

Managing Mesh-Based Data in a Semantic Database

Authors: Juha Kortelainen, Paul Klinge, Kai Katajamäki

Confidentiality: Public

Report's title Managing Mesh-Based Data in a Semantic Database		
Customer, contact person, address Tekes, Digital product process technology programme P.O. Box 69 FI-00101 Helsinki		Order reference 3088/31/08
Project name Computational models in product life cycle		Project number/Short name 31224/Codes
Author(s) Juha Kortelainen, Paul Klinge, Kai Katajamäki		Pages 30/–
Keywords Modelling, simulation, mesh, CFD, FEM, semantic, format		Report identification code VTT-R-03154-11
<p>Summary</p> <p>Several numerical methods, such as finite difference method, finite element method, and control volume method, use spatial discretisation for estimating spatial differentials. Common to these methods is that they estimate continuous volume with discrete finite volumes or nodes. Use of semantic data model is an attempt to unify the representation of modelling data in computational systems.</p> <p>The objective of this work is to create a generic and comprehensive description of spatially discretised geometric model usually called a mesh. The semantic representation should be applicable to different numerical computational methods and modelling domains. The fundamental objective is to enable the use of different tools like finite element analysis and computational fluid dynamics tools to access mesh data from a common semantic database. The emphasis on this representation development is on minimising the amount of data that is needed to store explicitly for a computational mesh and all data that is needed to run a computation.</p> <p>This study shows that spatially discretised data can be represented in semantic form and that there are several approaches for representing the data. The fully semantic approach provides the most flexible data model from semantic point of view by allowing semantic reasoning in single element level, due to semantic representation of each of the nodes and elements of the mesh. On the other hand, the fully semantic approach is also the most resource intensive and is probably too inefficient for industrial applications. An approach between fully semantic data representation and plain tabular data was selected as the best compromise for the data model.</p>		
Confidentiality	Public	
Espoo 9.5.2011		
Written by	Reviewed by	Accepted by
Juha Kortelainen, Senior Research Scientist	Juha Virtanen Senior Research Scientist	Pekka Koskinen, Technology Manager
VTT's contact address VTT Technical Research Centre of Finland, P.O. Box 1000, FI-02044 VTT, Finland		
Distribution (customer and VTT) Tekes: 1 piece VTT: 1 piece		
<p><i>The use of the name of the VTT Technical Research Centre of Finland (VTT) in advertising or publication in part of this report is only permissible with written authorisation from the VTT Technical Research Centre of Finland.</i></p>		

Contents

1	Introduction.....	3
1.1	Objectives	3
1.2	Used conventions and terminology	4
2	Introduction to computational mesh data	5
2.1	Structured mesh.....	5
2.2	Unstructured mesh.....	7
2.3	Boundary and initial conditions	8
2.4	Stored data and data types	8
3	Existing definitions for mesh data	10
3.1	CFD Generic Notation System CGNS	10
3.2	OpenFOAM internal mesh format	12
3.3	Universal File Format.....	13
4	General semantic data model for spatially discretised data.....	15
4.1	Possible strategies.....	15
4.1.1	A generic semantic mesh representation.....	15
4.1.2	Semantic native mesh representations	16
4.1.3	Semantic hybrid mesh representation.....	16
4.1.4	Fully semantic mesh representation	16
4.1.5	Meta-level semantic mesh representation.....	18
4.2	Problem bounding.....	18
4.3	Proposal for a generic semantic mesh data representation for CFD.....	18
4.3.1	Unstructured mesh.....	19
4.3.2	Structured mesh.....	20
4.3.3	Boundary and initial conditions	20
4.3.4	Results data representation	20
4.3.5	Semantic mesh data model.....	20
4.4	General semantic data model for FEM models	22
4.4.1	Description of general semantic data model for FEM meshes	23
5	Summary	27
	References	29

1 Introduction

Several numerical methods, such as finite difference method (FDM), finite element method (FEM), and control volume method (CVM), use spatial discretisation for estimating spatial differentials. Common to these methods is that they estimate continuous volume with discrete finite volumes (FEM and CVM) or nodes (FDM). Use of semantic data model is an attempt to unify the representation of modelling data in computational systems. The advantage of semantic data model is that it allows flexible description of new data types and relations between different kinds of data in the same data model.

This report is part of a Tekes (the Finnish Funding Agency for Technology and Innovation) funded research project *Computational models in product life cycle – Codes*. The research project focuses on two major areas: 1) integration of modelling and simulation seamlessly into product development process, and 2) applying modelling and simulation to the whole product life cycle so that also other aspects than product technical aspects, like business and environment, are taken into account. The backbone of the Codes project is semantic data model and its application to the projects focus areas.

According to the author's knowledge, there does not exist a general method to semantically describe spatially discretised models like structural analysis and computational fluid dynamics models. In this report, some existing definitions for spatially discretised model representations are described and a generic model for spatially discretised data model is proposed.

A semantic modelling and simulation environment, *Simantics* [1], has been developed at VTT to enable efficient use of semantic data model in numerical modelling and simulation. *Simantics* is a software platform that consists of a semantic database server, *Simantics Core*, and a client application, *Simantics Workbench*, for interactive modelling and database management. The platform does not contain solvers, but external third party components are integrated into the platform as plug-ins. The architecture contains also an internal data interface, *Simantics Databoard*, which provides mechanisms to define new data types and connect new data sources. In addition, there is an external data interface implementation, *Simantics OPC-UA*, utilising OPC Unified Architecture specification for communication with external data sources. The architecture of the platform is illustrated in Figure 1.

1.1 Objectives

The objective of this work is to create a generic and comprehensive description of spatially discretised geometric model usually called a mesh. The semantic representation should be applicable to different numerical computational methods and modelling domains. The fundamental objective is to enable the use of different tools like finite element analysis (FEA) and computational fluid dynamics (CFD) tools to access mesh data from a common semantic database. The emphasis on this representation development is on minimising the amount of data that is needed to store explicitly for a computational mesh and all data that is needed to run a computation. A mesh for some specific computational tool, like a flow solver, is generated from this model.

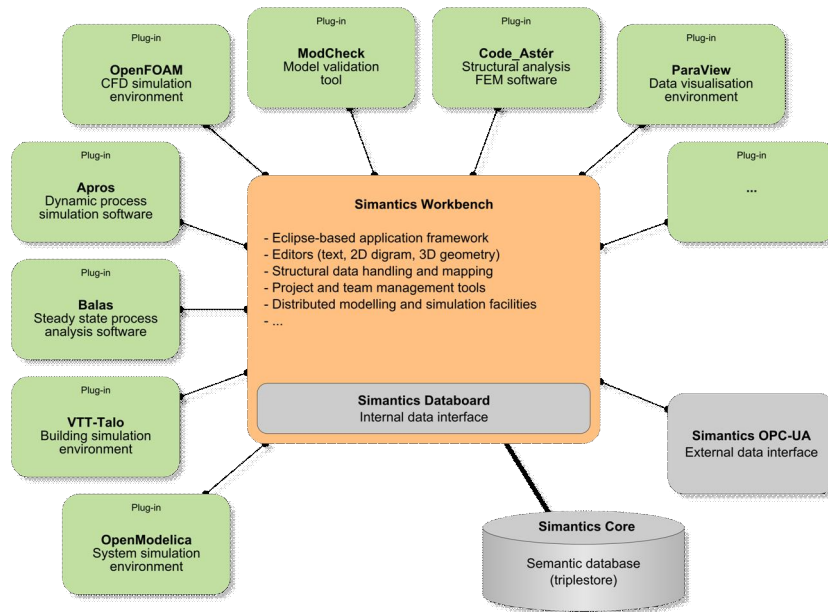


Figure 1: Simantics software architecture.

