices in each direction) is 5×4. ‘Core’ vertices are designated by ‘•’, and ‘rind’ vertices by ‘$\times$’. Default indexing is also shown for the vertices.

For a mesh, the minimum faces in each coordinate direction are denoted i-min, j-min and k-min; the maximum faces are denoted i-max, j-max and k-max. These are the minimum and maximum 'core' faces.

EXAMPLE 3 $i-$min is the face or mesh plane whose core vertices have minimum i index (which if using default indexing is 1).

4.1.2 Unstructured mesh

An unstructured mesh is composed of cells, where the cells need not form a regular pattern and the shape of the cells is not restricted to be uniform throughout the mesh. Cells have vertices at their corners and may also have nodes on cell edges, cell faces, and in the interior of the cell.

Each cell in an irregular mesh has at least one vertex in common with at least one other cell in the mesh. The connectivity and adjacency of the cells may be determined from the common vertices.

Each cell in an unstructured mesh is explicitly represented in terms of its shape and an ordered list of its vertices. The vertices are implied rather than being explicitly represented. Essentially all the vertices in a mesh can be mapped to a sequential list, and reference to a vertex is then equivalent to specifying the particular position in the list.

4.2 mesh_topology_schema type definitions

4.2.1 cell_shape

A **cell_shape** is an identifier of an unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape = EXTENSIBLE SELECT
    (cell_shape_0D,
     cell_shape_1D,
     cell_shape_2D,
     cell_shape_3D);
END_TYPE;
(*
```

4.2.2 cell_shape_0D

A **cell_shape_0D** is an identifier of a topologically 0–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_0D = EXTENSIBLE ENUMERATION OF
    (single);
END_TYPE;
(*
```

Enumerated item definitions:

single: singleton vertex.

4.2.3 cell_shape_1D

A **cell_shape_1D** is an identifier of a topologically 1–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_1D = EXTENSIBLE ENUMERATION OF
    (line);
END_TYPE;
(*
```

Enumerated item definitions:

line: a topological line requiring 2 vertices.

4.2.4 cell_shape_2D

A **cell_shape_2D** is an identifier of a topologically 2–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_2D = EXTENSIBLE ENUMERATION OF
    (quadrilateral,
     triangle);
END_TYPE;
(*
```

Enumerated item definitions:

quadrilateral: four sided cell requiring 4 vertices;

triangle: three sided cell requiring 3 vertices;

NOTE 1 This type is defined as EXTENSIBLE to enable other 2D cell shapes to be added to the list as required by particular applications.

4.2.5 cell_shape_3D

A **cell_shape_3D** is an identifier of a topologically 3–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_3D = EXTENSIBLE ENUMERATION OF
    (hexahedron,
     wedge,
     tetrahedron,
     pyramid);
END_TYPE;
(*
```

Enumerated item definitions:

hexahedron: hexahedral (six quadrilateral faces) requiring 8 vertices;

wedge: pentahedral (three quadrilateral faces and two triangular faces) requiring 6 vertices;

tetrahedron: tetrahedral form (four triangular faces) requiring 4 vertices;

pyramid: pyramidal form (one quadrilateral face and four triangular faces) requiring 5 vertices.

NOTE 1 This type is defined as EXTENSIBLE to enable other 3D cell shapes to be added to the list as required by particular applications.

4.2.6 indices_group

An **indices_group** is a selection of a group of indices into a multi-dimensional array.

EXPRESS specification:

```
*)
TYPE indices_group = SELECT
```



```
      (indices_list,
       indices_range);
END_TYPE;
(*
```

4.2.7 mesh_location

A **mesh_location** is an enumeration of locations with respect to a mesh.

EXPRESS specification:

```
*)
TYPE mesh_location = EXTENSIBLE ENUMERATION OF
      (unspecified,
       application_defined,
       vertices,
       cell_centre,
       face_centre,
       iface_centre,
       jface_centre,
       kface_centre,
       edge_centre);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: not specified;

application_defined: specified via an external agreement between the data creator and the data user;

vertices: at cell vertices for cells within the mesh;

cell_centre: the centre of a cell; this is also appropriate for entities associated with cells but not necessarily with a given location in a cell;

face_centre: the centre of a generic face which can point in any coordinate direction;

iface_centre: the centre of a face in 3-D whose computational normal points in the i direction;

jface_centre: the centre of a face in 3-D whose computational normal points in the j direction;

kface_centre: the centre of a face in 3-D whose computational normal points in the k direction;

edge_centre: the centre of an edge.

4.2.8 mesh_maths_space_type

A **mesh_maths_space_type** is an enumeration of the kinds of associations of a **mesh_derived_maths_-space** and a **mesh**.

EXPRESS specification:

```
*)
TYPE mesh_maths_space_type = EXTENSIBLE ENUMERATION OF
     (cells,
      vertices);
END_TYPE;
(*
```

Enumerated item definitions:

cells: data is associated with mesh cells;

vertices: data is associated with mesh vertices.

4.2.9 structured_mesh_type

A **structured_mesh_type** is an enumeration of the kinds of structured meshes.

EXPRESS specification:

```
*)
TYPE structured_mesh_type = EXTENSIBLE ENUMERATION OF
    (rectangular,
     pentahedral,
     pyramidal,
     tetrahedral);
END_TYPE;
(*
```

Enumerated item definitions:

rectangular: a structured mesh that is topologically linear in 1–D, quadrilateral in 2–D, hexahedral in 3–D, etc.

In 2–D the cells are all quadrilateral. In 3–D the cells are all hexahedral.

NOTE 1 Illustrations of rectangular mesh topologies are shown in Figure 4, Figure 5 and Figure 6.

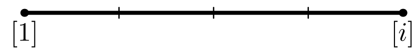


Figure 4 – A 1-D rectangular_mesh or pentahedral_mesh or pyramidal_mesh or tetrahedral_mesh (with $i = 5$)

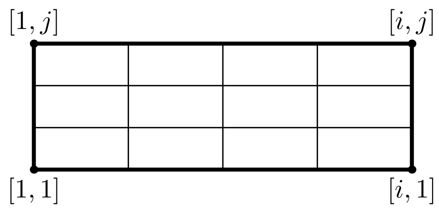


Figure 5 – A 2-D rectangular_mesh (with $i = 5$, $j = 4$)

pentahedral: a structured mesh that is topologically linear in 1–D, triangular in 2–D, and pentahedral, with 2 triangular and 3 quadrilateral faces forming a wedge-like shape, in 3–D. (where one of the edges between a pair of rectangular faces is analogous to the axis of a sector of a cylinder).

It is convenient to think of this kind of mesh as like a sector of a circle in 2–D where one apex point is anolagous to the centre point of the circle, and like a sector of a cylinder in 3–D where one of the edges between a pair of rectangular faces is analogous to the axis of the cylinder.

In 2–D the cells adjacent to the apex are triangular; the rest are quadrilateral.

In 3–D the cells adjacent to the axis edge are pentahedral; the rest are hexahedral.

NOTE 2 Illustrations of pentahedral mesh topologies are shown in Figure 4, Figure 7 and Figure 8.

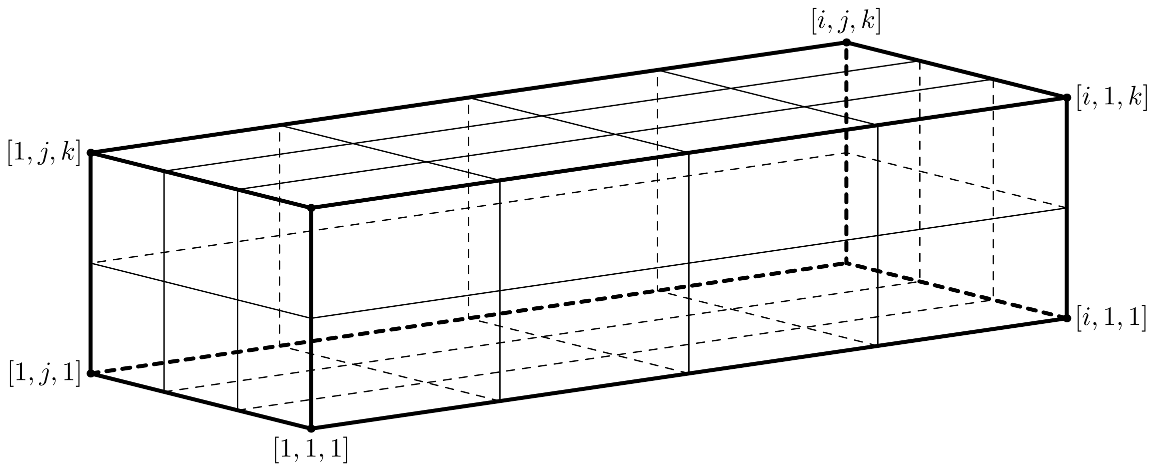


Figure 6 – A 3-D rectangular_mesh (with $i = 5$, $j = 4$, $k = 3$)

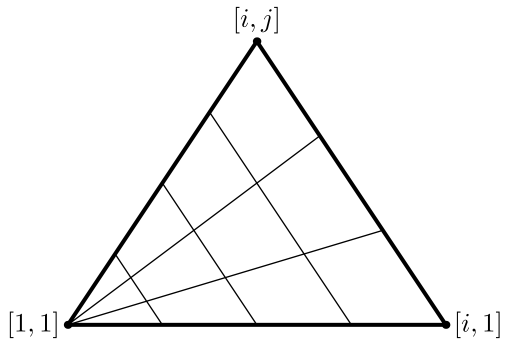


Figure 7 – A 2-D pentahedral_mesh or pyramidal_mesh or tetrahedral_mesh (with $i = 5$, $j = 4$)

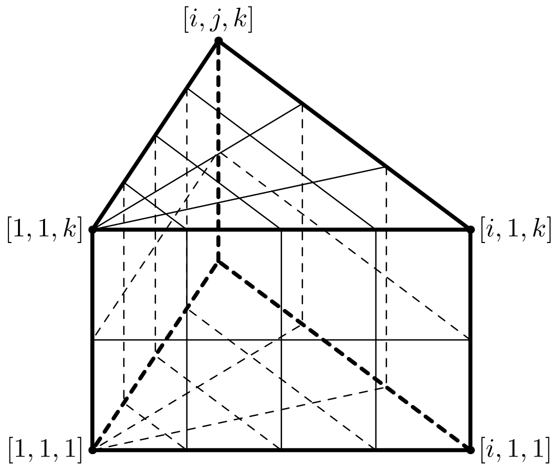


Figure 8 – A 3-D pentahedral_mesh (with $i = 5$, $j = 4$, $k = 3$)

pryamidal: a structured mesh that is topologically linear in 1–D, triangular in 2–D, and pyramidal in 3–D.

It is convenient to think of this kind of mesh as like a sector of a circle in 2–D where one apex point is analogous to the centre point of the circle, and like a sector of a sphere in 3–D where the apex point is analogous to the centre point of a sector of the sphere.

In 2–D the cells adjacent to the apex are triangular; the rest are quadrilateral.

In 3–D the cells adjacent to the apex are pyrimidal; the rest are hexahedral.

NOTE 3 Illustrations of pyramidal mesh topologies are shown in Figure 4, Figure 7 and Figure 9.

tetrahedral: a structured mesh that is topologically linear in 1–D, triangular in 2–D, and tetrahedral in 3–D.

It is convenient to think of this kind of mesh as like a sector of a circle in 2–D where one apex point is analogous to the centre point of the circle, and like

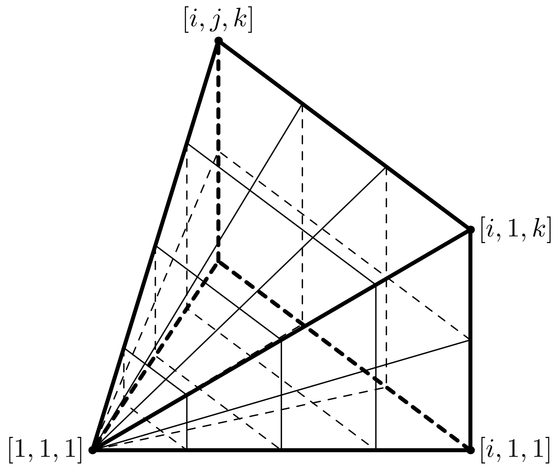


Figure 9 – A 3-D pyramidal_mesh (with $i = 5$, $j = 4$, $k = 3$)

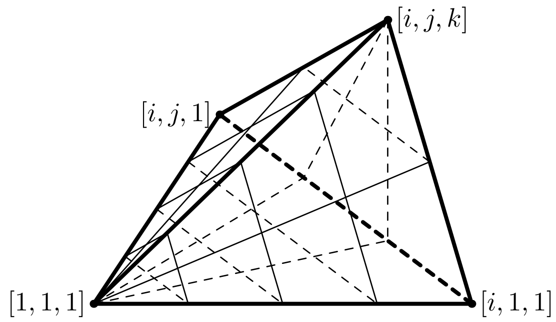


Figure 10 – A 3-D tetrahedral_mesh (with $i = 5$, $j = 4$, $k = 3$)

In 2–D the cells adjacent to the apex are triangular; the remainder are quadrilateral.

In 3–D the cells adjacent to the apex are tetrahedral and the cells adjacent to one edge from the apex are pentahedral; the remainder are hexahedral.

NOTE 4 Illustrations of tetrahedral mesh topologies are shown in Figure 4, Figure 7 and Figure 10.

4.3 mesh_topology_schema entity definitions

4.3.1 array_based_unstructured_mesh

An **array_based_unstructured_mesh** is a representation of an **unstructured_mesh** designed to minimise the amount of data by not requiring explicit identification of the vertices of the cells in the mesh.

EXPRESS specification:

```
*)
ENTITY array_based_unstructured_mesh
  SUBTYPE OF (unstructured_mesh);
  cells        : ARRAY [1:cell_count] OF vertex_defined_cell;
WHERE
  wr1 : SELF\mesh.index_count = 1;
END_ENTITY;
(*
```

Attribute definitions:

cells: the **vertex_defined_cell**s forming the mesh;

cell_count: (inherited) the number of cells in the mesh;

index_count: (inherited) the number of indices required to uniquely identify a vertex or cell in the mesh.

Formal propositions:

wr1: the value of **index_count** shall be 1.

4.3.2 array_based_unstructured_mesh_and_vertices

An **array_based_unstructured_mesh_and_vertices** is a kind of **array_based_unstructured_mesh** where the vertices of the mesh are explicity identified and ordered.

EXPRESS specification:

```
*)
ENTITY array_based_unstructured_mesh_and_vertices
  SUBTYPE OF (array_based_unstructured_mesh);
  vertex_count : INTEGER;
  vertices     : ARRAY [1:vertex_count] OF UNIQUE vertex;
WHERE
  wr1 : all_mesh_vertices(SELF);
END_ENTITY;
(*
```

Attribute definitions:

vertex_count: the number of unique **vertex** in the mesh;

vertices: an array of unique **vertex**s for the cells of the mesh.

Formal propositions:

wr1: the elements of **vertices** shall be all and only the unique vertices in the mesh.

4.3.3 cell

A **cell** is a **topological_representation_item** that is a manifold with a boundary.