1.2 Used conventions and terminology

Spatial discretisation is usually referred as mesh or grid depending on the context. In this document, any spatial discretisation is referred as *mesh* for simplicity. In addition, individual discretisation volumes have different name depending on the context and the numerical method. In the finite element method (FEM), like the term already hints, these volumes are called *elements*. The elements are defined by *nodes* e.g. points at the vertexes or on the edges. In the control volume method (CVM), these volumes are called either *control volumes* or *cells*. For clarity, in this document *cells* are used when talking about individual discretisation volumes. Cells are constructed of *faces* that close the cell volume; faces do have *face normals*. A face is constructed of *points*, that define the corner points of the face, and *edges*, that connect the points and bound a face area. These terms are illustrated in Figure 2.

The following acronyms are used in the document:

2-D	Two dimensional.
3-D	Three dimensional.
AAIA	American Institute of Aeronautics and Astronautics.
ADF	Advanced data format.
BEM	Boundary element method.
CAD	Computer-aided design.
CAE	Computer-aided engineering.
CDF	Common data format by National Aeronautics and Space Administration.
CFD	Computational fluid dynamics.
CGNS	CFD general notation system.
CVM	Control volume method.
FDM	Finite difference method.
FEA	Finite element analysis.
FEM	Finite element method.

HDF	Hierarchical data format. HDF5 is the version 5 of the format.
I/O	Input/output.
NASA	National Aeronautics and Space Administration.
NetCDF	Network common data form by University Corporation for Atmospheric Research.
OWL	Web Ontology Language.
SDRC	Structural Dynamics Research Corporation.
SIDS	Standard interface data structures; definition of CGNS file content.
UCAR	University Corporation for Atmospheric Research.

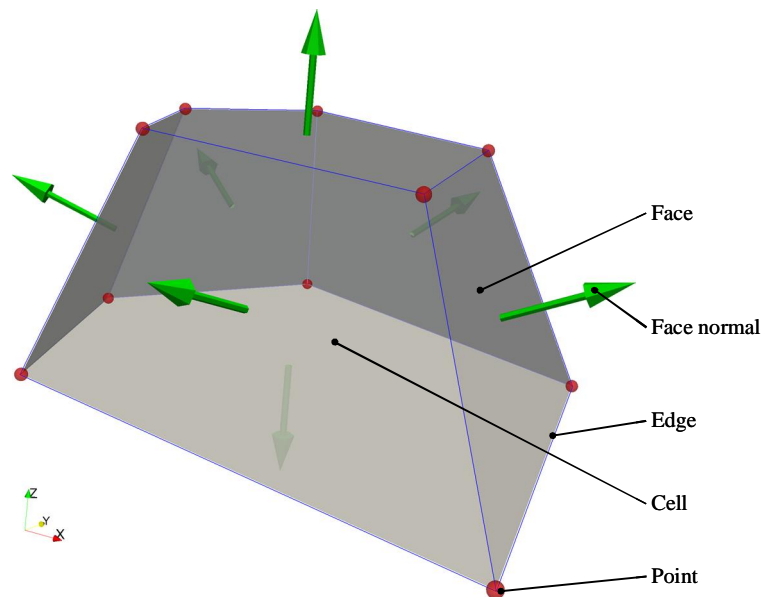


Figure 2: Mesh terminology used in this report.

2 Introduction to computational mesh data

The minimum information needed to describe an irregular and unstructured mesh is to describe the cell point coordinates and describe how cells are defined using the information of points. Also often, the information of neighbour cells is needed.

Some numerical methods need additional cells outside the actual computation domain for solving. This can be due to using differencing schemes that compute values based on neighbouring cells. However, at the boundary of a domain there are no cells on one side. In these cases so-called ghost cells are used. Ghost cells are additional cells outside the actual problem domain volume that fulfil the requirements for both the numerical scheme and the required boundary condition. The possible need for describing the ghost cells in the mesh representation should be taken into account when designing the data model for the mesh.

2.1 Structured mesh

A structured mesh usually means a computational mesh that has a topological structure. In a topological structure, the cells of the mesh can be indexed using curvilinear coordinates. E.g. in 3-dimensional model, there are cell indexes I , J , and K in x , y , and z directions respectively. With this representation, it is simple to refer to any neighbour of a cell with indexes:

- Direction x^- : $i-1$
- Direction x^+ : $i+1$
- Direction y^- : $j-1$
- Direction y^+ : $j+1$
- Direction z^- : $k-1$
- Direction z^+ : $k+1$

In Figure 3 two simple examples of structured meshes are presented, the first having orthogonal shape and the second having warped shape.

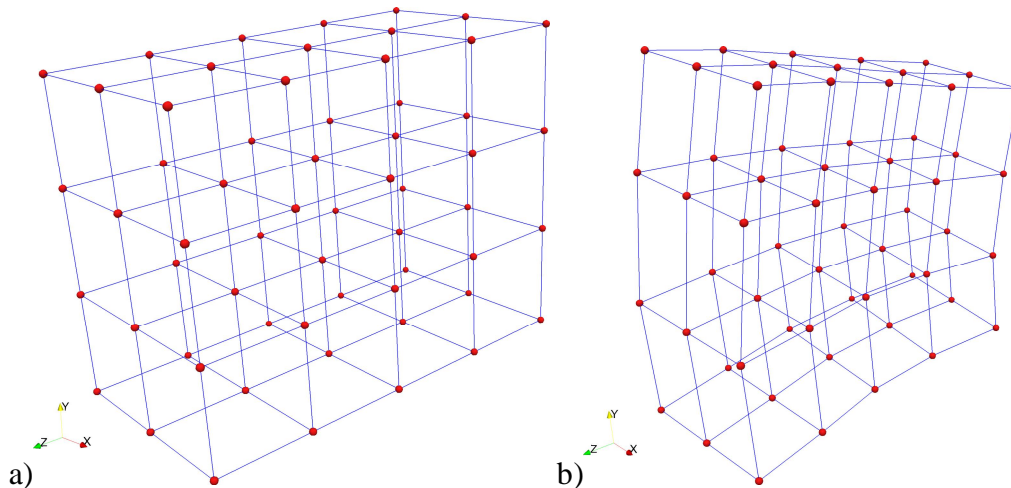


Figure 3: Examples of a) a mesh with regular positions and connections and b) a mesh with irregular positions and regular connections.

The minimal information for orthogonal equidistant mesh in 3-D is the domain dimensions in all coordinate directions and number of cells in each direction. That means a mesh like this in 3-D can be represented with two three-component vectors, one for dimensions and one for cell counts. Similar simplified representation can be applied to all analytical geometric shapes like cubes and cylinders. Only parameters that uniquely define the shape together with the cell counts in all characteristic directions are needed. The advantage of a representation like this is that it is almost insensitive to the size of the mesh and thus can very efficiently represent huge meshes. The big disadvantage is that these parametric mesh representations are limited to well-formed, relatively simple geometries, which are usually too simple typical engineering tasks.