EXPRESS specification:

```
*)
ENTITY cell
  SUPERTYPE OF (ONEOF(cell_of_structured_mesh, vertex_defined_cell))
  SUBTYPE OF (topological_representation_item);
  description : text;
  dimension   : INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

description: annotation;

dimension: the topological dimension of the region.

4.3.4 cell_with_explicit_boundary

A **cell_with_explicit_boundary** is a **cell** that has a specified boundary.

EXPRESS specification:

```
*)
ENTITY cell_with_explicit_boundary
  SUBTYPE OF (cell);
  boundary : SET [1:?] OF topological_representation_item;
END_ENTITY;
(*
```

Attribute definitions:

boundary: the elements forming the boundary of the region.

4.3.5 cell_of_structured_mesh

A **cell_of_structured_mesh** is an identified cell of a **structured_mesh**.

EXPRESS specification:

```
*)
ENTITY cell_of_structured_mesh
  SUBTYPE OF (cell);
  the_mesh : structured_mesh;
  cell_identifier : ARRAY [1:index_count] OF INTEGER;
DERIVE
  index_count : INTEGER := the_mesh\mesh.index_count;
END_ENTITY;
(*
```

Attribute definitions:

the_mesh: the **structured_mesh**;

cell_identifier: the indices of the cell;

index_count: the number of indices required to uniquely identify a vertex or cell in the mesh.

4.3.6 explicit_unstructured_mesh

An **explicit_unstructured_mesh** is a representation of an **unstructured_mesh** that is similar, but not entirely identical to, that specified in ISO 10303-104.

EXPRESS specification:

```
*)
ENTITY explicit_unstructured_mesh
  SUBTYPE OF (unstructured_mesh);
  explicit_model : fea_model;
  cells          : ARRAY [1:cell_count] OF UNIQUE element_representation;
END_ENTITY;
(*
```


Attribute definitions:

explicit_model: the finite element model;

cell_count: (inherited) the number of **element_representations**;

cells: the set of **element_representation**s comprising the mesh.

Informal propositions:

ip1: every **element_representation** in the **cells** shall belong to the **explicit_model**.

4.3.7 extraction_of_structured_submesh

An **extraction_of_structured_submesh** is a type of **extraction_of_mesh** and is a relationship between two **structured_mesh**es that indicates one is part of the other.

EXPRESS specification:

```
*)
ENTITY extraction_of_structured_submesh
SUBTYPE OF(extraction_of_submesh);
  lower_vertex : ARRAY [1:whole_indices] OF INTEGER;
  used_indices : ARRAY [1:part_indices] OF INTEGER;
  used_senses  : ARRAY [1:part_indices] OF BOOLEAN;
DERIVE
  whole_indices : INTEGER := SELF\extraction_of_submesh.whole\mesh.index_count;
  part_indices  : INTEGER := SELF\extraction_of_submesh.part\mesh.index_count;
WHERE
  WR1: ('MESH_TOPOLOGY_SCHEMA.STRUCTURED_MESH' IN  TYPEOF(
             SELF\extraction_of_submesh.whole));
  WR2: ('MESH_TOPOLOGY_SCHEMA.STRUCTURED_MESH' IN  TYPEOF(
             SELF\extraction_of_submesh.part));

END_ENTITY;
(*
```

Attribute definitions:

part: the **structured_mesh** that is part of the **whole**;

whole: the **structured_mesh** that contains the **part**;

lower_vertex: the position of the **vertex** in the **whole** that is the origin of the **part**. This is specified with respect to each index of the **whole**.

used_indices: the indices of the **whole** that are also indices of the **part** in the order that they are used in the **part**;

used_senses: the sense for each index of **part** as:

— TRUE if the **part** uses the index of the **whole** in the same direction;

— FALSE if the **part** uses the index of the **whole** in the reverse direction;

whole_indices: the number of indices required to uniquely identify a vertex or cell in the **whole**;

part_indices: the number of indices required to uniquely identify a vertex or cell in the **part**;

Formal propositions:

WR1: The **mesh** referenced as **whole** shall be of type **structured_mesh**.

WR1: The **submesh** referenced as **part** shall be of type **structured_mesh**.

4.3.8 extraction_of_submesh

An **extracrtion_of_submesh** is a relationship between a **mesh** and a **submesh** that defines the **submesh** as being part of the **mesh**.

EXPRESS specification:

```
*)
ENTITY extraction_of_submesh;
   whole:  mesh;
   part:   submesh;
END_ENTITY;
(*
```

Attribute definitions:

whole: the **mesh** from which the **submesh** is extracted.

part: the resulting **submesh**.

4.3.9 extraction_of_submesh_by_cells

An **extraction_of_submesh_by_cells** is a type of **extraction_of_submesh** that extracts a submesh by listing the cells.

EXPRESS specification:

```
*)
ENTITY extraction_of_submesh_by_cells
  SUBTYPE OF (extraction_of_submesh);
  cell_count: INTEGER;
  cells : ARRAY [1:cell_count] OF cell;
END_ENTITY;
(*
```

Attribute definitions:

cell_count: the number of cells extracted to form the submesh.

cells: the collection of **cell**s defining the **submesh**.

4.3.10 extraction_of_submesh_by_vertices

An **extraction_of_submesh_by_vertices** is a type of **extraction_of_submesh** that extracts a submesh by listing the vertices.

NOTE 1 A submesh of lower topological dimension than the parent (whole) mesh can be specified by **extraction_of_mesh_by_vertices**.

EXPRESS specification:

```
*)
ENTITY extraction_of_submesh_by_vertices
  SUBTYPE OF (extraction_of_submesh);
  vertex_count: INTEGER;
  vertices : ARRAY [1:vertex_count] OF vertex;
END_ENTITY;
(*
```

Attribute definitions:

vertex_count: the number of vertices extracted to form the submesh.

vertices: the collection of **vertex**s defining the **submesh**.

4.3.11 indices_list

An **indices_list** specifies a list of indices into a multi-dimensional array.

EXPRESS specification:

```
*)
ENTITY indices_list;
  nindices : INTEGER;
  indices  : LIST [1:?] OF ARRAY [1:nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: the number of indices required to map to a unique array location;

indices: the indices.

4.3.12 indices_range

An **indices_range** specifies the beginning and ending indices of a subrange in a multi-dimensional array.

EXPRESS specification:

```
*)
ENTITY indices_range;
  nindices : INTEGER;
  start    : ARRAY [1:nindices] OF INTEGER;
  finish   : ARRAY [1:nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: the number of indices required to map to a unique array location;

start: the indices of the minimal corner of the subrange;

finish: the indices of the maximal corner of the subrange.

4.3.13 mesh

A **mesh** is a **topological_representation_item** consisting of one or more cells. The **mesh** is the basis of all mesh topology representations. There are several ways of representing a mesh.

EXPRESS specification:

```
*)
ENTITY mesh
  ABSTRACT SUPERTYPE OF (ONEOF(structured_mesh,
        unstructured_mesh) ANDOR submesh)
   SUBTYPE OF (topological_representation_item);
  description : text;
  index_count : INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

description: annotation;

index_count: the number of indices required to identify uniquely a vertex or cell in the mesh.

NOTE 1 It inherits a **name** attribute of type **label** via its **topological_representation_item** supertype.

4.3.14 mesh_derived_maths_space

A **mesh_derived_maths_space** associates data values and a **mesh**.

NOTE The association commences with the **property_distribution_description** entity whose **abstract_function** attribute is of type **math_function** (a table function in this case) whose **range** and **domain** are of type **maths_space**, and hence to a **mesh**.

EXPRESS specification:

```
*)
ENTITY mesh_derived_maths_space
  SUBTYPE OF (maths_space);
  description : text;
  name        : label;
  id          : identifier;
  the_mesh    : mesh;
  kind        : mesh_maths_space_type;
END_ENTITY;
(*
```

Attribute definitions:

description: annotation;

name: a user interpretable identifier;

id: an identifier;

the_mesh: the **mesh**;

kind: the kind of association.

4.3.15 product_of_mesh

A **product_of_mesh** is a relationship that is between:

— two operands that are a 1-dimensional **mesh** and an n-dimensional **mesh**; and

— a product that is an $(n+1)$-dimensional **mesh**,

that indicates the $(n+1)$-dimensional **mesh** is the Cartesian product of the operands.

The ordering of cells and vertices of the product **mesh** is:

— cell $i + n(j-1)$ of the product mesh corresponds to cell i of the first operand and cell j of the second operand, where n is the total number of cells of the first operand;

— vertex $i + m(j-1)$ of the product mesh corresponds to vertex i of the first operand and vertex j of the second operand, where m is the total number of vertices of the first operand.

EXPRESS specification:

```
*)
ENTITY product_of_mesh;
  operands : LIST [2:2] OF mesh;
  product  : mesh;
WHERE
  WR1 : (this_schema+'.STRUCTURED_MESH' IN TYPEOF(operands[1])) AND
        (this_schema+'.STRUCTURED_MESH' IN TYPEOF(operands[2])) AND
        (this_schema+'.STRUCTURED_MESH' IN TYPEOF(product));
  WR2 : operands[1].index_count = 1;
  WR3 : operands[1].index_count + operands[2].index_count
        = product.index_count;
END_ENTITY;
(*
```

Attribute definitions:

operands: the two **mesh**es that define the **product**;

product: the **mesh** that is the Cartesian product of the **operands**.

Formal propositions:

WR1: all meshes shall be **structured_mesh**es;

WR2: the first operand shall have an **index_count** of one;

WR3: the **index_count** of the **product** shall equal the sum of the **index_count**s of the **operands**.

4.3.16 rind

A **rind** describes the number of rind planes associated with a structured mesh.

EXPRESS specification:

```
*)
ENTITY rind;
  index_count : INTEGER;
  planes      : ARRAY [1:2*index_count] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

index_count: the number of indices required to reference a vertex;

planes: contains the number of rind planes attached to the minimum and maximum faces of a structured mesh. The face corresponding to each index n of **planes** in 3-D is:

$$n = 1 \rightarrow i\text{-min} \qquad n = 2 \rightarrow i\text{-max}$$
$$n = 3 \rightarrow j\text{-min} \qquad n = 4 \rightarrow j\text{-max}$$
$$n = 5 \rightarrow k\text{-min} \qquad n = 6 \rightarrow k\text{-max}$$

EXAMPLE 1 For a 3-D mesh whose 'core' size is `II×JJ×KK`, a value of **planes** = `[a,b,c,d,e,f]` indicates that the range of indices for the mesh with this rind is:

i: `(1 - a, II + b)`
j: `(1 - c, JJ + d)`
k: `(1 - e, KK + f)`

4.3.17 structured_mesh

A **structured_mesh** has a regular topology. A **structured_mesh** has a parametric coordinate system; the parametric coordinate systems for one- two-, and three-dimensional structured meshes are shown in Figure 11 through Figure 13.

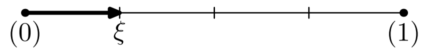


Figure 11 – Parametric coordinate system for a 1-D structured mesh

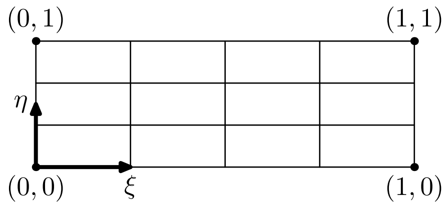


Figure 12 – Parametric coordinate system for a 2-D structured mesh

For each cell within a **structured_mesh**, the parametric coordinate system for that cell is identical to the parametric coordinate system for the mesh, except for an origin shift. The parametric coordinates of vertex (i, j, k) in a 3-D mesh with n, m and p cells in the 3 dimensions are:

$$((i-1)/n, (j-1)/m, (k-1)/p)$$

EXPRESS specification:

```
*)
ENTITY structured_mesh
  SUBTYPE OF (mesh);
  vertex_counts : ARRAY [1:SELF\mesh.index_count] OF INTEGER;
  cell_counts   : ARRAY [1:SELF\mesh.index_count] OF INTEGER;
```

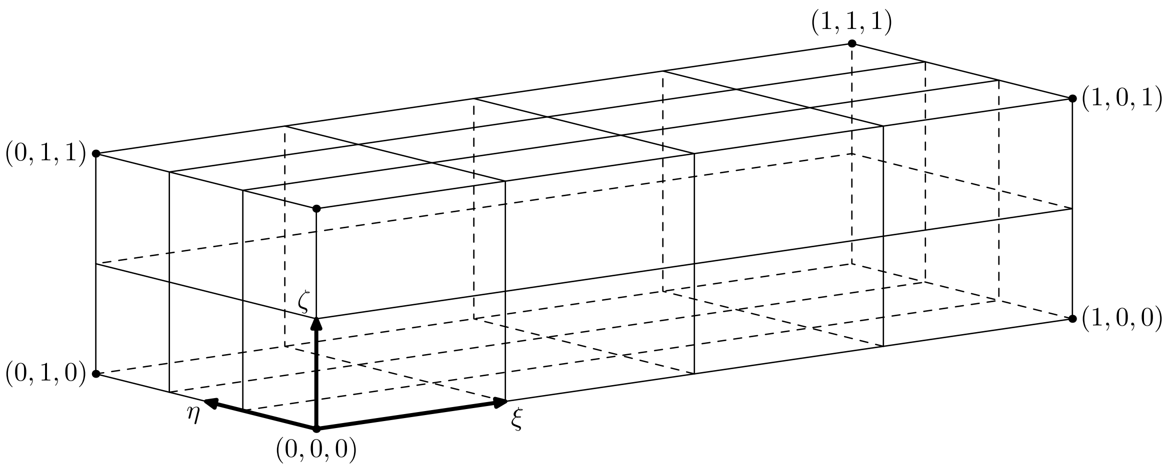


Figure 13 – Parametric coordinate system for a 3-D structured mesh

Table 1 – Number of vertices in a structured_mesh

Index_count	Rectangular	Pentahedral	Pyramidal	Tetrahedral
1	i	i	i	i
2	ij	$j(i-1)+1$	$j(i-1)+1$	$j(i-1)+1$
3	ijk	$jk(i-1)+k$	$jk(i-1)+1$	$jk(i-1)-(i-2)(j-1)$

```
  kind          : structured_mesh_type;
END_ENTITY;
(*
```

Attribute definitions:

vertex_counts: the number of vertices in each dimension of the mesh. The product of the array elements is the number of vertices defining the mesh (i.e., excluding any rind points). The number of vertices in one- two- and three-dimensional regular mesh topologies is given in Table 1, where i, j and k correspond to the array elements `vertex_counts[1]`, `vertex_counts[2]` and `vertex_counts[3]`, respectively.

cell_counts: the number of cells in each dimension of the mesh. The product of the array elements is the number of cells on the interior of the mesh;

kind: the kind of mesh;

index_count: (inherited) the number of indices required to identify uniquely a vertex or cell in the mesh is the same as the topological dimensionality (e.g., 1–D, 3–D) of the mesh.

4.3.18 structured_mesh_with_rind

A **structured_mesh_with_rind** is a **structured_mesh** with specified **rind** planes.

EXPRESS specification:

```
*)
ENTITY structured_mesh_with_rind
  SUBTYPE OF (structured_mesh);
  rind_planes : rind;
END_ENTITY;
(*
```

Attribute definitions:

rind_planes: the **rind** planes associated with the mesh.

4.3.19 submesh

A **submesh** is a **mesh** that is part of another **mesh**. The **submesh** is related to the parent **mesh** by an **extraction_of_submesh** relationship. The cells of the **submesh** are not necessarily connected.

EXPRESS specification:

```
*)
ENTITY submesh
  SUBTYPE OF (mesh);
END_ENTITY;
(*
```

4.3.20 unstructured_mesh

An **unstructured_mesh** is composed of cells, where the cells do not form a regular pattern and the shape of the cells is not uniform throughout the mesh. Cells have vertices at the ends of cell edges, and may also have nodes on cell edges, cell faces and in the interior of the cell.

It conceptually consists of the vertices of the mesh and the cells forming the volume of the mesh. The cells shall all be connected by each cell having at least one vertex in common with another cell. The shape of each cell in an unstructured mesh is explicitly specified.

NOTE 1 A vertex at the corner of a cell is a cell vertex.

EXPRESS specification:

```
*)
ENTITY unstructured_mesh
  ABSTRACT SUPERTYPE OF (ONEOF(array_based_unstructured_mesh,
        explicit_unstructured_mesh))
  SUBTYPE OF (mesh);
  cell_count   : INTEGER;
END_ENTITY;


(*
```

Attribute definitions:

cell_count: the number of cells in the mesh.

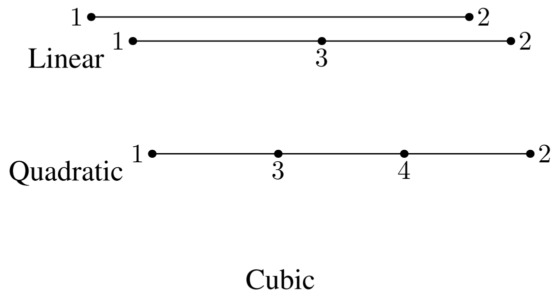


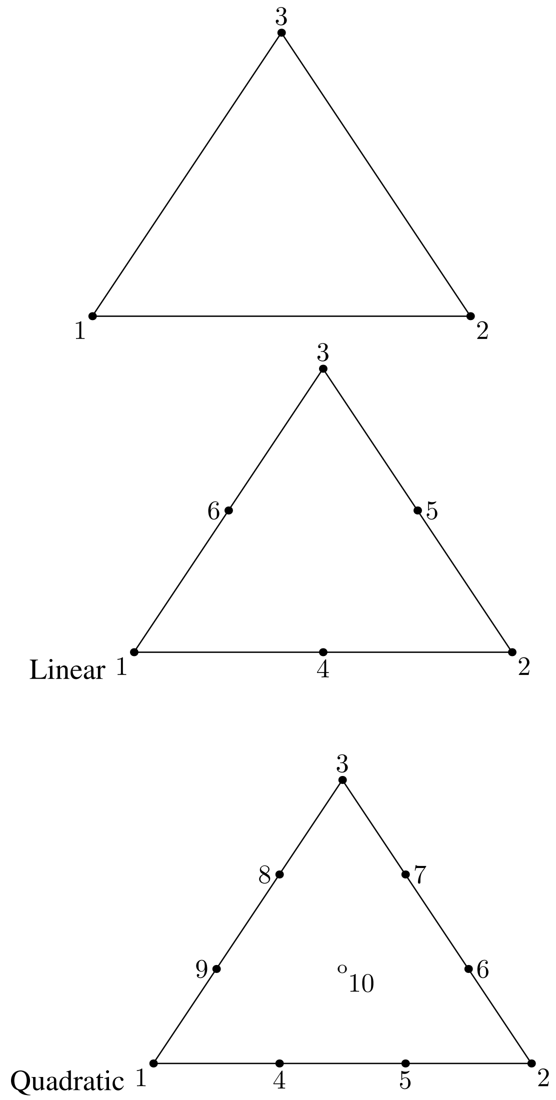
Figure 14 – Linear, quadratic and cubic line cells

Table 2 – Edges of triangle, quadrilateral and polygon cells

triangle		quadrilateral		n-sided polygon	
edge	**vertices**	**edge**	**vertices**	**edge**	**vertices**
1	1, 2	1	1, 2	1	1, 2
2	2, 3	2	2, 3	2	2, 3
3	3, 1	3	3, 4	3	3, 4
		4	4, 1	n	n, 1

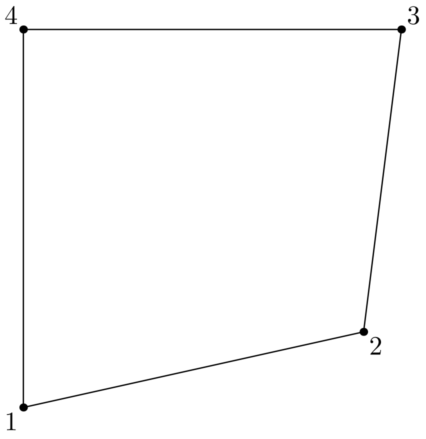
Table 3 – Edges of hexahedron, wedge, tetrahedron and pyramid cells

hexahedron		wedge		tetrahedron		pyramid	
edge	**vertices**	**edge**	**vertices**	**edge**	**vertices**	**edge**	**vertices**
1	1, 2	1	1, 2	1	1, 2	1	1, 2
2	2, 3	2	2, 3	2	2, 3	2	2, 3
3	3, 4	3	3, 1	3	3, 1	3	3, 4
4	4, 1	4	4, 5	4	1, 4	4	4, 1
5	5, 6	5	5, 6	5	2, 4	5	1, 5
6	6, 7	6	6, 4	6	3, 4	6	2, 5
7	7, 8	7	1, 4			7	3, 5
8	8, 5	8	2, 5			8	4, 5
9	1, 5	9	3, 6				
10	2, 6						
11	3, 7						
12	4, 8						

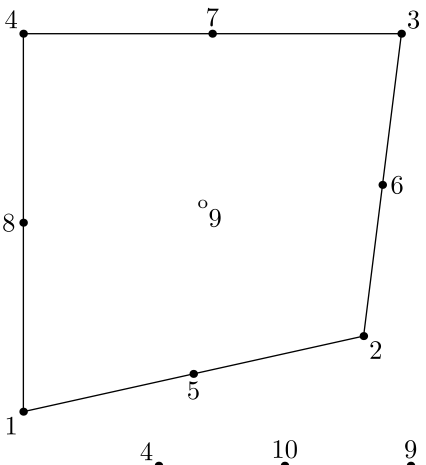


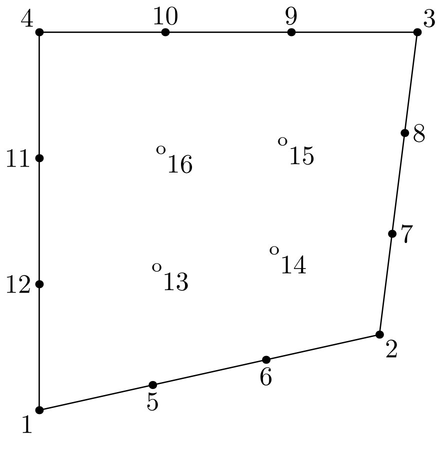
Cubic

Figure 15 – Linear, quadratic and cubic triangle cells



Linear





Quadratic

Cubic

Figure 16 – Linear, quadratic and cubic quadrilateral cells

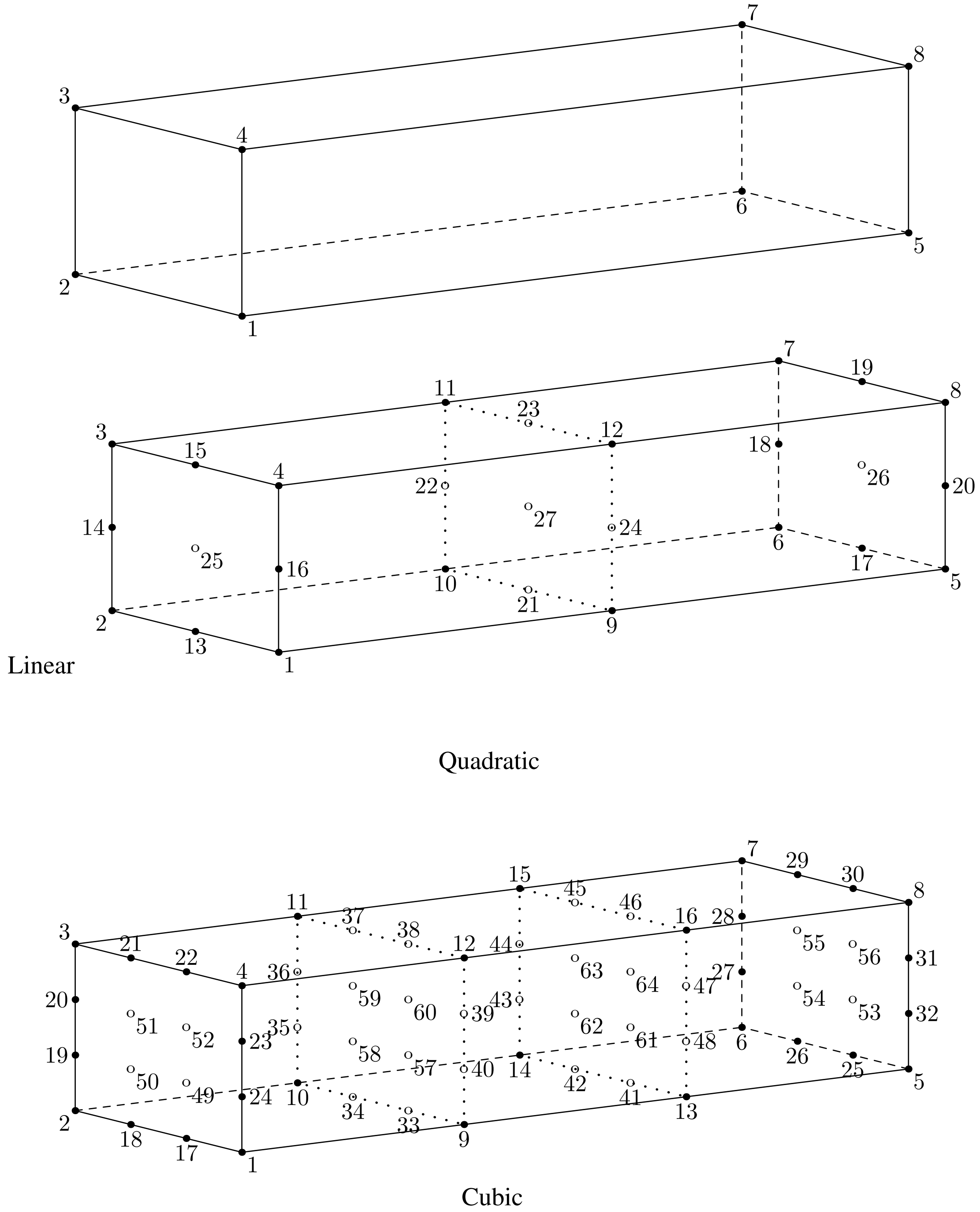


Figure 17 – Linear, quadratic and cubic hexahedron cells

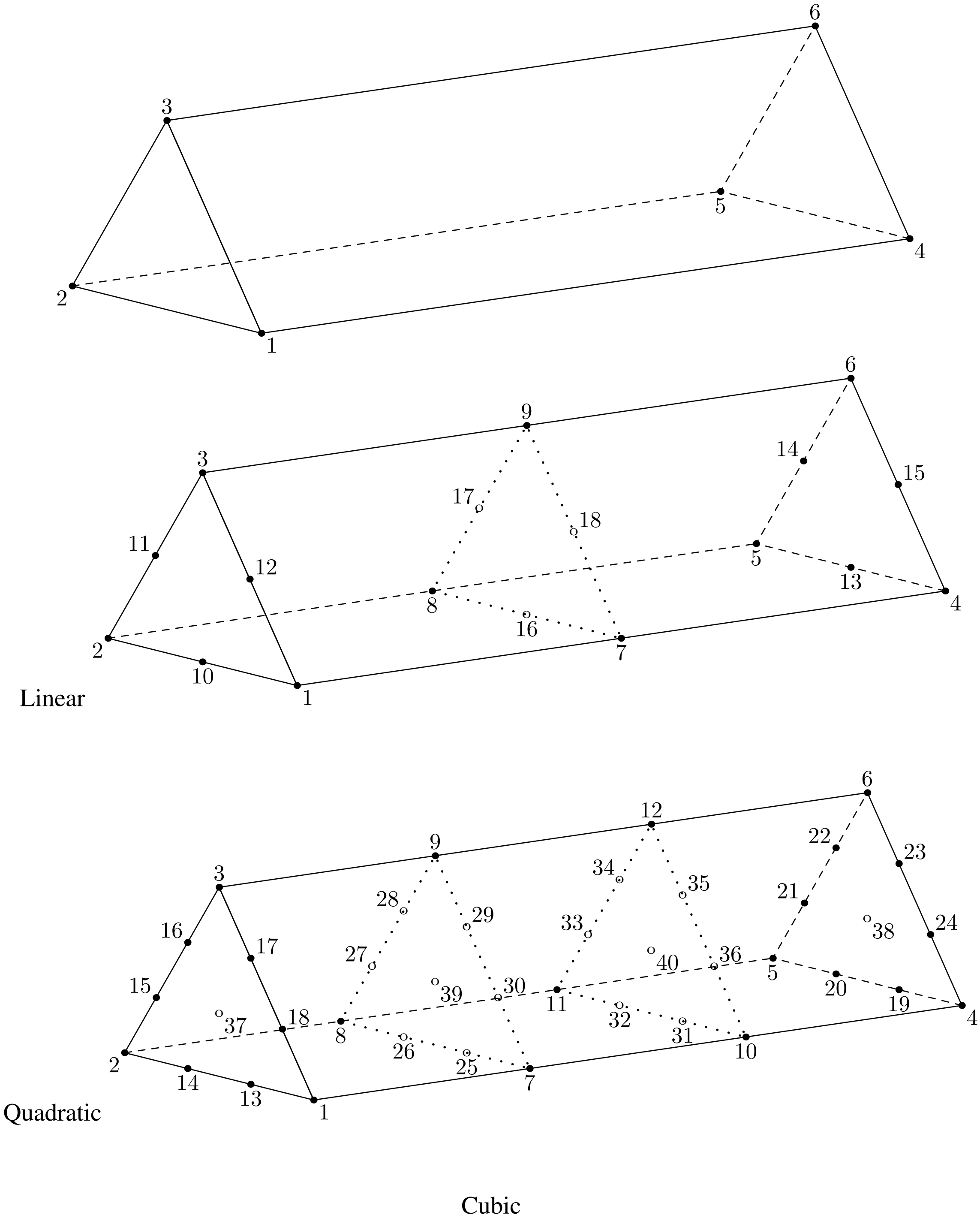


Figure 18 – Linear, quadratic and cubic wedge cells

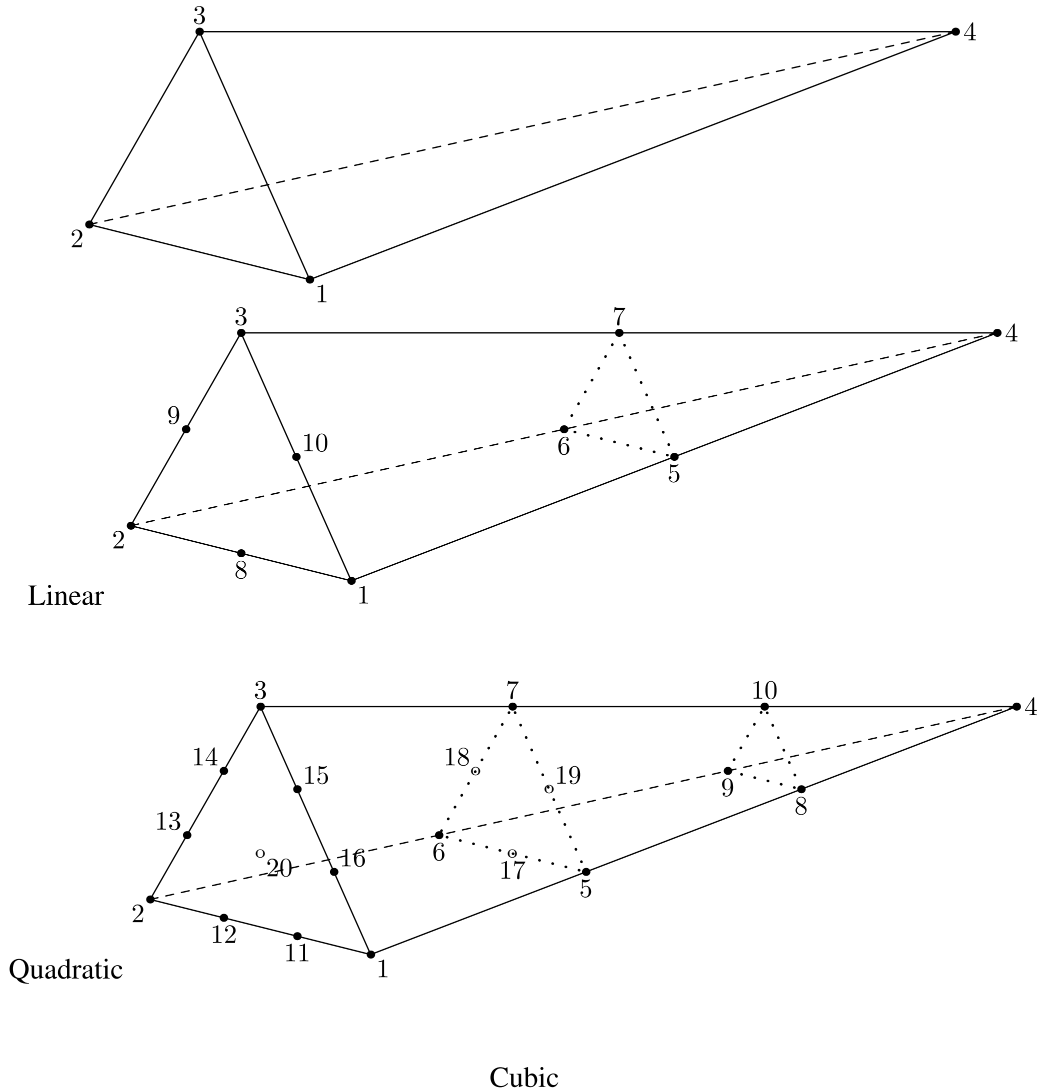
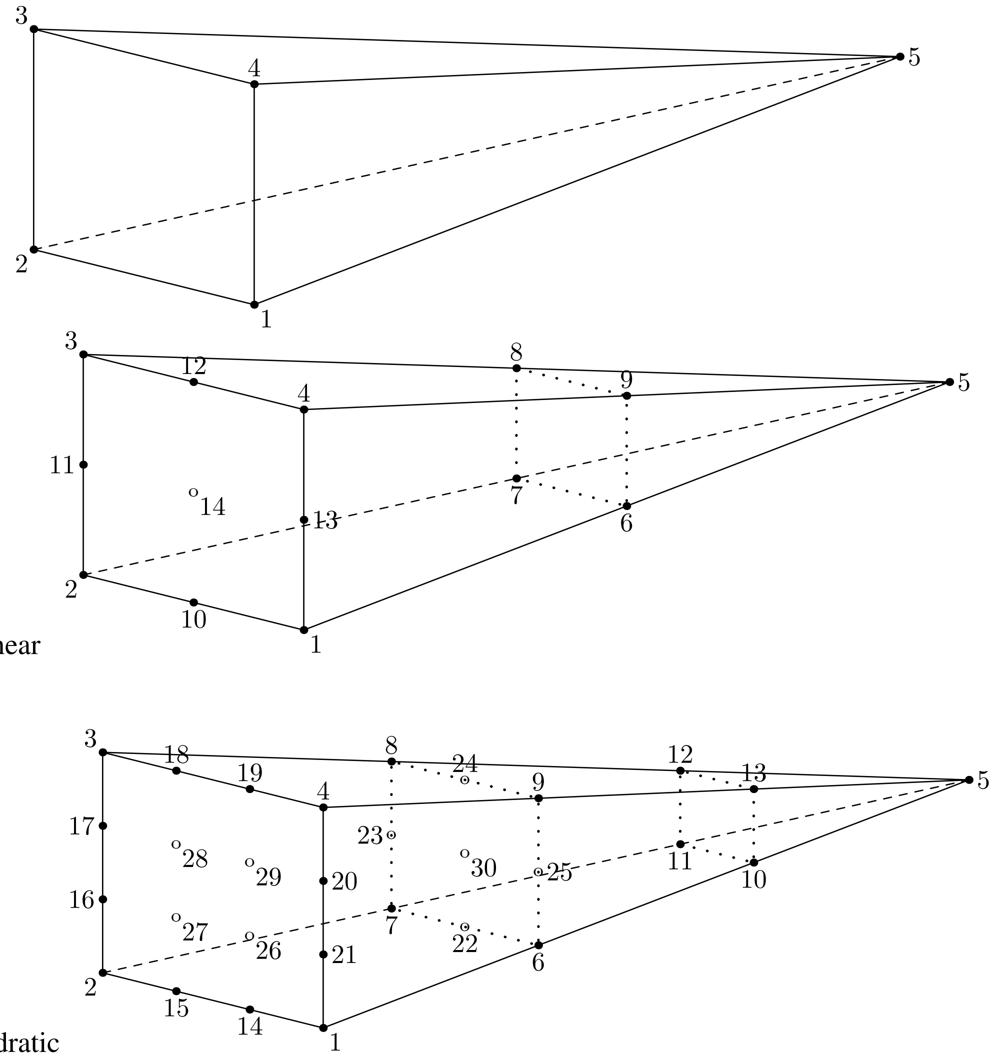


Figure 19 – Linear, quadratic and cubic tetrahedron cells

Table 4 – Faces of hexahedron, wedge, tetrahedron and pyramid cells

hexahedron		wedge		tetrahedron		pyramid	
face	**vertices**	**face**	**vertices**	**face**	**vertices**	**face**	**vertices**
1	1, 4, 3, 2	1	1, 3, 2	1	1, 2, 3	1	1, 4, 3, 2
2	5, 6, 7, 8	2	4, 5, 6	2	1, 4, 2	2	1, 2, 5
3	1, 2, 6, 5	3	1, 2, 4, 5	3	2, 4, 3	3	2, 3, 5
4	3, 7, 6, 2	4	2, 3, 6, 5	4	3, 4, 1	4	3, 4, 5
5	3, 4, 8, 7	5	1, 4, 6, 3			5	4, 1, 5
6	1, 5, 8, 4						



Linear

Quadratic

Cubic

Figure 20 – Linear, quadratic and cubic pyramid cells

4.3.21 vertex_defined_cell

A **vertex_defined_cell** is a cell that is defined by its vertices. The number of vertices depends on the topological shape of the cell. The cell may have edge, face and interior vertices. The maximum number of edge, face and interior vertices depends on both the shape and the order of the cell.

EXPRESS specification:

```
*)
ENTITY vertex_defined_cell
  SUBTYPE OF (cell);
  shape       : cell_shape;
  order       : element_order;
  vertices    : ARRAY [1:vn_count] OF OPTIONAL vertex;
DERIVE
  bound_count        : INTEGER := cell_counts(SELF)[1];
  edge_node_count    : INTEGER := cell_counts(SELF)[2];
  opt_node_count     : INTEGER := cell_counts(SELF)[3];
  required_count     : INTEGER := bound_count + edge_node_count;
  vn_count           : INTEGER := required_count + opt_node_count;
  required_vertices : ARRAY [1:required_count] OF vertex
                         := shorten_array(vertices, vn_count, required_count);
END_ENTITY;
(*
```

Attribute definitions:

shape: the topological shape of the cell;

order: the order of the cell geometric interpolation;

vertices: the vertices at the ends of cell edges, and within cell edges, cell faces and the interior of the cell. The position of a vertex or an edge node in the array depends on the shape of the cell as established graphically in Figures 10 through 39 in ISO 10303-104, noting that a **polygon** cell is a generalisation of the **triangle** and **quadrilateral** cells.

NOTE 1 For convenience, most of the ISO 10303-104 Figures have been redrawn in this part of ISO 10303 as Figure 14 to Figure 20, where a vertex or edge node is indicated by a dot. The vertex labelled '1' is the first index in the array, that labelled '2' is the second index in the array, and so on. Edge and face information from ISO 10303-104 is given in Table 2 to Table 4.

bound_count: the number of cell bounding vertices; it is determined by the value of **shape**;

edge_node_count: the number of interior cell nodes located on the cell edges; it is determined by the combination of the values of **shape** and **order**;

opt_node_count: the potential number of interior cell nodes which are not located on the cell edges; it is determined by the combination of the values of **shape** and **order**;

NOTE 2 In Figure 14 to Figure 20 the non-edge interior nodes are indicated by circles.

required_count: the total number of bounding vertices plus the number of edge nodes;

vn_count: the total number of bounding vertices plus the number of nodes (both edge and non-dege nodes);

required_vertices: the vertices and nodes excluding any non-edge interior nodes. There shall be **required_count** of these.

4.4 mesh_topology_schema function definitions

4.4.1 all_mesh_vertices

The function **all_mesh_vertices** takes an **array_based_unstructured_mesh_and_vertices** as its argument and returns TRUE if the members of the **vertices** attribute are exactly the vertices in the mesh.

EXPRESS specification:

```
*)
FUNCTION all_mesh_vertices(arg : array_based_unstructured_mesh_and_vertices)
         : BOOLEAN;
LOCAL
  vertex_set : SET OF vertex := [];
  cell : vertex_defined_cell;
END_LOCAL;
  REPEAT i := 1 TO arg.cell_count;
    cell := arg.cells[i];
    REPEAT j := 1 TO cell.vn_count;
      vertex_set := vertex_set + cell.vertices[j];
    END_REPEAT;
  END_REPEAT;
  IF (SIZEOF(vertex_set) <> arg.index_count) THEN
    RETURN(FALSE);
  END_IF;
  REPEAT i := 1 TO arg.index_count;
    IF (NOT (arg.vertices[i] IN vertex_set) ) THEN
      RETURN(FALSE);
    END_IF;
  END_REPEAT;
  RETURN(TRUE);
END_FUNCTION;
(*
```

Argument definitions:

arg: an **array_based_unstructured_mesh_and_vertices**;

RETURNS: TRUE if the members of the **vertices** attribute of **arg** are exactly the vertices in the mesh, otherwise FALSE.

4.4.2 cell_counts

The function **cell_counts** takes a **vertex_defined_cell** as its argument and returns the numbers of vertices and nodes required to define the cell.

EXPRESS specification:

```
*)
FUNCTION cell_counts(arg : vertex_defined_cell) : ARRAY[1:3] OF INTEGER;
LOCAL
  om1    : INTEGER := 0;       -- (order - 1)
  om1sq  : INTEGER := om1**2; -- (order - 1) squared
  vts    : INTEGER;            -- number of bounding vertices
  eds    : INTEGER;            -- number of edges
  qf     : INTEGER := 0;       -- number of quadrilateral faces
  tf     : INTEGER := 0;       -- number of triangular faces
  result : ARRAY [1:3] OF INTEGER := [0,0,0];
END_LOCAL;
  CASE arg.order OF
    linear    : om1 := 0;
    quadratic : om1 := 1;
    cubic     : om1 := 2;
    OTHERWISE : RETURN(result);
  END_CASE;
  om1sq := om1**2;
  CASE arg.shape OF
    single :
        BEGIN
          vts := 1; eds := 0; qf := 0; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 0, 0
          result[3] := 0;                          -- 0, 0, 0
        END;
    line :
        BEGIN
          vts := 2; eds := 1; qf := 0; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 1, 2
          result[3] := 0;                          -- 0, 0, 0
        END;
    quadrilateral :
        BEGIN
          vts := 4; eds := 4; qf := 1; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 4, 8
          result[3] := om1sq*qf;                   -- 0, 1, 4
```



```
    END;
triangle :
    BEGIN
      vts := 3; eds := 3; qf := 0; tf := 1;
      result[1] := vts;
      result[2] := om1*eds;                  -- 0, 3, 6
      result[3] := (om1-1)*tf;               --    0, 1
      CASE arg.order OF
        linear : result[3] := 0;             -- 0
      END_CASE;
    END;
polygon :
    BEGIN
      vts := arg.vn_count; eds := arg.vn_count;
      result[1] := vts;
      result[2] := 0;
      result[3] := 0;
    END;
hexahedron :
    BEGIN
      vts := 8; eds := 12; qf := 6; tf := 0;
      result[1] := vts;
      result[2] := om1*eds;                  -- 0, 12, 24
      result[3] := om1sq*(qf+om1);           -- 0, 7, 32
    END;
wedge :
    BEGIN
      vts := 6; eds := 9; qf := 3; tf := 2;
      result[1] := vts;
      result[2] := om1*eds;                  -- 0, 9,  18
      result[3] := om1sq*qf + om1*tf;        -- 0, 3, 16
    END;
tetrahedron :
    BEGIN
      vts := 4; eds := 6; qf := 0; tf := 4;
      result[1] := vts;
      result[2] := om1*eds;                  -- 0, 6, 12
      result[3] := (om1-1)*tf;               --    0, 4
      CASE arg.order OF
        linear : result[3] := 0;             -- 0
      END_CASE;
    END;
pyramid :
    BEGIN
      vts := 5; eds := 8; qf := 1; tf := 4;
      result[1] := vts;
      result[2] := om1*eds;                  -- 0, 8, 16
      result[3] := om1sq*qf + (om1-1)*tf;    --    1, 9
      CASE arg.order OF
        linear : result[3] := 0;             -- 0
      END_CASE;
    END;
```

```
  END_CASE;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: a **cell**;

RETURNS: a 3 element array of INTEGER, where the first element is the number of vertices defining the bounds of the cell, the second is the number of interior nodes located on an edge, and the third is the maximum number of (potential) interior nodes not located on an edge.

4.4.3 shorten_array

The function shorten_array inputs a one dimensional array and returns a shorter one dimensional array containing the first **newl** elements of the input array.

EXPRESS specification:

```
*)
FUNCTION shorten_array(longa : ARRAY OF GENERIC : T;
                      oldl, newl : INTEGER) : ARRAY OF GENERIC : T;
   LOCAL
     shorta : ARRAY [1:newl] OF GENERIC : T;
   END_LOCAL;

   IF (newl > oldl) THEN
     RETURN(?);
   ELSE
     REPEAT i := 1 TO newl;
       shorta[i] := longa[i];
     END_REPEAT;
     RETURN(shorta);
   END_IF;
END_FUNCTION;
(*
```

Argument definitions:

longa: The input array.

oldl: The number of elements in **longa**.

newl: The smaller number of elements in the output array.

RETURN: The shortened array of length **newl**

4.4.4 this_schema

The function **this_schema** returns a STRING containing the name of the schema.

EXPRESS specification:

```
*)
FUNCTION this_schema : STRING;
  RETURN('MESH_TOPOLOGY_SCHEMA');
END_FUNCTION;
(*
```

Argument definitions:

RETURNS: the uppercase name of the schema.

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of mesh_topology_schema
(*
```

5 Mesh connectivity

The following EXPRESS declaration begins the **mesh_connectivity_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA mesh_connectivity_schema;
  REFERENCE FROM mesh_topology_schema                            -- ISO 10303-52
    (mesh,
     unstructured_mesh,
     structured_mesh,
     mesh_location,
     indices_group,
     indices_range);
  REFERENCE FROM mathematical_description_of_distribution_schema -- ISO 10303-51
    (property_distribution_description);
  REFERENCE FROM mathematical_functions_schema                   -- ISO 10303-50
    (listed_real_data);
```

```
  REFERENCE FROM support_resource_schema                          -- ISO 10303-41
    (identifier,
     label,
     text);
(*
```

NOTE The schemas referenced above can be found in the following parts of ISO 10303:

mesh_topology_schema	Clause 4 of this part of ISO 10303
mathematical_description_of_distribution_schema	ISO 10303-52
mathematical_functions_schema	ISO 10303-50
support_resource_schema	ISO 10303-41

5.1 General

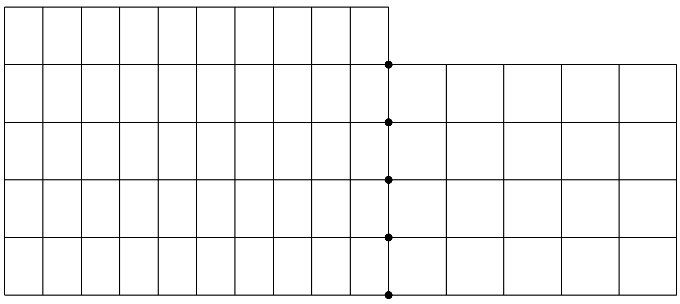
This schema defines and describes the structures for describing interface connectivity between meshes.

5.2 Fundamental concepts and assumptions

Meshes have interfaces which are either abutting or overlapping. For the abutting interface this can be either matched, or mis-matched.This clause describes and illustrates such interfaces.

NOTE 1 Figures 21 to Figure 23 show three types of mesh interfaces.

EXAMPLE 1 Figure 21 illustrates a 1-to-1 abutting interface, also referred to as matching or C0 continuous. The interface is a plane of vertices that are physically coincident (i.e., they have identical coordinate values) between the adjacent meshes; mesh-coordinate lines perpendicular to the interface are continuous from one mesh to the next. In 3-D, a 1-to-1 abutting interface is always a logically rectangular region.



• left-mesh vertices on interface

Figure 21 – A 1-to-1 abutting interface

EXAMPLE 2 The second type of interface, is mismatched abutting, where two meshes touch but do not overlap (except for vertices and cell faces on the mesh plane of the interface). Vertices on the interface need not be physically concident between the two meshes. Figure 22 indentifies the vertices and face centers of the left mesh that lie on the interface. In 3-D, the vertices of a mesh that constitute an interface patch may not form a logically rectangular region.