To represent an arbitrary structured mesh usually requires the mesh points to be defined explicitly. This means all the coordinate values for each point have to be defined. For a large mesh, this means large value arrays to be managed. A structured computation mesh for a complex geometry is either very difficult or practically impossible to represent using just one computational mesh. Representing the computational mesh for an apparently well-formed geometry as shown in Figure 4 may be very difficult. To ease the creation of a structured mesh for complex geometry a multi-block representation can be used. Then the complex mesh is assembled from simpler sub-meshes that may or may not have conforming interfaces. A conforming interface means that the points in the mesh interfaces coincident so that the mesh continues seamlessly from one sub-mesh to another. The third option is to interpolate the data in overlapping areas of the sub-meshes between each other. In Figure 5 the different sub-mesh interfacing methods are illustrated in 2-D.

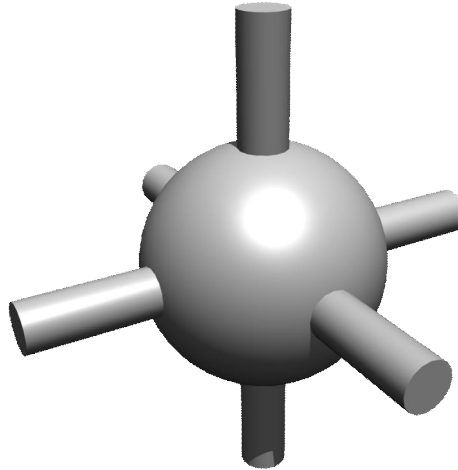


Figure 4: An example of a relatively complex geometry, which is very difficult to mesh using one structured mesh.

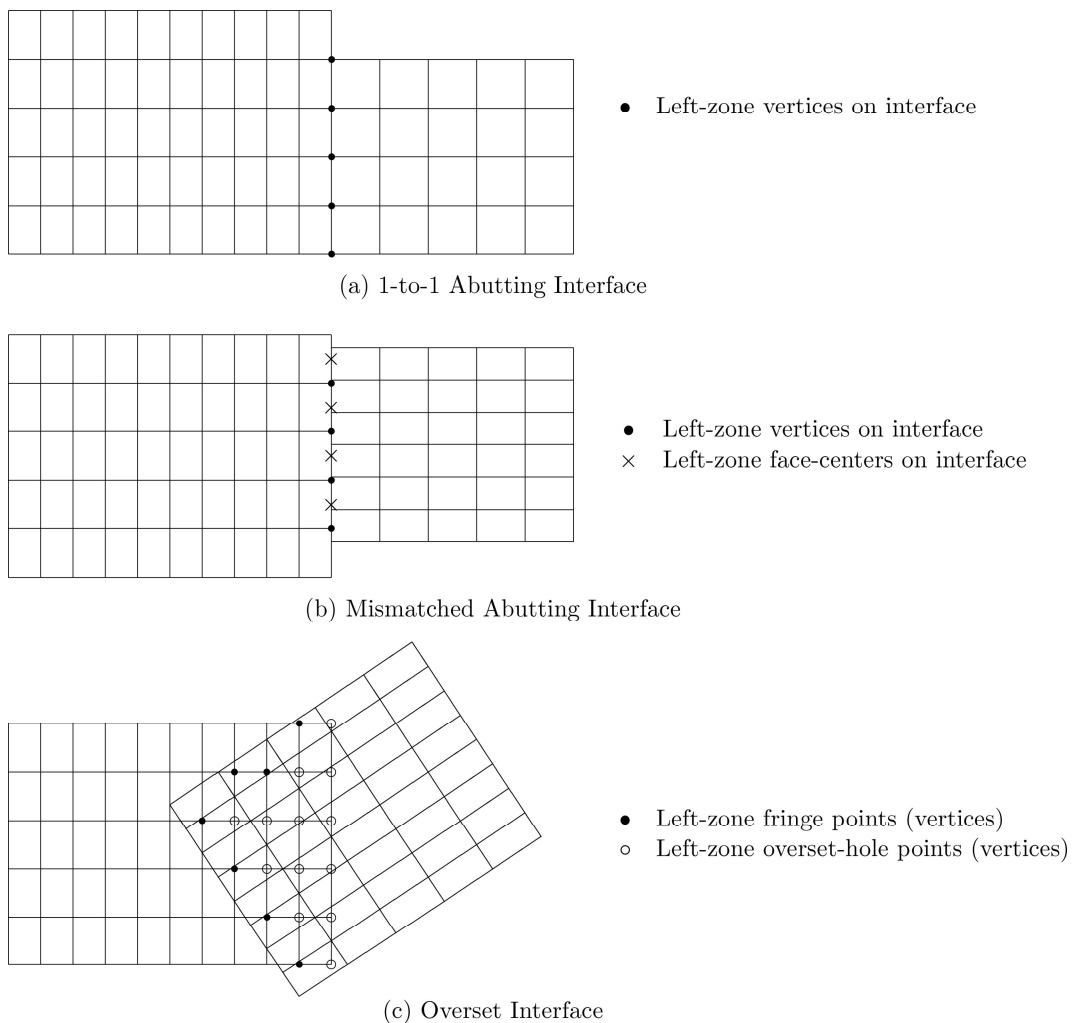


Figure 5: Three different methods to interface two sub-meshes in a block-mesh representation. [7]

2.2 Unstructured mesh

In unstructured mesh, there is no topological structure, meaning cells and their neighbours cannot be located using index formulas. To define unstructured mesh explicitly, all the point

coordinates and all element constructs have to be defined explicitly. Two examples of unstructured meshes are presented in Figure 6. The example geometry is so simple that also mapping could have been used to create a structured mesh. Instead, automatic meshing with tetrahedron elements was used. The example meshes represent a first order and a second order FEM meshes.

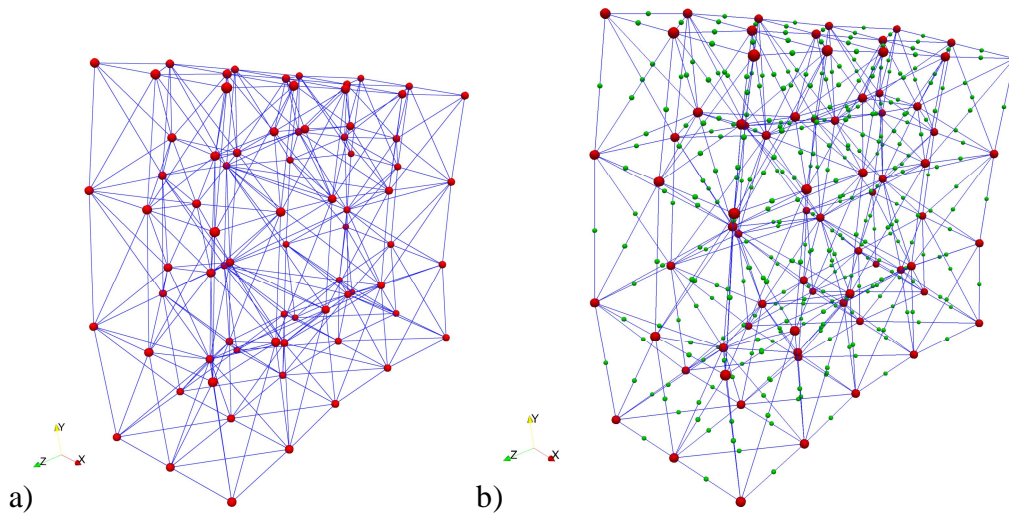


Figure 6: Example of unstructured meshes. Cells and points cannot be located using indexing. a) An example of a unstructured CFD mesh, also referred as a first order FEM mesh. b) An example of higher order elements in a finite element method mesh. These second order elements have midpoints on edges (marked in green).