The third type of interface is called overset and occurs when two meshes overlap; in 3-D, the overlap is a 3-D region. For overset interfaces, one of the two meshes takes precedence over the other; this establishes which solution in the overlap region to retain and which one to discard. The region in a given

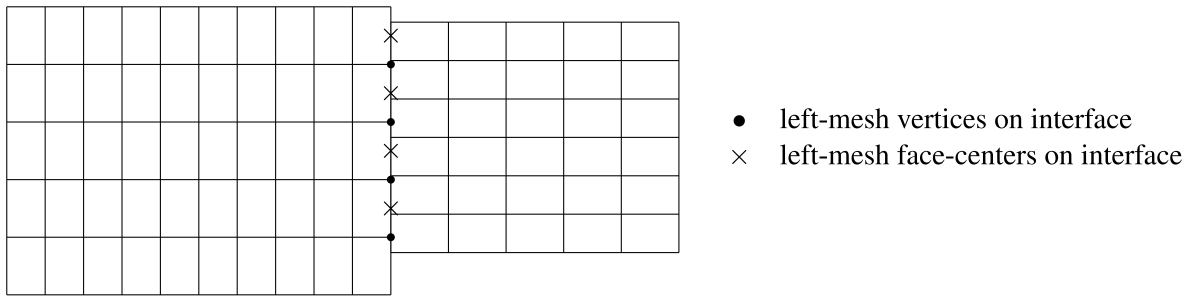


Figure 22 – A mismatched abutting interface

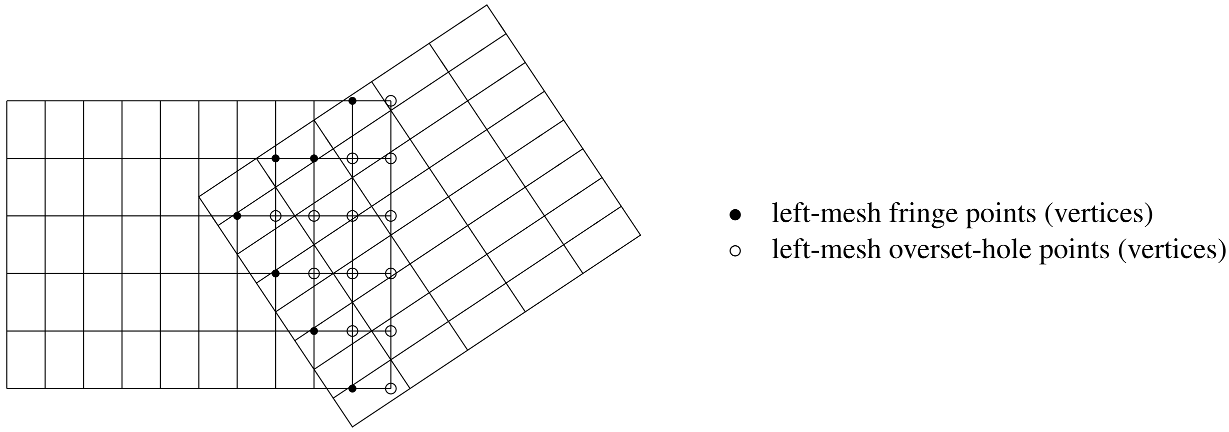


Figure 23 – An overset interface

mesh where the solution is discarded is called an overset hole and the mesh vertices outlining the hole are called fringe points.

EXAMPLE 3 Figure 23 depicts an overlap region between two meshes, where the right mesh takes precedence over the left mesh. The points identified in Figure 23 are the fringe points and overset-hole points for the left mesh. In addition, for the mesh taking precedence, any bounding points (i.e., vertices on the bounding faces) of the mesh that lies within the overlap must also be identified.

Overset interfaces may include multiple layers of fringe points outlining holes and at mesh boundaries.

For the mismatched abutting and overset interfaces in Figure 22 and Figure 23, the left mesh plays the role of receiver mesh and the right plays the role of donor mesh.

Mesh vertices that are included in a mesh interface are, in general, termed interface points.

5.3 mesh_connectivity_schema type definitions

5.3.1 mismatched_region_type

A **mismatched_region_type** is an enumeration of the kinds of mismatched mesh regions.

EXPRESS specification:

```
*)
TYPE mismatched_region_type = EXTENSIBLE ENUMERATION OF
    (abutting,
     overset);
END_TYPE;
(*
```

Enumerated item definitions:

abutting: abutting region;

overset: overset region.

5.4 mesh_connectivity_schema entity definitions

5.4.1 matched_mesh_connection

A **matched_mesh_connection** contains connectivity information for a mesh interface patch that is abutting with 1-to-1 matching between adjacent structured mesh indices (also referred to as C0 connectivity). An interface patch is the subrange of the face of a mesh that touches one and only one other mesh. This structure identifies the subrange of indices for the two adjacent meshes that make up the interface and gives an index transformation from one mesh to the other. It also identifies the adjacent mesh.

EXPRESS specification:

```
*)
ENTITY matched_mesh_connection
  SUBTYPE OF (mesh_connectivity);
  SELF\mesh_connectivity.current : structured_mesh;
  range       : indices_range;
  donor       : structured_mesh;
  donor_range : indices_range;
  transform   : ARRAY [1:index_count] OF INTEGER;
WHERE
  WR1 : current :<>: donor;
  WR2 : donor.index_count = index_count;
```



```
  WR3 : range.nindices = index_count;
  WR4 : donor_range.nindices = index_count;
END_ENTITY;
(*
```

Attribute definitions:

current: (inherited) the current mesh;

index_count: (inherited) the number of indices required to reference a vertex;

range: contains the subrange of indices that makes up the interface patch in the **block**;

donor: the adjacent mesh;

donor_range: contains the interface patch subrange of indices for the **donor**;

transform: contains a shorthand notation for the transformation matrix describing the relationship between indices of the two adjacent meshes (see below).

Formal propositions:

WR1: **current** and **donor** shall be different;

WR2: the **index_count**s of **current** and **donor** shall have the same value;

WR3: the **index_count**s of **current** and **range** shall have the same value;

WR4: the **index_count**s of **donor** and **donor_range** shall have the same value.

The shorthand matrix notation used in **transform** has the following properties. The matrix itself has rank **index_count** and contains elements $+1$, 0, and -1; it is orthonormal and its inverse is its transpose. The transformation matrix (`T`) works as follows: If `Index1` and `Index2` are the indices of a given point on the interface, where `Index1` is in the current mesh and `Index2` is in the adjacent mesh, then their relationship is,

```
Index2 = T.(Index1 - Start1) + Start2
Index1 = Transpose[T].(Index2 - Start2) + Start1
```

where the '`.`' notation indicates matrix-vector multiply, `Start1` and `Finish1` are the subrange indices contained in **range**, and `Start2` and `Finish2` are the subrange indices contained in **donor_range**.

The short-hand notation used in **transform** is as follows. Each element shows the image in the adjacent mesh's face of a positive index increment in the current mesh's face. The first element is the image of a positive increment in i; the second element is the image of an increment in j; and the third (in 3-D) is the image of an increment in k in the current mesh's face. For 3-D, the transformation matrix T is constructed from **transform** = $[\pm a, \pm b, \pm c]$ as follows:

$$\texttt{T} = \begin{bmatrix} \text{sgn}(a)\text{del}(a-1) & \text{sgn}(b)\text{del}(b-1) & \text{sgn}(c)\text{del}(c-1) \\ \text{sgn}(a)\text{del}(a-2) & \text{sgn}(b)\text{del}(b-2) & \text{sgn}(c)\text{del}(c-2) \\ \text{sgn}(a)\text{del}(a-3) & \text{sgn}(b)\text{del}(b-3) & \text{sgn}(c)\text{del}(c-3) \end{bmatrix},$$

where,

$$\text{sgn}(x) \equiv \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \qquad \text{del}(x-y) \equiv \begin{cases} 1, & \text{if abs}(x) = \text{abs}(y) \\ 0, & \text{otherwise} \end{cases}$$

EXAMPLE 1 **transform** = `[-2, +3, +1]` gives the transformation matrix,

$$\texttt{T} = \begin{bmatrix} 0 & 0 & +1 \\ -1 & 0 & 0 \\ 0 & +1 & 0 \end{bmatrix}$$

NOTE 1 For establishing relationships between adjacent and current mesh indices lying on the interface itself, one of the elements of **transform** is superfluous since one component of both interface indices remains constant.

NOTE 2 The transform matrix and the two index pairs overspecify the interface patch. For example, `Finish2` can be obtained from **transform**, `Start1`, `Finish1` and `Start2`.

5.4.2 mesh_connectivity

A **mesh_connectivity** specifies the connectivity of a mesh interface.

EXPRESS specification:

```
*)
ENTITY mesh_connectivity
 ABSTRACT SUPERTYPE OF(ONEOF
    (matched_mesh_connection,
        mismatched_mesh_connection);
  name        : label;
  description : text;
  id          : identifier;
  current     : mesh;
DERIVE
  index_count : INTEGER := current.index_count;
END_ENTITY;

(*
```

Attribute definitions:

name: user-specified instance identifier;

description: annotation;

id: an identifier;

current: the current (receiver) mesh;

index_count: the number of indices required to identify uniquely a vertex or cell in the block.

5.4.3 mesh_overset_hole

An **overset_hole** is a hole, or holes, in an overset mesh.

Mesh connectivity for overset meshes may also include 'holes' within meshes, where any mesh data is ignored or 'turned off', because the data in some other overlapping mesh applies instead.

EXPRESS specification:

```
*)
ENTITY mesh_overset_hole
  SUBTYPE OF (mismatched_mesh_connection);
END_ENTITY;
(*
```

NOTE 1 The interface points making up a hole within a mesh may be specified by an element in the **range** list if they constitute a logically rectangular region. Likewise further elements in the list may be used for further logically rectangular holes. The more general alternative is to use **vertices** to list all interface points making up the holes within a mesh. Using the list of **range** specifications, or using **range** in combination with **vertices**, may result in a given hole being specified more than once.

5.4.4 mismatched_donor_mesh

A **mismatched_donor_mesh** is a mesh that acts as a donor for a **mismatched_mesh_region**.

EXPRESS specification:

```
*)
ENTITY mismatched_donor_mesh
ABSTRACT SUPERTYPE OF(ONEOF (structured_donor_mesh, unstructured_donor_mesh));
  donor : mesh;
INVERSE
  connect : mismatched_mesh_region FOR donor;
END_ENTITY;
(*
```

Attribute definitions:

donor: the donor mesh.

connect: the **mismatched_mesh_region** for which this is the **donor**.

5.4.5 mismatched_mesh_connection

A **mismatched_mesh_connection** contains conectivity information for generalized mesh interfaces. Its purpose is to describe mismatched-abutting and overset interfaces for both structured and unstructured meshes, and can also be used for 1-to-1 abutting interfaces.

For abutting interfaces, also referred to as patched or mismatched, an interface patch is the subrange of the face of a mesh that touches one and only one other mesh. This structure identifies the subrange of indices (or array of indices) that make up the interface and gives their image in the adjacent (donor) mesh. It also identifies the adjacent mesh. If a given face of a mesh touches several (say N) adjacent meshes, then N different instances of **mismatched_mesh_connection** are needed to describe all the interfaces. For a single abutting interface, two instances of **mismatched_mesh_connection** are needed — one for each adjacent mesh.

For overset interfaces, this structure identifies the fringe points of a given mesh that lie in one and only one other mesh. If the fringe points of a mesh lie in several (say N) overlapping meshes, then N different instances of **mismatched_mesh_connection** are needed to describe the overlaps. It is possible with overset meshes that a single fringe point may actually lie in several overlapping meshes (though in typical usage, linkage to only one of the overlapping meshes is kept). There is no restriction against a given fringe point being contained within multiple instances of **mismatched_mesh_connection**; therefore, this structure allows the description of a single fringe point lying in several overlapping meshes.

EXPRESS specification:

```
*)
ENTITY mismatched_mesh_connection
ABSTRACT SUPERTYPE OF (ONEOF(mismatched_mesh_region,
        mesh_overset_hole))
 SUBTYPE OF (mesh_connectivity);
  points   : indices_group;
  gridloc  : mesh_location;
END_ENTITY;
(*
```

Attribute definitions:

current: (inherited) the current mesh (the receiver mesh);

index_count: (inherited) the number of indices required to reference a vertex;

points: the indices of the interface points within the current mesh;

gridloc: the location of indices within the current mesh described by **points**. It also identifies the location of indices described by an **index_range** in a donor mesh. This allows the flexibility to describe overset interfaces for cell-centered quantities.

5.4.6 mismatched_mesh_region

A **mismatched_mesh_region** is a mismatched connection that is abutting or overset.

EXPRESS specification:

```
*)
ENTITY mismatched_mesh_region
  SUBTYPE OF (mismatched_mesh_connection);
  donor : mismatched_donor_mesh;
  kind  : mismatched_region_type;
WHERE
  WR1 : donor.donor :<>: SELF\mesh_connectivity.current;
END_ENTITY;
(*
```

Attribute definitions:

donor: an adjacent structured or unstructured donor mesh;

kind: the kind of connection.

Formal propositions:

WR1: the donor mesh shall not be the same as the **current**.

Informal propositions:

ip1: when the **kind** is **abutting** the **range** or **vertices** shall describe a face subrange (i.e., points in a single computational mesh plane);

ip2: when the **kind** is **abutting** the **structured_donor** shall also describe a face subrange;

5.4.7 multiple_mesh_block

A **multiple_mesh_block** is a grouping of connected meshes. All mesh connectivity information pertaining to the group is contained in the **multiple_mesh_block** structure. This includes abutting interfaces (general mismatched and 1-to-1), overset-mesh interfaces, and overset-mesh holes.

All the interface patches for a given mesh in the group are contained in the **multiple_mesh_block** entity for that group. If a face of a mesh touches several other meshes (say N), the N different instances of the **mesh_connectivity** structure must be included in the **multiple_mesh_block** to describe each interface patch.

NOTE 1 This convention requires that a single interface patch be described twice — once for each adjacent mesh. It also means that the **multiple_mesh_block** is symmetrical with regard to interface patches.

EXPRESS specification:

```
*)
ENTITY multiple_mesh_block;
  name           : label;
  description    : text;
  id             : identifier;
  connectivities : LIST OF mesh_connectivity;
END_ENTITY;
(*
```

Attribute definitions:

name: user-specified instance identifier;

description: annotation;

id: an identifier;

connectivities: the connectivity information.

5.4.8 structured_donor_mesh

A **structured_donor_mesh** is a **mismatched_donor_mesh** that is structured.

EXPRESS specification:

```
*)
ENTITY structured_donor_mesh
  SUBTYPE OF (mismatched_donor_mesh);
  SELF\mismatched_donor_mesh.donor  : structured_mesh;
  points : listed_real_data;
  vsize  : INTEGER;
DERIVE
  index_count : INTEGER := donor.index_count;
END_ENTITY;
(*
```


Attribute definitions:

donor: the structured donor mesh;

points: the image of the receiver mesh interface points in the donor mesh. These may be thought of as bi- or tri-linear interpolants (depending on **dimension**) in the computational mesh of the donor mesh. FORTRAN multidimensional array ordering shall be used;

vsize: the size of the data array necessary to contain the interface points;

index_count: the number of indices required to reference a vertex.

5.4.9 unstructured_donor_mesh

An **unstructured_donor_mesh** is a **mismatched_donor_mesh** that is unstructured.

EXPRESS specification:

```
*)
ENTITY unstructured_donor_mesh
  SUBTYPE OF (mismatched_donor_mesh);
  SELF\mismatched_donor_mesh.donor  : unstructured_mesh;
  cells       : indices_group;
  interpolant : property_distribution_description;
  vsize       : INTEGER;
DERIVE
  index_count : INTEGER := donor.index_count;
END_ENTITY;
(*
```

Attribute definitions:

donor: the unstructured donor mesh;

cells: contains the donor cell where the node is located;

interpolant: contains the interpolation factors to locate the node in the donor cell;

vsize: the size of the data array necessary;

index_count: the number of indices required to reference a vertex;

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of mesh_connectivity_schema
(*
```

6 Mesh function

The following EXPRESS declaration begins the **mesh_function_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA mesh_function_schema;
  REFERENCE FROM mathematical_functions_schema       -- ISO 10303-50
    (application_defined_function,
     maths_function,
     function_is_table);
  REFERENCE FROM mesh_topology_schema                 -- ISO 10303-52
    (mesh);
  REFERENCE FROM ISO13584_generic_expressions_schema  -- ISO 13584-20
    (generic_expression,
     unary_generic_expression);
(*
```

NOTE The schemas referenced above can be found in the following parts of ISO 10303:

mathematical_constructs_schema	ISO 10303-50
mesh_topology_schema	Clause 4 of this part of ISO 10303
ISO13584_generic_expressions_schema	ISO 13584-20

6.1 General

This schema defines and describes the structure types for describing mathematical functions defined over meshes.

6.2 Fundamental concepts and assumptions

The mesh functions defined in this schema are interpolation functions defined over a mesh. The mesh may be either structured or unstructured and individual interpolation functions can be defined for each cell of the mesh.

6.3 mesh_function_schema entity definitions

6.3.1 mesh_function

A **mesh_function** is an **application_defined_function** that:

— has a domain that consists of a table of real tuple spaces;

The table is a one dimensional array for an unstructured mesh, but can be a rectangular array of any dimension for a structured mesh.

The real tuples can have different dimensions, if the mesh contains cells of different dimensions.

The real tuple spaces can have different bounds, if the mesh contains cells of different shape, such as wedges and hexhedra.

— has a set of control values, such that the function within each cell is determined by a subset of the control values and basis for that cell.

The assignment of control values to cells is determined by the topology of the mesh. A control value is usually, but not necessarily, the value of the function at a position within the mesh.

— interpolates or extrapolates separately within each cell from the control values assigned to that cell.

EXPRESS specification:

```
*)
ENTITY mesh_function
  SUBTYPE OF (application_defined_function,
              unary_generic_expression);
  mesh          : mesh;
  basis         : LIST OF mesh_function_basis;
  uniform       : BOOLEAN;
  vertex_values : BOOLEAN;
DERIVE
  control_values : maths_function := SELF\unary_generic_expression.operand;
WHERE
  WR1 : function_is_table(control_values);
  WR2 : (uniform AND (SIZEOF(basis) = 1)) XOR
        (NOT uniform);
END_ENTITY;
(*
```

Attribute definitions:

mesh: the **mesh** for the **mesh_function**;

basis: the **mesh_function_basis** specifying the methods of interpolation or extraction for use within each cell of the **mesh**;

uniform: a flag which indicates whether or not the the **mesh_function** has a uniform basis, as follows:

— if **uniform** is true, then each cell of the mesh has the same **mesh_function_basis**;

— if **uniform** is false, then a **mesh_function_basis** is specified separately for each cell of the **mesh**.

vertex_values: a flag that indicates whether or not the **control_values** are specified for the vertices or the cells of the **mesh**, as follows:

Table 5 – Domain of the control values table for a mesh_function

mesh	location	domain
unstructured	values at cell vertices	i where: — i is the position in the vertex array specified for the unstructured mesh.
unstructured	discretisation points for each cell	$j_1, j_2, \dots j_m, i$ where: — $j_1, j_2, \dots j_m$ is the position in the pattern of discretisation points, for the cell basis; — i is the position in the cell array specified for the unstructured mesh.
structured	values at vertices	$i_1, i_2, \dots i_n$ where: — $i_1, i_2, \dots i_n$ is the index of the vertex in the structured mesh.
structured	discretisation points for each cell	$j_1, j_2, \dots j_m, i_1, i_2, \dots i_n$ where: — $j_1, j_2, \dots j_m$ is the position in the pattern of discretisation points, for the cell basis; — $i_1, i_2, \dots i_n$ is the index of the vertex the structured mesh.

— if **vertex_values** is true, then the **control_values** are specified for the array of unique **vertex** for the cells of the **mesh**;

— if **vertex_values** is false, then the **control_values** are specified separately for the pattern of discretisation points for each cell of the **mesh**.

control_values: the table that specifies the control values at the vertices of the cells of the **mesh** or at the discretisation points of each cell.

The table shall be a function with:

domain: integer tuple space that identifies the vertices or discretisation points of the **mesh_function**;

range: that is the same as the range of the **mesh_function**.

NOTE 1 The range and domain of this function are the attributes **explicit_range** and **explicit_domain** inherited from the supertype **application_defined_function**.

The components of the integer tuple space domain are shown in Table 5.

Formal propositions:

WR1: the control values shall be a table;

WR2: if **uniform_basis** is true, then the length of the basis list shall be 1.

6.3.2 mesh_function_basis

A **mesh_function_basis** is an **application_defined_function** that has:

— a domain that is a real tuple space; and

— a range that is a space of table functions.

NOTE 1 A **mesh_function_basis** is used such that the domain is a parametric space that identifies points within a cell of a mesh. Hence the dimension of the real tuple space is the same as the topological dimension of the cell.

Each table function in the range of a **mesh_function_basis** has:

— a domain that is a subscript space; and

— a range that is the reals.

NOTE 2 A **mesh_function_basis** is used such that:

— the domain of each table function in its range is a parametric space that identifies the control values for a cell;

— the range of each table function is a weighting for a control value.

Each table function specifies the weightings for control values that define the value of a **mesh_function** at a point in a cell.

NOTE 3 An Application Module can define a subtype of **mesh_function_basis** and **externally_defined_item** to record a standard instance of a function such as the 'serendipity' function commonly used in finite element analysis.

EXPRESS specification:

```
*)
ENTITY mesh_function_basis
  SUBTYPE OF (application_defined_function,
              unary_generic_expression);
  cell_topological_dimension : INTEGER;
  value_array_dimension      : INTEGER;
  value_array_order          : ARRAY [1:value_array_dimension] OF INTEGER;
DERIVE
  value_positions : maths_function := SELF\unary_generic_expression.operand;
```

```
WHERE
  value_positions_as_table : function_is_table(value_positions);
END_ENTITY;
(*
```

Attribute definitions:

cell_topological_dimension: the dimension of the real tuple space that is the domain of the **mesh_function_basis**;

value_array_dimension: the dimension of the subscript space for each table function in the range of the **mesh_function_basis**;

value_array_order: the number of control values less one, for each value array direction;

value_positions: the table that specifies the 'positions' of the control values within the domain of the **mesh_function_basis**. The 'position' of a control value is the point within the domain that has a weighting of 1.0 for that control value and a weighting of 0.0 for all other control values.

The **value_positions** table shall be a function that has:

— a domain that is an integer tuple space that identifies discretisation points for a cell; and

— a range that is a real tuple space that identifies positions in a cell.

A control value for a **mesh_function_basis** need not have a position. Whether or not a control value has a position is determined by the **mesh_function_basis**. If a control value does not have a position then the corresponding value of the **value_positions** function is of no signficance.

Formal propositions:

value_positions_as_table: the **value_positions** shall be a table.

Informal propositions:

consistent_topology_dimension: the **cell_topological_dimension** shall be the same as the **topological_dimension** of any cell to which the **mesh_function_basis** is assigned.

consistent_value_position_table_domain: the domain of the table of control value positions shall be consistent with the **value_array_order**. It shall be the tuple space that is the Cartesian product of integer intervals $[1, n_1] \times [1, n_2] \times \cdots [1, n_m]$, where:

— m is the **value_array_dimension**; and

— n_i is **value_array_order**[i] + 1.

consistent_value_position_table_range: the dimension of the tuple space that is the range of the table of control positions shall be equal to the **cell_topological_dimension**.

valid_weighting_at_value_position: if a control value is assigned to a position, then the value of the **mesh_function_basis** at that position shall be:

— 1 for the control value assigned to the position;

— 0 for all other control values.

6.4 mesh_function_schema subtype constraint definitions

6.4.1 sc1_application_defined_function

There is a ONEOF relationship between the **mesh_function** and **mesh_function_basis** subtypes of **application_defined_function**.

EXPRESS specification:

```
*)
SUBTYPE_CONSTRAINT sc1_application_defined_function FOR
                   application_defined_function;
  ONEOF(mesh_function,
        mesh_function_basis);
END_SUBTYPE_CONSTRAINT;
(*
```

6.4.2 sc1_unary_generic_expression

There is a ONEOF relationship between the **mesh_function** and **mesh_function_basis** subtypes of **unary_generic_expression**.

EXPRESS specification:

```
*)
SUBTYPE_CONSTRAINT sc1_unary_generic_expression FOR
                   unary_generic_expression;
  ONEOF(mesh_function, mesh_function_basis);
END_SUBTYPE_CONSTRAINT;
(*
```