2.3 Boundary and initial conditions

Boundary conditions in general mean given values for some specific properties on some specific computational domain boundary area. These can be e.g. surface temperature on a flow container inner surface (see Figure 7) or flow velocity derivative on inlet. Boundary condition properties can be various computational values that fix the computation so that there is an explicit solution. For mesh-based computation, the boundary conditions are usually applied to named boundary patches, i.e. sets of cell faces, and they can be constant or vary in time.

For a computation, *initial condition* means initial values for independent variables in the computational formulation. For a simulation, i.e. computing solution in time, computations practically always require proper initial conditions to be set. Initial condition is typically set to every computational cell in the computational domain. Initial values can be obtained from previous computations or they can be set analytically or based on experimental values.

2.4 Stored data and data types

The computational mesh can be thought to be a container for the actual computational data. The data can be e.g. flow field, structural stress, or magnetic field flux. In practice, the data type is integer, long integer, floating point, or double, and for one cell it can be in scalar, vector, matrix, or tensor form. In addition, when the computation is used for simulation, there is the time or frequency component. Thus, the results for a simulation are multi-dimensional, which introduce challenges to results data management. Depending on the task in hand the primary dimension of handling the data may vary and still the data should be efficiently usable. In Figure 8 is illustrated the multi-dimensionality of spatially discretised simulation data. In the figure below, the *mesh* dimension contains all the computational cells of the mesh in

used spatial directions, the *results components* are the separate quantities, like pressure and flow velocity, which in turn can be multi-dimensional (e.g. 3-component vector for velocity or 9-component rank 2 tensor for stress). The third dimension in the figure is *time* (discretised into time steps).

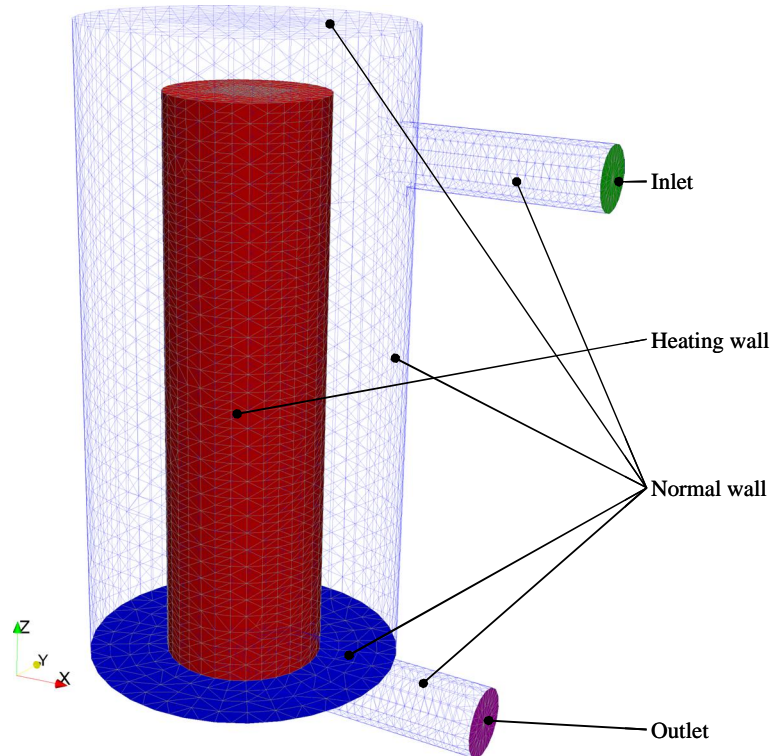


Figure 7: An example of a flow container model; patches and boundaries in a computational mesh. Here different patches are marked with different colours. Note that patches do not have to be bound by geometrical features (e.g. edges).

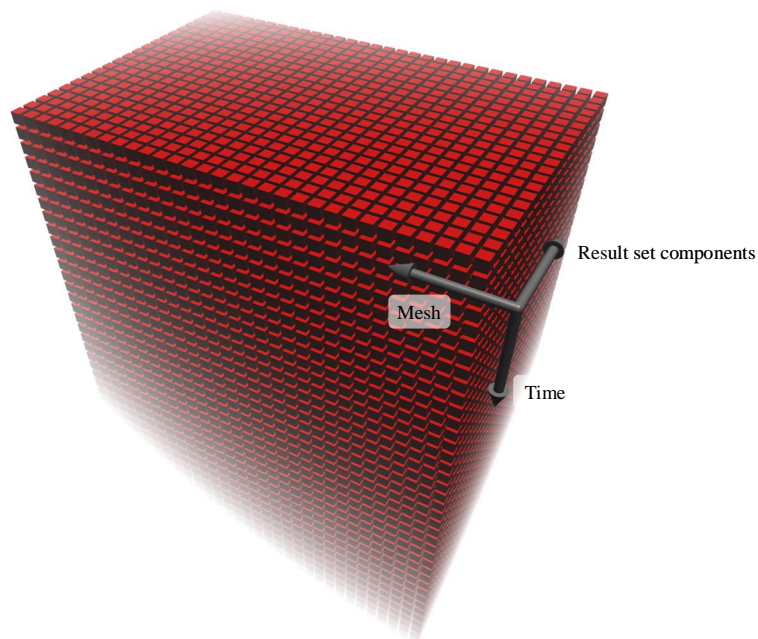


Figure 8: Dimensionality of computational results.

3 Existing definitions for mesh data

Spatial discretisation is strongly dependent on tools that use computational meshes. Mesh types, like structured and unstructured mesh, are often handled differently in computational tools and are often not interchangeable. Some tools require the mesh to be specific type, e.g. structured with hexahedral cells (elements) and others are flexible in mesh and cell type. This has to be taken into account when trying to develop a general method to describe computational meshes, and the fact that there cannot be just one universal representation has to be accepted. In addition, conversions from one mesh type to another may be impossible due to lack of necessary information e.g. structural differences of mesh types. One cannot convert unstructured mesh to structured one, if the original mesh does not have a suitable structure.

There are many definitions for mesh representations, both for numerical computing and for visualisation. In this report, the following three common mesh representations are selected for further study for basis of a semantic generic mesh representation: CFD General Notation System CGNS, OpenFOAM internal mesh representation, and Universal File Format (UNV or UFF). Formats, like *Common Data Format (CDF)* by National Aeronautics and Space Administration (NASA) and *Network Common Data Form (NetCDF)* by University Corporation for Atmospheric Research (UCAR), are not included into this study.

3.1 CFD Generic Notation System CGNS

CGNS [2], [8] is an effort to standardise CFD input, storage, and output, including grid (both structured and unstructured), flow solution, connectivity, boundary conditions, and auxiliary information [3], [4]. The CGNS specification is published as an AAIA (American Institute of Aeronautics and Astronautics) recommended practice (R-101A-2005) [5]. A CGNS database describes the current state of one or more entire CFD problems, including the following:

- mesh,
- flow field,
- boundary conditions,
- topological connection information, and
- auxiliary data (e.g., non-dimensionalisation parameters, and reference states).

CGNS is in practice a general file format for CFD-specific data. The intention is to ease exchange, maintenance and storage of both CFD model and results data. The specification is designed to fulfil various needs of different software tools. The CGNS package includes the specification documents and software implementation of I/O routines and utility programmes.

CGNS defines structure for the stored data as well as naming and semantics of the data in the database. In Figure 9 is shown the principle how data is organised in a CGNS database file. The CGNS definition can be divided into two parts: 1) the specification of the form of data in database, and 2) the implementation of the database file routines. The latter is the implementation for the first, using either the original CGNS file I/O system Advanced Data Format (ADF) or more general Hierarchical Data Format, version 5 (HDF5). In Figure 10 is shown how the CGNS data is organised in a HDF5 file.

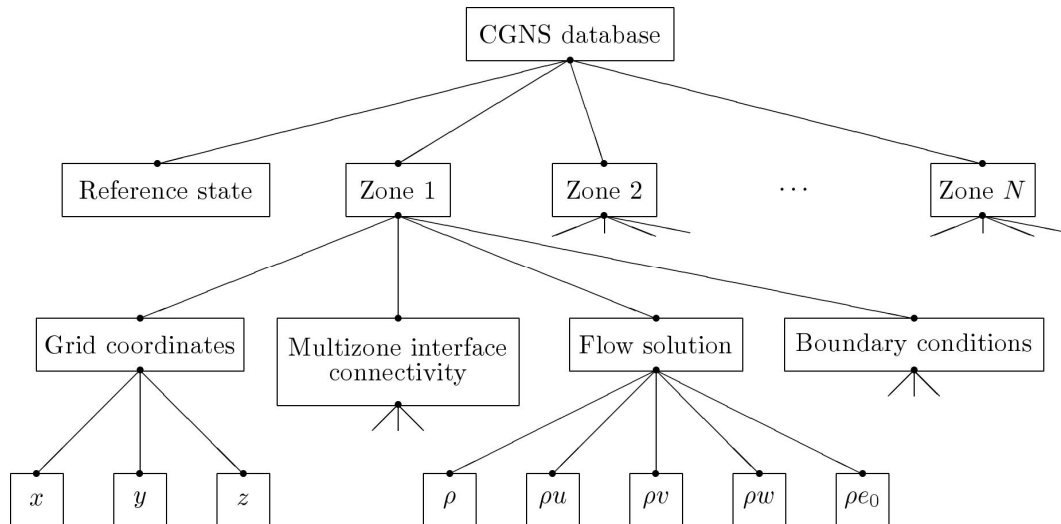


Figure 9: Data organisation in CGNS database file. [6]

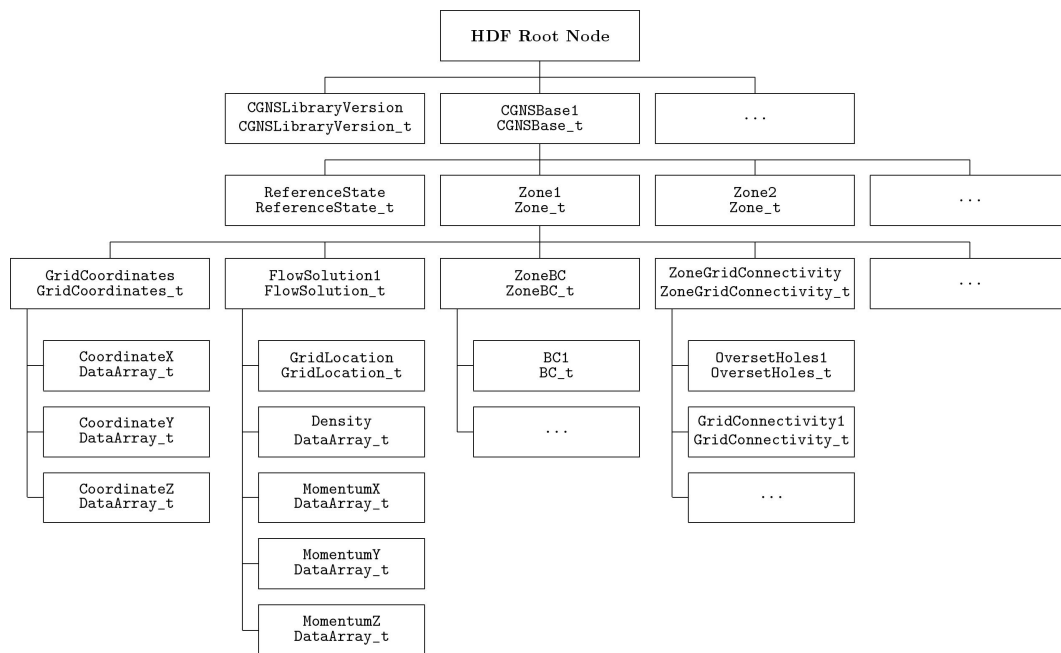
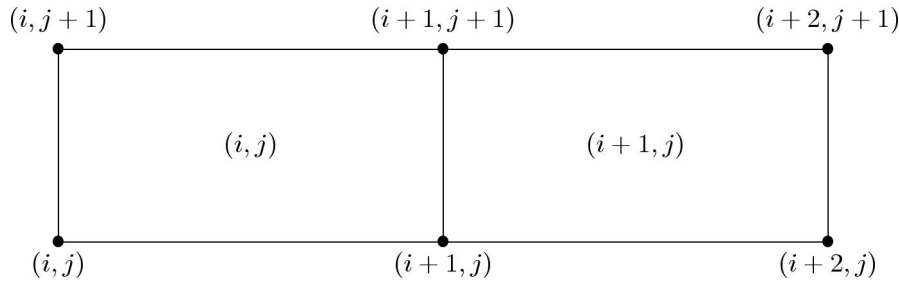


Figure 10: An example of CGNS data hierarchy inside a HDF file. [7]

The *CFD General Notation System Standard Interface Data Structures* [6] document describes well the form of structured mesh data CGNS [p. 17–18]:

A grid is defined by its vertices. In a 3-D structured grid, the volume is the ensemble of cells, where each cell is the hexahedron region defined by eight nearest neighbor vertices. Each cell is bounded by six faces, where each face is the quadrilateral made up of four vertices. An edge links two nearest-neighbor vertices; a face is bounded by four edges.

Cell centers, face centers, and edge centers are indexed by the minimum i , j , and k indices of the connecting vertices. For example, a 2-D cell center (or face center on a 3-D grid) would have the following convention:



In addition, the default beginning vertex for the grid in a given zone is $(1, 1, 1)$; this means the default beginning cell centre of the grid in that zone is also $(1, 1, 1)$.

Unstructured mesh representation is described in detail in the *CFD General Notation System Standard Interface Data Structures* [6] (CGNS SIDS) document (e.g. Section 3 *Unstructured Grid Element Numbering Conventions*). The main principle is that an unstructured mesh is described explicitly listing point coordinates and coordinate point indexes for given cell types. There are eight basic cell types in CGNS (listed in table below):

Table 1: The eight CGNS supported cell types. [6]

Dimensionality of the Element	Shape	Linear Interpolation	Quadratic Interpolation
0-D	Point	NODE	NODE
1-D	Line	BAR_2	BAR_3
2-D	Triangle	TRI_3	TRI_6
3-D	Quadrangle	QUAD_4	QUAD_8, QUAD_9
	Tetrahedron	TETRA_4	TETRA_10
	Pyramid	PYRA_5	PYRA_14
	Pentahedron	PENTA_6	PENTA_15, PENTA_18
	Hexahedron	HEXA_8	HEXA_20, HEXA_27

There is an additional general element type NGON_n for cells that do not fit into these. The index numbering order is essential for unstructured mesh. The CGNS SIDS [6] describes in detail the index numbering for each cell type.

CGNS specification defines 21 simple and 3 compound boundary condition types [6]. Common to these boundary conditions is that they refer to a specified set of cell faces and they have type specific properties.