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of mesh_function_schema
(*
```

Annex A
(normative)

Short names of entities

Table A.1 provides the short names of entities specified in this part of ISO 10303. Requirements on the use of short names are found in the implementation methods included in ISO 10303.

NOTE 1 The short names are available from the Internet — see annex C.

Table A.1 – Short names of entities

Entity data types names	Short names
array_based_unstructured_mesh	ABUM
array_based_unstructured_mesh_and_vertices	ABUMAV
cell	CELL
cell_of_structured_mesh	COSM
cell_with_explicit_boundary	CWEB
explicit_unstructured_mesh	EXUNMS
extraction_of_structured_submesh	EOSS
extraction_of_submesh	EXOFSB
extraction_of_submesh_by_cells	EOSBC
extraction_of_submesh_by_vertices	EOSBV
indices_list	INDLST
indices_range	INDRNG
matched_mesh_connection	MTMSCN
mesh	MESH
mesh_connectivity	MSHCNN
mesh_derived_maths_space	MDMS
mesh_function	MSHFNC
mesh_function_basis	MSFNBS
mesh_overset_hole	MSOVHL
mismatched_donor_mesh	MSDNMS
mismatched_mesh_connection	MSMSCN
mismatched_mesh_region	MSMSRG
multiple_mesh_block	MLMSBL
product_of_mesh	PROFMS
rind	RIND
structured_donor_mesh	STDNMS
structured_mesh	STRMSH
structured_mesh_with_rind	SMWR
submesh	SBMSH
unstructured_donor_mesh	UNDNMS
unstructured_mesh	UNSMSH
vertex_defined_cell	VRDFCL

Annex B
(normative)

Information object registration

B.1 Document identification

To provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part(52) version(-1) }

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

B.2 Schema identification

To provide for unambiguous identification of the **mesh_topology_schema** in an open information system, the object identifier

{ iso standard 10303 part(52) version(1) schema(1) mesh-topology-schema(1) }

is assigned to the **mesh_topology_schema** schema (see 4). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

To provide for unambiguous identification of the **mesh_connectivity_schema** in an open information system, the object identifier

{ iso standard 10303 part(52) version(1) schema(1) mesh-connectivity-schema(1) }

is assigned to the **mesh_connectivity_schema** schema (see 5). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

To provide for unambiguous identification of the **mesh_function_schema** in an open information system, the object identifier

{ iso standard 10303 part(52) version(1) schema(1) mesh-function-schema(1) }

is assigned to the **mesh_function_schema** schema (see 6). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex C
(informative)

Computer-interpretable listings

This annex provides a listing of the EXPRESS entity names and corresponding short names as specified in this Part of ISO 10303 without comments or other explanatory text. This annex is aavailable in computer-interpretable form and can be found at the following URLs:

Short names: <http://www.tc184-sc4.org/Short_Names/>

EXPRESS: <http://www.tc184-sc4.org/EXPRESS/>

Annex D
(informative)

EXPRESS-G diagrams

Figures D.1 through D.10 correspond to the EXPRESS given in annex C. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in ISO 10303-11.

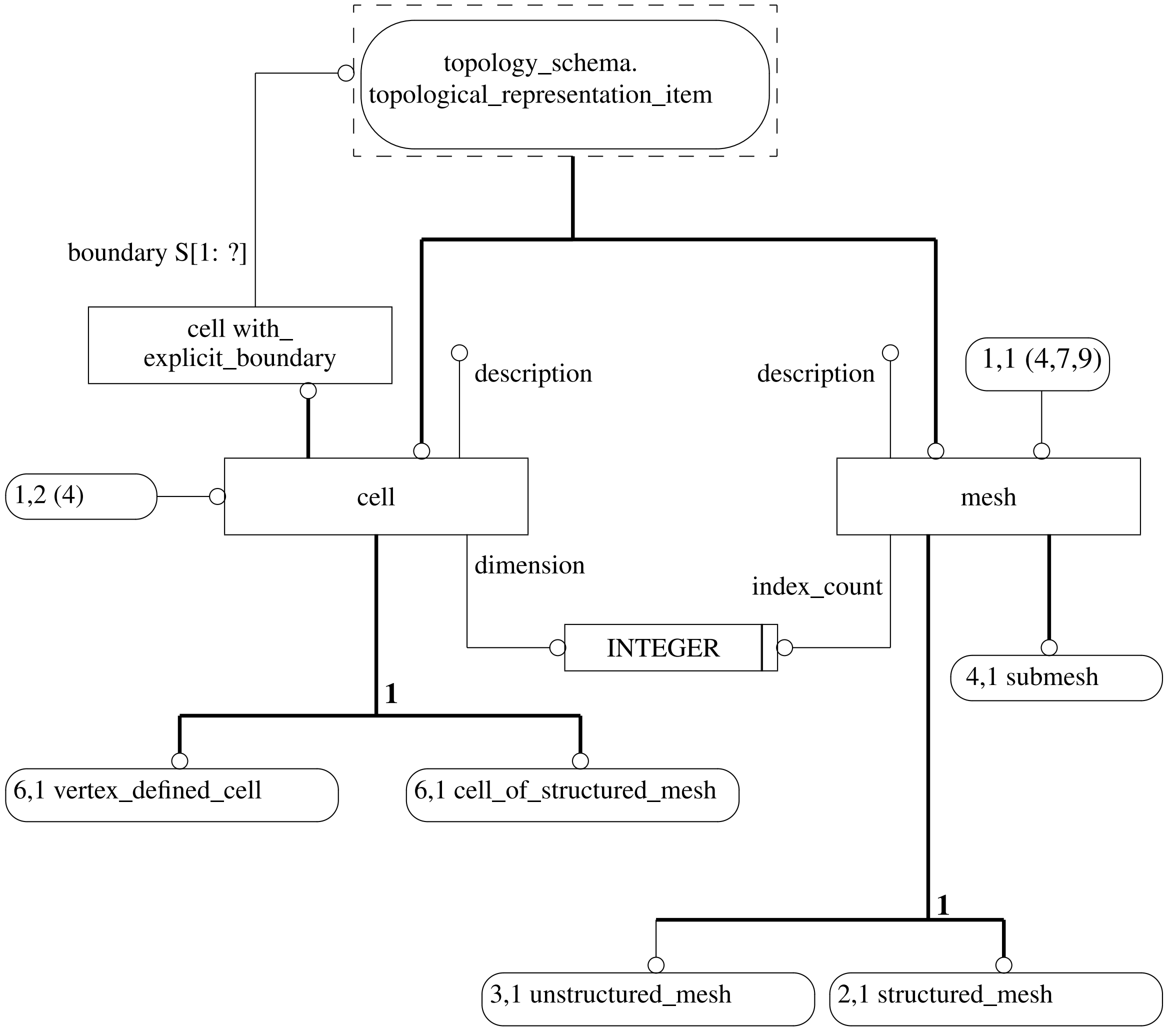


Figure D.1 – Entity level diagram of mesh_topology_schema schema (page 1 of 10)

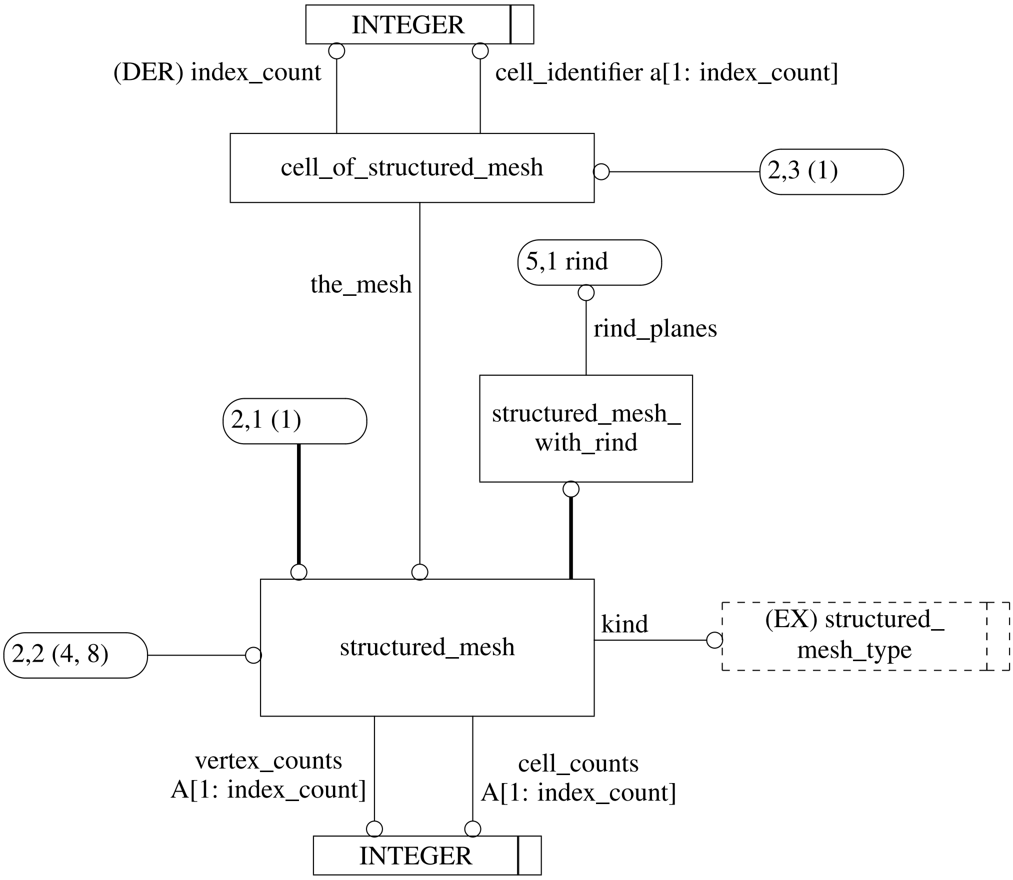


Figure D.2 – Entity level diagram of mesh_topology_schema schema (page 2 of 10)

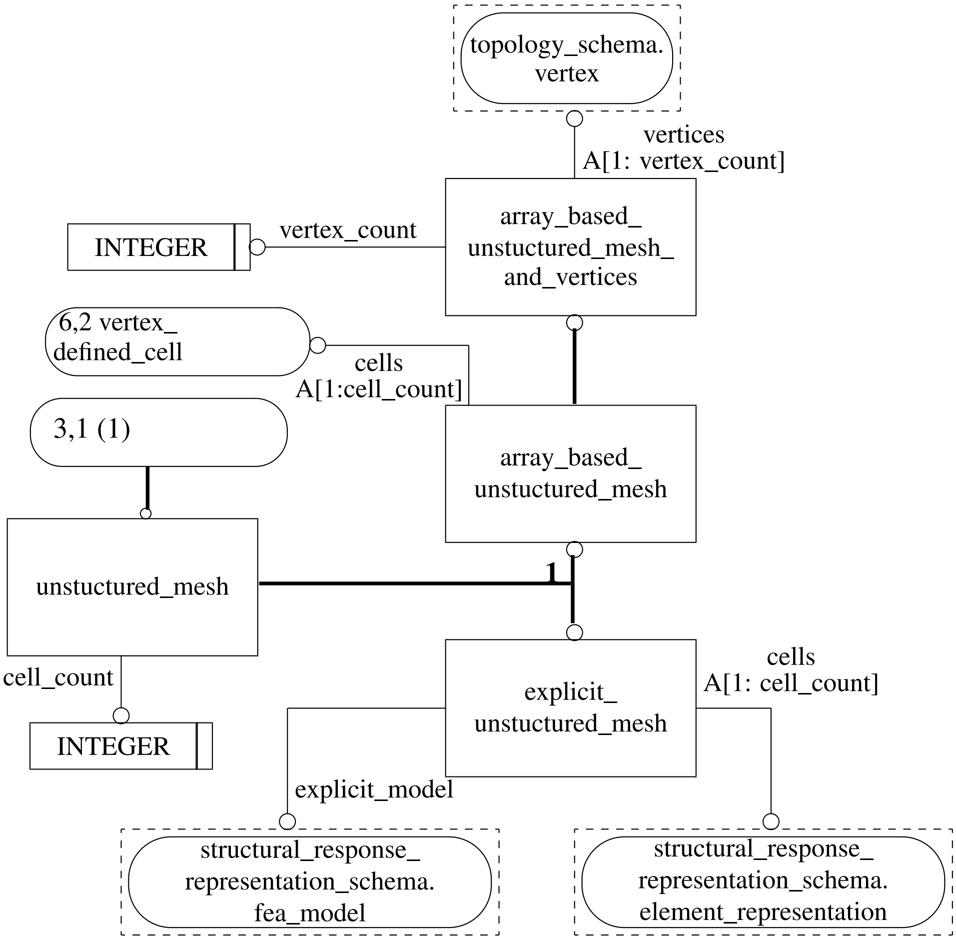


Figure D.3 – Entity level diagram of mesh_topology_schema schema (page 3 of 10)

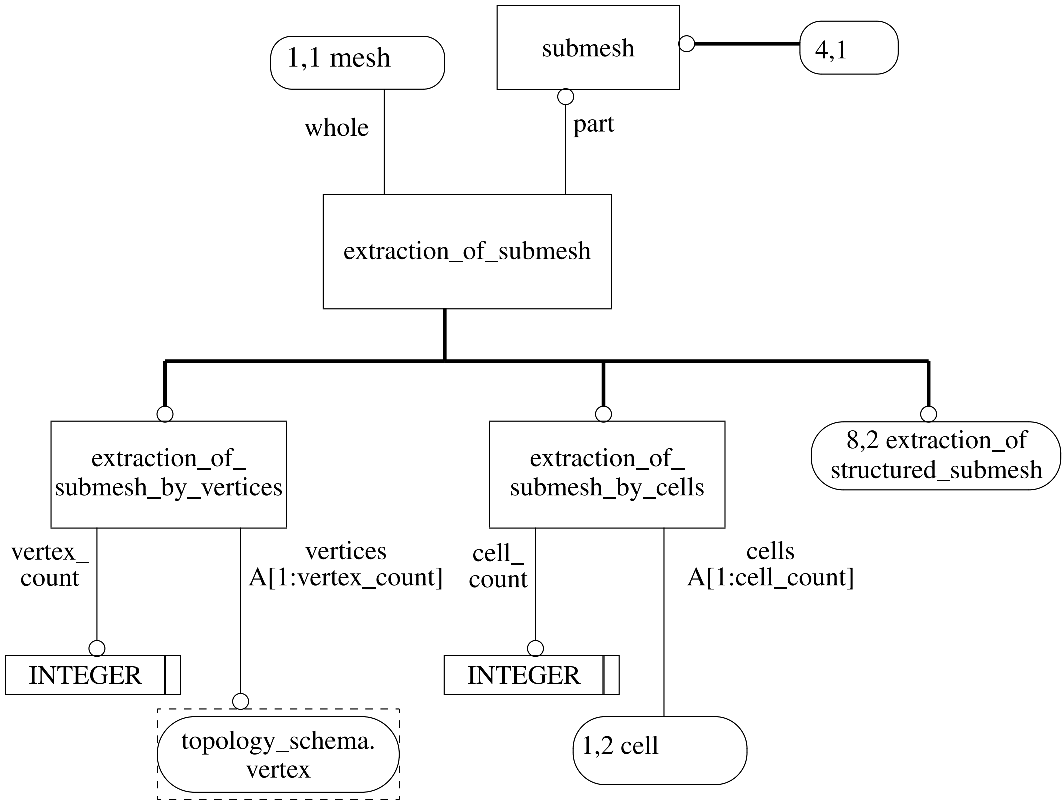


Figure D.4 – Entity level diagram of mesh_topology_schema schema (page 4 of 10)

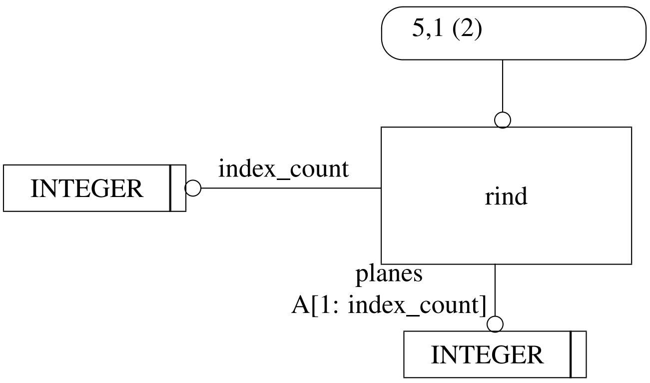


Figure D.5 – Entity level diagram of mesh_topology_schema schema (page 5 of 10)

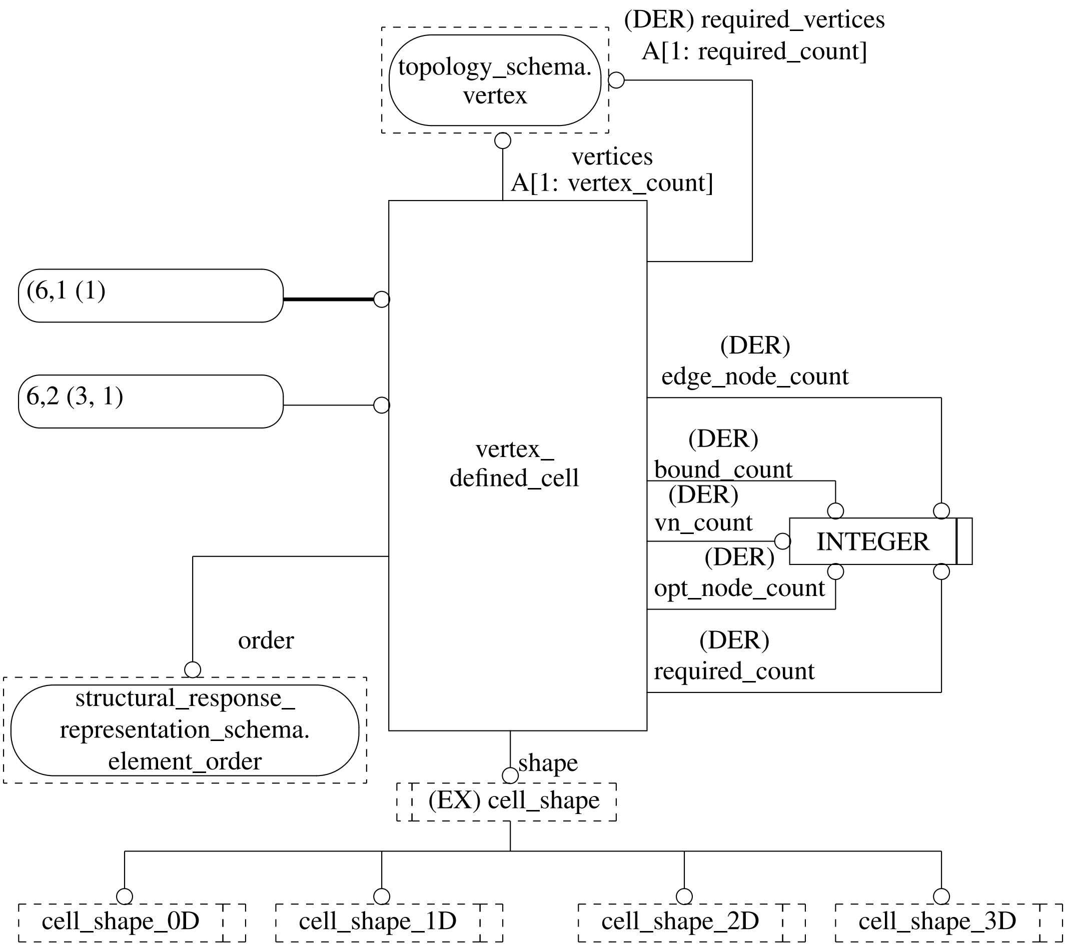


Figure D.6 – Entity level diagram of mesh_topology_schema schema (page 6 of 10)

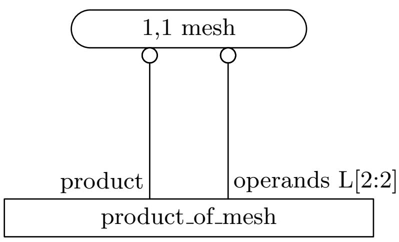


Figure D.7 – Entity level diagram of mesh_topology_schema schema (page 7 of 10)

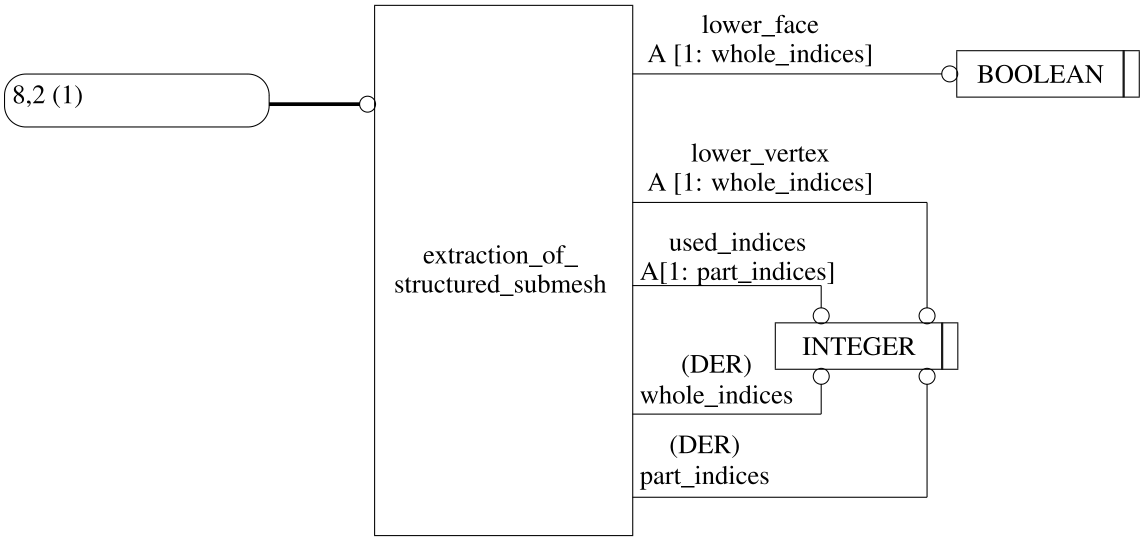


Figure D.8 – Entity level diagram of mesh_topology_schema schema (page 8 of 10)

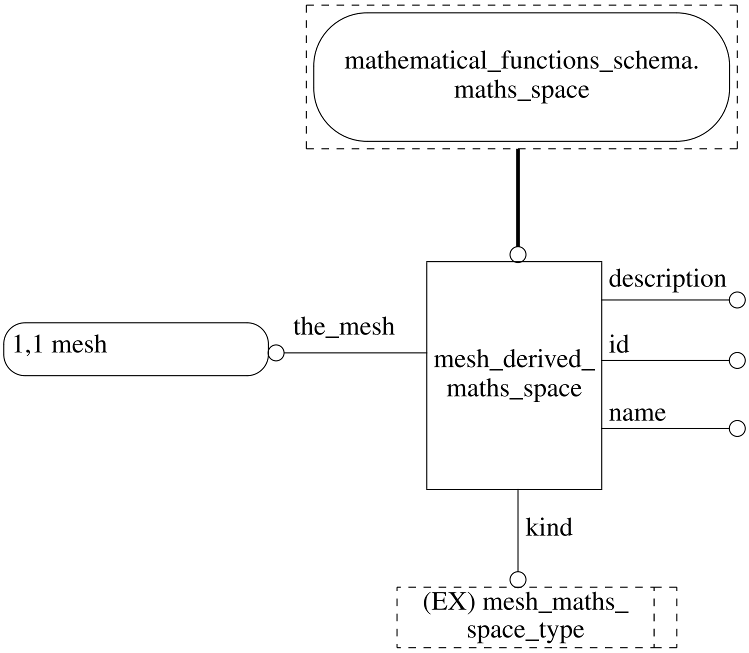


Figure D.9 – Entity level diagram of mesh_topology_schema schema (page 9 of 10)

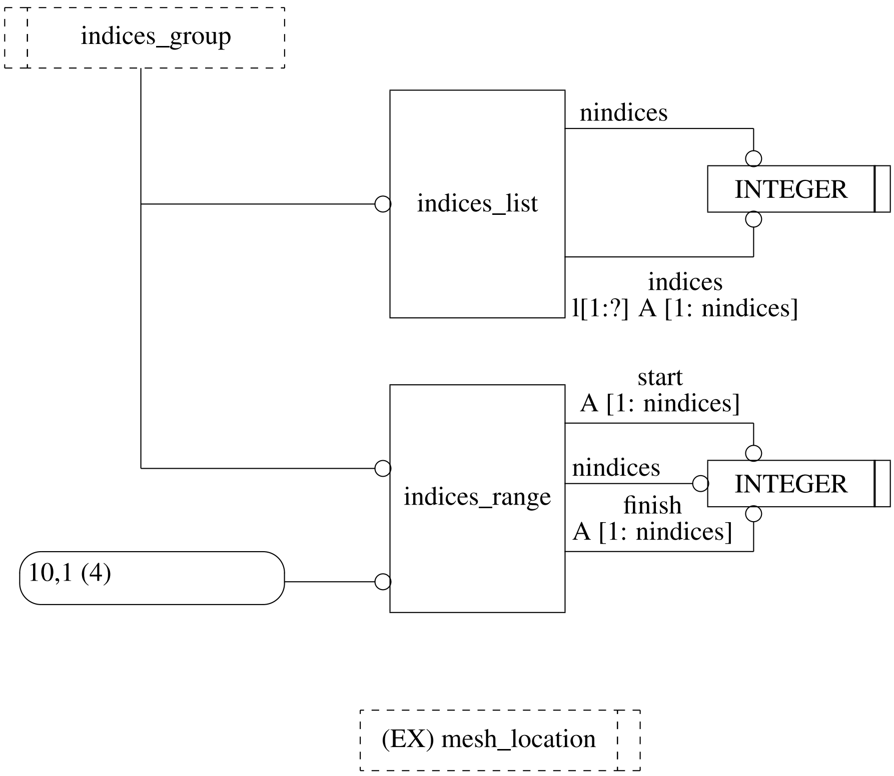


Figure D.10 – Entity level diagram of mesh_topology_schema schema (page 10 of 10)

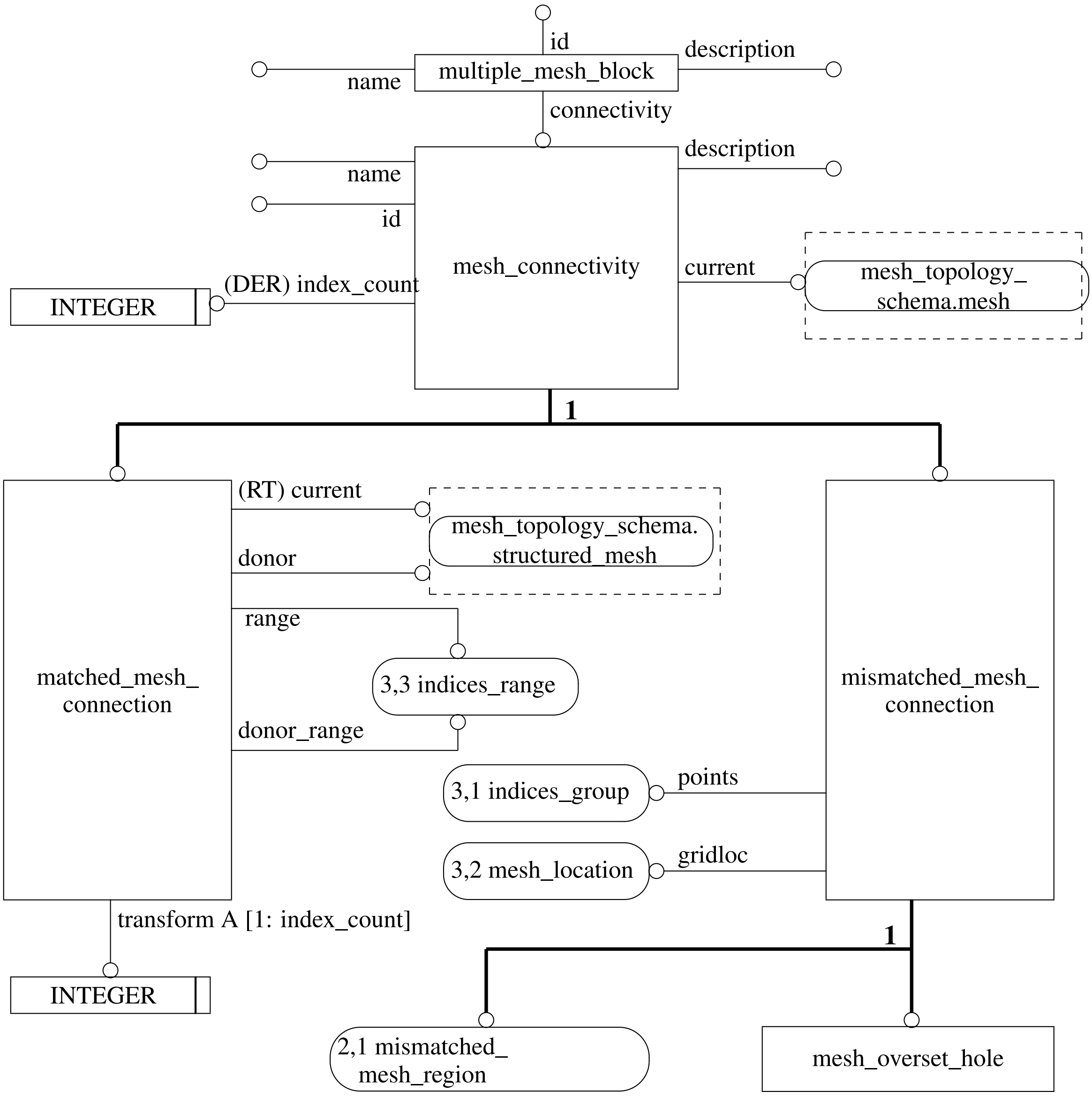


Figure D.11 – Entity level diagram of mesh_connectivity_schema schema (page 1 of 3)

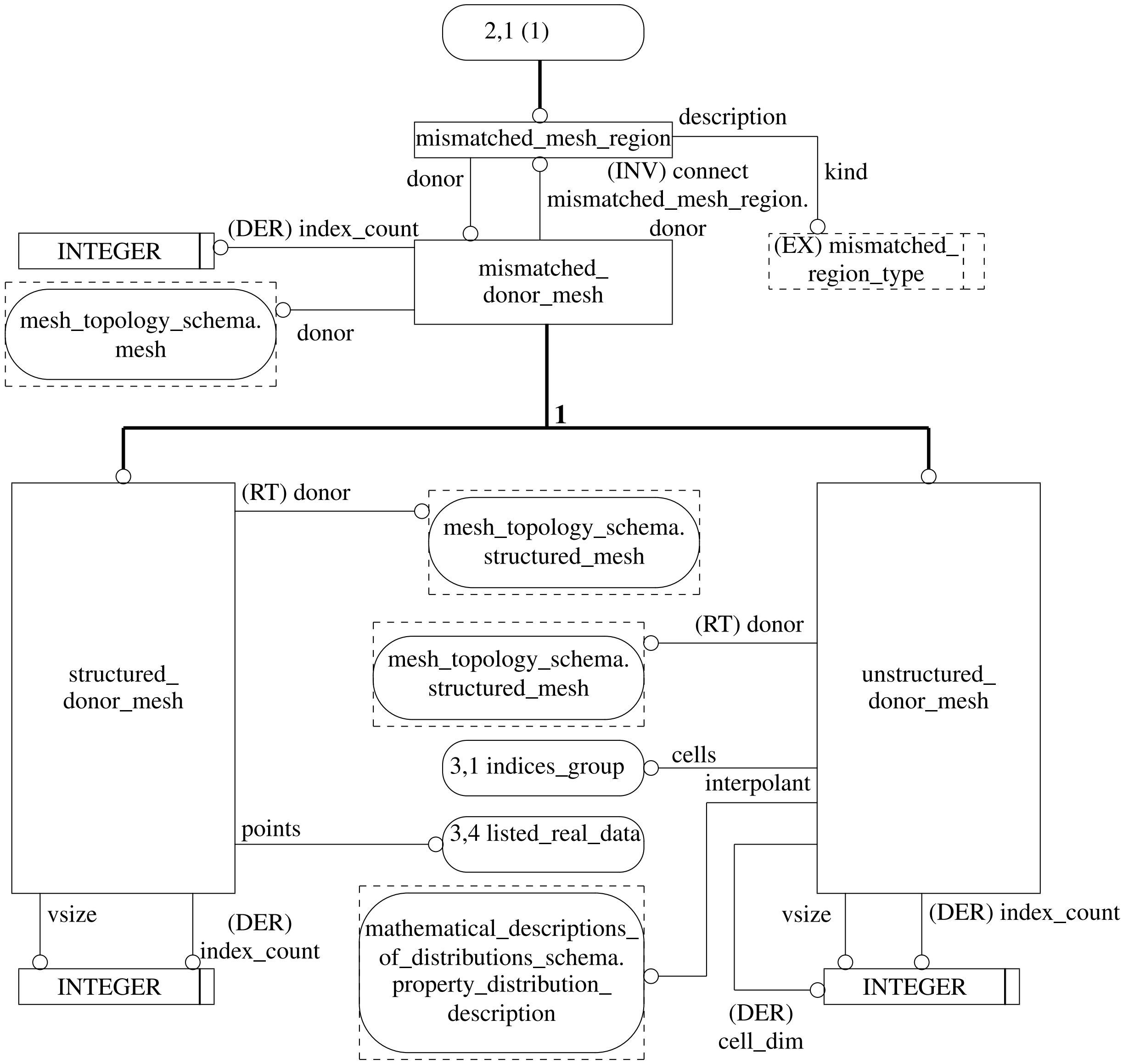


Figure D.12 – Entity level diagram of mesh_connectivity_schema schema (page 2 of 3)

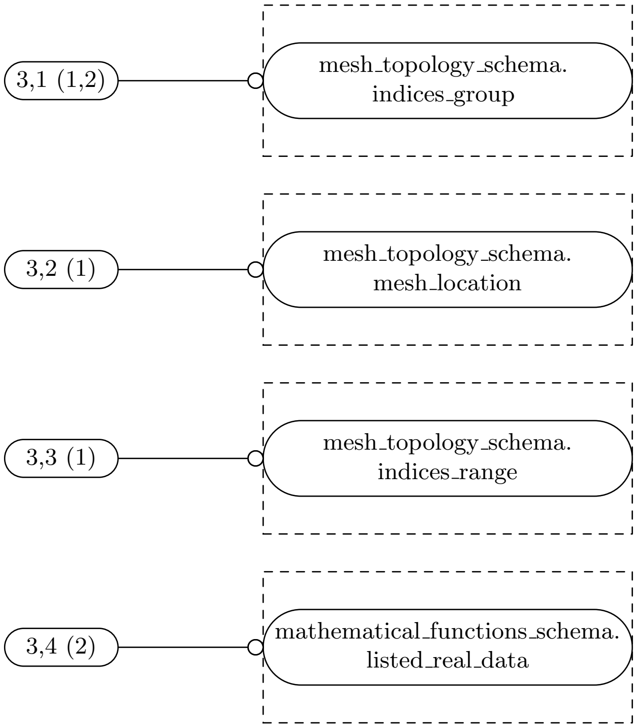


Figure D.13 – Entity level diagram of mesh_connectivity_schema schema (page 3 of 3)

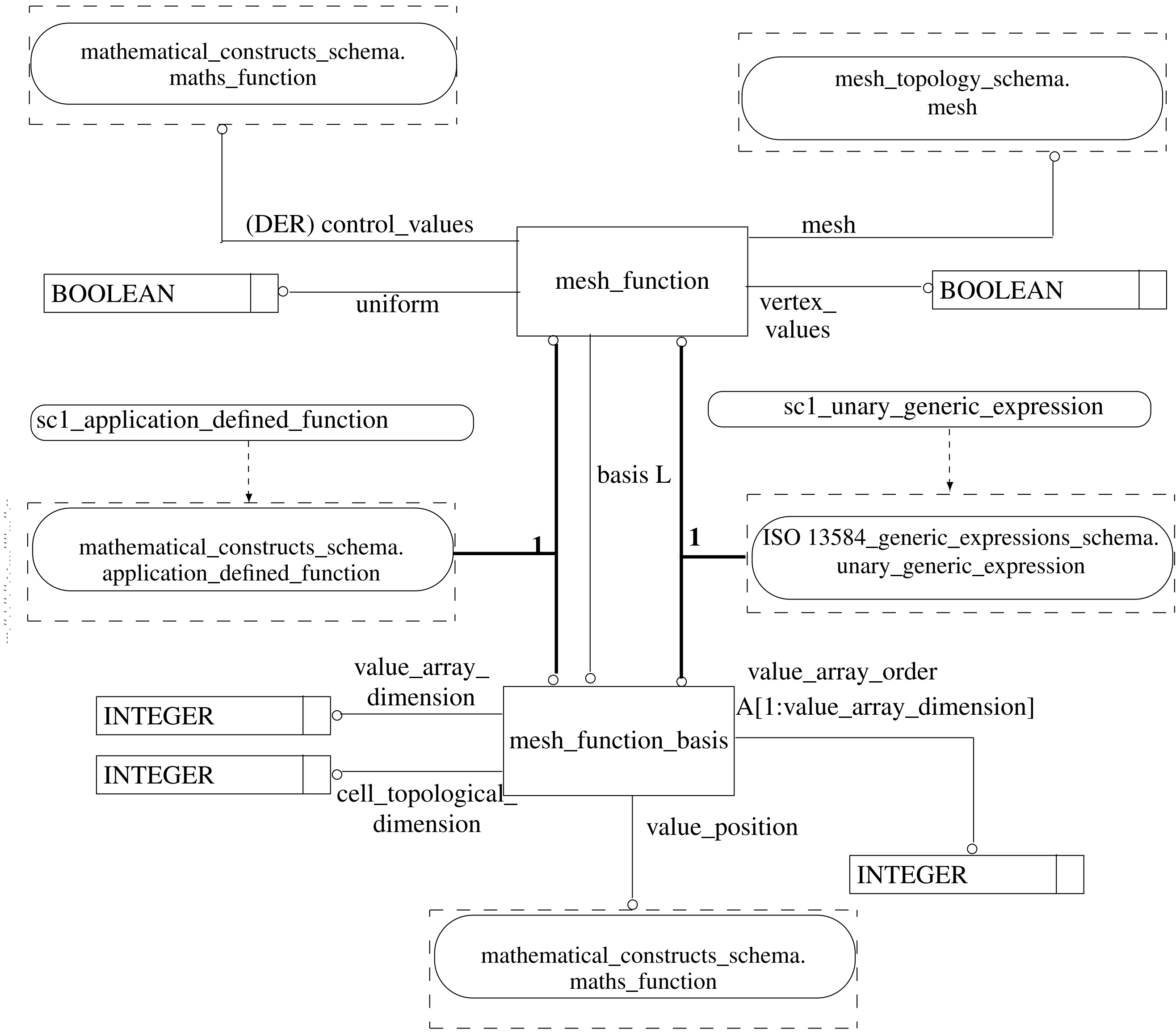


Figure D.14 – Entity level diagram of mesh_function_schema schema (page 1 of 1)

Annex E
(informative)

Additional information

Table E.1 lists the elements of the **mesh_topology_schema** that are used by other schemas.

Table E.1 – Elements of mesh_topology_schema used by other schemas

Element	Used in schema	By element
indices_group	**mesh_connectivity_schema**	**mismatched_mesh_connection** **unstructured_donor_mesh**
indices_range	**mesh_connectivity_schema**	**matched_mesh_connection**
mesh	**mesh_connectivity_schema** **mesh_function_schema**	**mesh_connectivity** **mesh_function**
mesh_location	**mesh_connectivity_schema**	**mismatched_mesh_connection**
structured_mesh	**mesh_connectivity_schema**	**matched_mesh_connection** **structured_donor_mesh**
unstructured_mesh	**mesh_connectivity_schema**	**unstructured_donor_mesh**

Bibliography

[1] ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1) – Part 1: Specification of basic notation.*

Index

all_mesh_vertices (function), ... 35
application protocol (AP), ... 2
array_based_unstructured_mesh (entity), ... 14

cell , ... 3
cell (entity), ... 15
cell edge , ... 3
cell face, ... 3
cell vertex, ... 3
cell_counts (function), ... 36
cell_of_structured_mesh (entity), ... 16
cell_shape (type), ... 6
cell_shape_0D (type), ... 7
cell_shape_1D (type), ... 7
cell_shape_2D (type), ... 7
cell_shape_3D (type), ... 8
cell_with_explicit_boundary (entity), ... 15
CFD, ... 3

explicit_unstructured_mesh (entity), ... 16
extraction_of_structured_submesh (entity), ... 17
extraction_of_submesh (entity), ... 18
extraction_of_submesh_by_cells (entity), ... 19
extraction_of_submesh_by_vertices (entity), ... 19

indices_group (type), ... 8
indices_list (entity), ... 20
indices_range (entity), ... 20
integrated resource, ... 2

matched_mesh_connection (entity), ... 42
mesh, ... 3
mesh (entity), ... 21
mesh_connectivity (entity), ... 44
mesh_connectivity_schema (schema), ... 39
mesh_derived_maths_space (entity), ... 21
mesh_function (entity), ... 51
mesh_function_basis (entity), ... 53
mesh_function_schema (schema), ... 50
mesh_location (type), ... 9
mesh_maths_space_type (type), ... 10
mesh_overset_hole (entity), ... 45
mesh_topology_schema (schema), ... 4
mismatched_donor_mesh (entity), ... 45
mismatched_mesh_connection (entity), ... 46

mismatched_mesh_region (entity), 47
mismatched_region_type (type), 42
multiple_mesh_block (entity), 48

product, 2
product_of_mesh (entity), 22

rind, 2
rind (entity), 23

sc1_application_defined_function (subtype constraint), 55
sc1_mismatched_donor_mesh (subtype constraint), 45
sc1_mismatched_mesh_connection (subtype constraint), 46
sc1_unary_generic_expression (subtype constraint), 55
shorten_array (function), 38
structured_donor_mesh (entity), 48
structured_mesh (entity), 24
structured_mesh_type (type), 10
structured_mesh_with_rind (entity), 25
submesh (entity), 26

this_schema (function), 39
topological region, 3

unstructured_donor_mesh (entity), 49
unstructured_mesh (entity), 26
URL, 3

vertex, 3
vertex_defined_cell (entity), 34

ICS 25.040.40

Price based on 76 pages