3.2 OpenFOAM internal mesh format

The OpenFOAM internal mesh format is designed for flexible CFD mesh representation. The mesh is in principle unstructured, meaning cells are not indexed in curvilinear coordinate directions, and its cell type is an arbitrary polyhedral cell. This type of mesh is also called “arbitrarily unstructured” [9]. That means one computational mesh can contain different kind of cells, if the main mesh criteria is fulfilled. A valid OpenFOAM mesh must conform to the following criteria [10]:

1. **Contiguous:** The cells must completely cover the computational domain and are must not overlap one another.
2. **Convex:** Every cell must be convex and its cell centre inside the cell.
3. **Closed:** Every cell must be closed, both geometrically and topologically where:

- geometrical closedness requires that when all face area vectors are oriented to point outwards of the cell, their sum should equal the zero vector to machine accuracy;
 - topological closedness requires that all the edges in a cell are used by exactly two faces of the cell in question.
4. **Orthogonality:** For all internal faces of the mesh, we define the centre-to-centre vector as that connecting the centres of the 2 cells that it adjoins oriented from the centre of the cell with smaller label to the centre of the cell with larger label. The orthogonality constraint requires that for each internal face, the angle between the face area vector, oriented as described above, and the centre-to-centre vector must always be less than 90° .

The advantage with OpenFOAM mesh definition is that it can represent almost any kind of volume mesh, including structured and unstructured meshes. The disadvantage is that once a structured mesh is described in OpenFOAM notation, it loses the information of its structural topology. A complex algorithm is needed to analyse the data for unstructured mesh to check if the points are actually structurally ordered and indexable.

In OpenFOAM the polyhedral mesh is represented with a list point coordinates, a list of cell faces represented with references to the point indexes (point location in the point list), and two cell lists one representing owner cells for each face and the other representing neighbouring cells for each face. In Figure 11 is shown an example of a simple two-cell mesh with hexahedral elements. In Figure 12 is shown the OpenFOAM representation of this mesh.

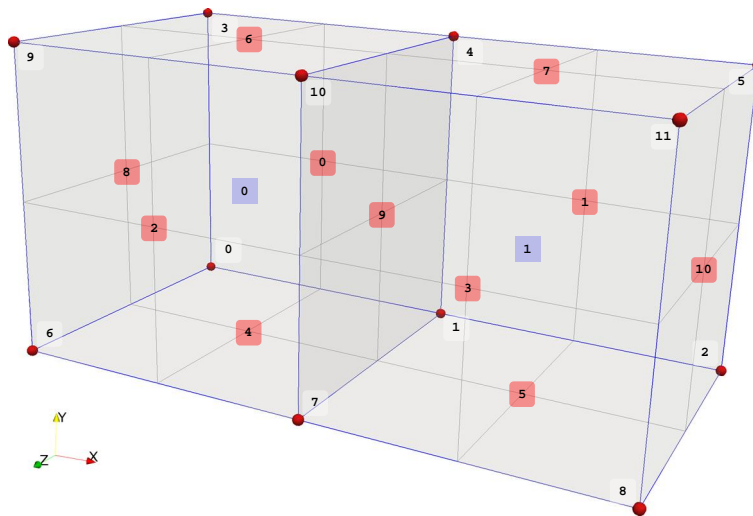
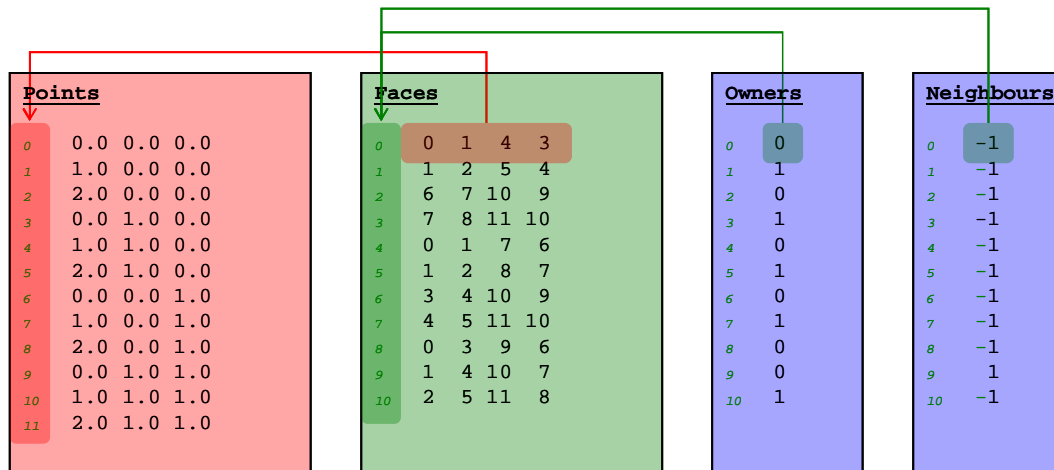


Figure 11: An example of a two-cell mesh with hexahedral cells.

3.3 Universal File Format

Universal File Formats (file type .unv or .uff) are a set of file format descriptions originally developed by Structural Dynamics Research Corporation (SDRC), nowadays part of Siemens AG. The origin of Universal File Formats is at late 1960's and early 1970's and they were developed to transfer data between CAD/CAE systems. Although these file formats are rather old and their design relatively simple and limited, they are still quite often used due to the open specification of the formats. There are currently 420 different valid dataset definitions in the Universal File Format list. [11]



Points				
0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0
2	2.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0
4	1.0	1.0	0.0	0.0
5	2.0	1.0	0.0	0.0
6	0.0	0.0	1.0	0.0
7	1.0	0.0	1.0	0.0
8	2.0	0.0	1.0	0.0
9	0.0	1.0	1.0	0.0
10	1.0	1.0	1.0	0.0
11	2.0	1.0	1.0	0.0

Faces				
0	0	1	4	3
1	1	2	5	4
2	6	7	10	9
3	7	8	11	10
4	0	1	7	6
5	1	2	8	7
6	3	4	10	9
7	4	5	11	10
8	0	3	9	6
9	1	4	10	7
10	2	5	11	8

Owners	
0	0
1	1
2	0
3	1
4	0
5	1
6	0
7	1
8	0
9	0
10	1

Neighbours	
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	1
10	-1

Figure 12: OpenFOAM Mesh representation for a two-cell mesh with hexahedral cell type.

Universal files are ASCII data files designed so that they may be easily read and written using user-written programs. Universal file is a sequential formatted file with lines of 80 characters maximum.

Blocks of information called datasets make up the basic structure of a universal file. Each block begins and ends with a block delimiter. This is a line containing a minus sign ‘-’ in column 5 and a one ‘1’ in column 6 and nothing else. The second line has an integer, which is the dataset number. For example, the node coordinates are written with dataset number 2411. Following the data type record, the body of the dataset contains data, which is dependent on the dataset type. The final record of the dataset is the delimiter line again.

Processing of the universal file begins by searching for the first delimiter line. Next, the dataset type line is processed to determine whether or not the program should process the dataset. If it is processed, the data is read per the specifications in the remainder of this section in the manual. If the dataset is not processed, the program continues reading until the next delimiter line indicating that the end of the dataset is encountered. Then the program searches forward for the next delimiter indicating the beginning of a new dataset. This processing continues dataset-by-dataset until the end of the file.

The most important datasets used in conjunction with mesh data are listed in Table 2. It should be noted that the same data can be given with different formats and dataset numbers depending on the revision of the I-DEAS software.

The general structure of a dataset can be seen in dataset 2411 for nodes. A couple of lines from the beginning of this dataset is listed below:

```

-1
2411
    11      1      1      11
0.000000000000000000D+00  0.000000000000000000D+00  4.90000000000000005D-01
    12      1      1      11
-3.000000000000000004D-01  0.000000000000000000D+00  4.90000000000000005D-01
    13      1      1      11
.
.
-1
    
```

The dataset begins with -1 followed by dataset number in the following line. In the first data line, parameters for the node are given including the node label, coordinate system number used in giving the location in space, coordinate system number for displacements and node

colour. In the second data line the coordinates of the node are given. These two datelines are repeated for each node. The end delimiter for the data is ‘-1’.

Table 2: Universal datasets for mesh data.

Dataset	Description
55	Data at Nodes
58	Function at Nodal degree of freedom
151	Header
164	Units
775 & 776	Beam Cross Section Properties
754	Constraint Sets
791	Restraint Sets
1716	Material Database Material
2400	Model Header
2411	Nodes - Double Precision
2412	Elements
2420	Coordinate Systems
2470	Physical Properties
2477	Permanent Groups

4 General semantic data model for spatially discretised data

4.1 Possible strategies

Three major strategies for mesh representation in a semantic database can be selected:

1. A generic representation for all different kind of meshes.
2. Native mesh representation for different kind of meshes and mappings from one to another (applicable parts).
3. A hybrid solution with couple of different major mesh representations and mappings between them (applicable parts).

Another aspect, when selecting the data model for a semantic representation, is the level of semantics in the model. For a mesh representation, at least two solutions are available:

- Fully semantic mesh representation, in which all data is described using triples, and
- Meta-level semantic mesh representation, in which only mesh topology and high-level components are represented semantically, and lists or table structures are used for detailed data.

These strategies are discussed briefly in the following sections.

4.1.1 A generic semantic mesh representation

The first strategy above is very demanding and in practice difficult to achieve. If all different mesh types (e.g. structure hexahedral meshes and unstructured tetrahedral meshes) are represented using one general data model, either some information of the original mesh is lost or the mesh definition becomes very complex. Because conversion from e.g. unstructured tetrahedral mesh to structured mesh is not possible, due to missing information about the structure, the general mesh representation had to be unstructured. Then, with structured hexahedral

mesh, if the structural information is lost, the conversion back from unstructured to structured mesh is challenging; that would require an algorithm to be developed which could figure the structure of the mesh. This strategy would require implementation of import and export routines that convert from specific formats to the generic semantic representation and vice versa.

4.1.2 Semantic native mesh representations

The second strategy is quite straight forward, but the drawback comes with several conversions and mappings between different mesh representations. Although, the advantage of a semantic database is the expressive representation and ability to map different representations, the management of tens of different mesh representations may become too difficult. Mesh reading and writing routines should be quite straightforward to implement, when the semantic representation strictly follows the original format. On the other hand, the design and implementation of mappings between different mesh formats may be more complicated, depending on the conceptual differences between the formats.

4.1.3 Semantic hybrid mesh representation

The third strategy is a hybrid method in which couple of major representations are described in semantic form and mappings of applicable parts are done in the semantic database. This would require implementation of both format specific reading and writing routines and mappings between different semantic mesh representations, but might be the most flexible solution for future requirements.

4.1.4 Fully semantic mesh representation

A full semantic mesh representation means all the data for a computational mesh is described using semantic triples, including individual points, cell faces, cells, and the whole mesh. This procedure introduces full flexibility of using all semantic methods for querying, modifying and managing the mesh, but is very resource intensive and would not be possible for large meshes. A one cell mesh and its semantic connections are shown in Figure 13. In this case, the model contains 16 actual objects, 31 empty objects, 8 ending nil objects and 70 properties, totally 125 entities. For a 10 million cells model this would mean about 1.25 billion entities just to describe to plain mesh. The computational case would require also boundary and initial conditions to be defined. This example shows that the fully semantic mesh representation is most likely far too resource intensive and cannot be the solution for large computational meshes.

In Table 3 is represented a simple model for a fully semantic mesh representation. This model contains only main entities and their properties. Cell face normals are defined using the right hand rule, meaning the counter clockwise order of points defining a face define the positive face normal direction.

Table 3: Basic classes in the fully semantic mesh ontology.

Class	Description
Case	Collects all the components to define a computational case, including mesh, computation parameters and results.
Model	Collects all the components to define a model, including mesh and additional data but excluding computation parameters and results.
Mesh	Collects possible sub-meshes into one computational mesh.
SubMesh	Defines one sub-mesh.

Cell	Defines one computational cell.
Face	Defines one cell face.
Point	Defines one point with three coordinate values x , y , and z .
Boundary	Defines a boundary with one or more patches and additional data for boundary type.
Patch	Defines a face set that can be used for defining boundaries.
Results	Collects all computation results into one set.
TimeStep	Collects all the results data of one computational time step.
ResultSetComponent	Result of one quantity, e.g. pressure, for one time step.

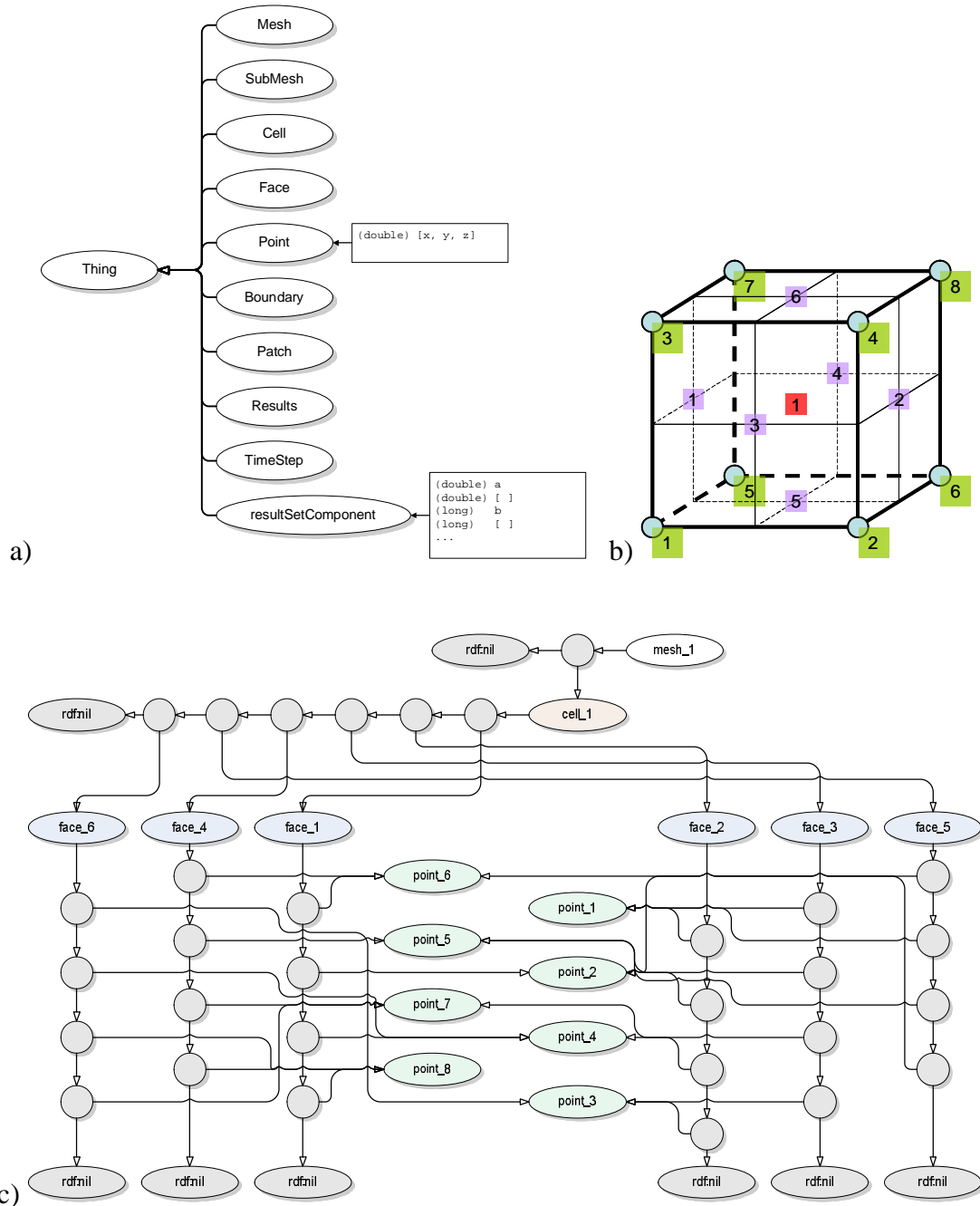


Figure 13: An example of fully semantic representation of a mesh of one cell. a) The ontology graph for fully semantic mesh representation. b) The example mesh and its components. c) The semantic model of the example mesh. Only some of the properties are shown.

4.1.5 Meta-level semantic mesh representation

The meta-level semantic mesh representation means that all the detailed mesh data, such as point locations and face and cell definitions, are presented as lists or tables and relations between these components are represented using semantic notation. This approach is suitable for e.g. the OpenFOAM mesh model in which the mesh is represented with lists containing the point location data and indexes to lists of faces and cells. The same example of a one-cell mesh as in Figure 13 is represented in meta-level semantic form in Figure 14. Here, the semantic model is used only to link the tables representing the relationship between the points, faces, cells, and the mesh. The *PointList* represents three coordinates for each point. The *FaceList* defines which points define each face by referring to *PointList* point index. Similarly, the *CellList* refers to *FaceList* by listing the face indexes that form cells.

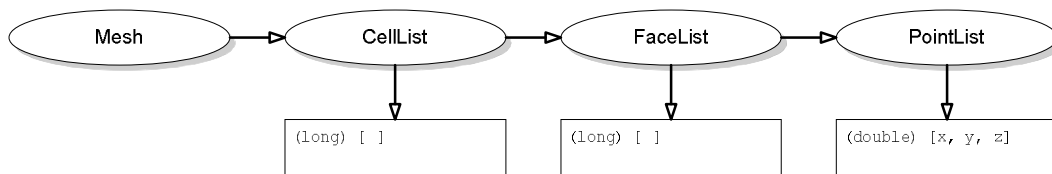


Figure 14: An example of meta-level mesh representation for the same mesh than in Figure 13. The semantic model is very lean compared to fully semantic representation, but on the other hand, the impressiveness of the representation is more limited.

4.2 Problem bounding

Computational meshes cannot be easily represented using one general notation due to different requirements of computational tools and mesh types. Due to this, the notation has to be flexible and extendable so that different needs can be fulfilled in the future.

When several representation formats are involved and when there is a need to convert formats from one to another, an efficient solution is to use an intermediate format and do the conversions through this format. By this method one need to create only $2 \times N$ format conversion routines for N formats; with straight conversions the number of conversion routines would be $N \times (N-1)$. Also, when a new format is added only two conversion routines are needed, one from the new format to the intermediate format and another to the other direction.

Before it is possible to actually implement a semantic mesh model and use it for real world modelling and simulation, the specific model for results data representation is left out from the scope of this report. Representing results of cases that may have a mesh of tens of millions of cells and thousands of time steps requires specific care for the design of the data model. In addition, the implementation and physical storage of the data in separate files introduce more requirements for the data model design.

4.3 Proposal for a generic semantic mesh data representation for CFD

In this section, a conceptual design of a generic semantic data model for CFD mesh data is discussed. The detailed design of a generic mesh data representation would require deeper analysis of the data models of existing CFD software.

4.3.1 Unstructured mesh

The proposal for notation of unstructured and thus more generic mesh representation is based on point coordinate value list, cell face mappings to the coordinate list, and cell mappings to the face list. The conforming rules presented in section 3.2 are applied.

The format consists of 1) a list of points presenting point coordinate values, 2) a list of cell faces represented using points (mapping to the point list), and 3) a list of cells represented using faces (mapping to the face list).

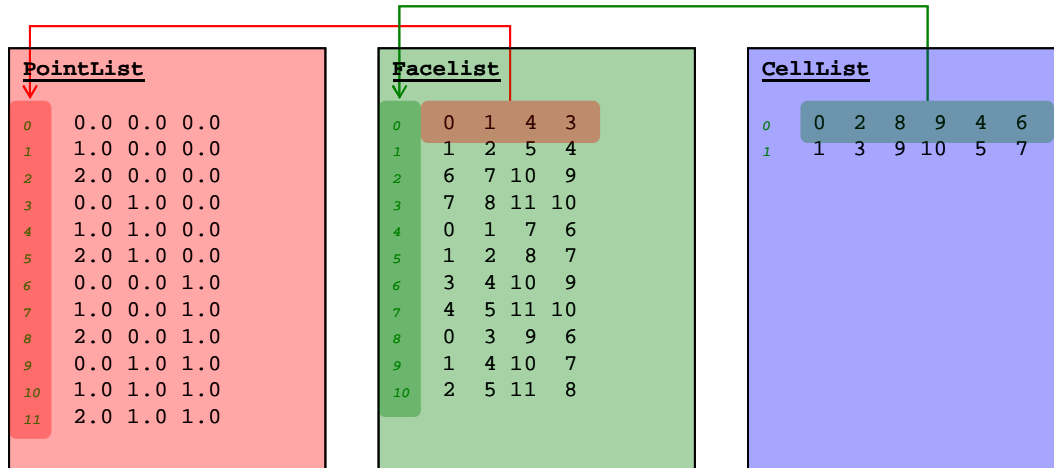


Figure 15: An example of mesh representation in the semantic database. The example mesh is shown in Figure 11.

Point list is a list of point coordinates:

```

0      x01 x02 x03
1      x11 x12 x13
2      x21 x22 x23
...
N      xN1 xN2 xN3
    
```

Where points are referred using the list index number starting from 0 (point place in the list, in green in above) and coordinate values x_{ni} are represented using doubles.

Face list is a list of indexes i_{mj} of points (in green in above) representing the corners of a face; the number of corner points n may differ for different faces in the list:

```

0      i01 i02 i03 ... i0n
1      i11 i12 i13 ... i1n
2      i21 i22 i23 ... i2n
...
M      iM1 iM2 iM3 ... iMn
    
```

Faces are referred using the list index number starting from 0 (face place in the list, in green in above).

Cell list is a list of indexes i_{mj} of faces. (in green in above) representing the faces of a cell; the number of faces n may differ for different cells in the list:

```

0      i01 i02 i03 ... i0n
1      i11 i12 i13 ... i1n
2      i21 i22 i23 ... i2n
...
M      iM1 iM2 iM3 ... iMn
    
```

Cells are referred using the list index number starting from 0 (face place in the list, in green in above). Neighbouring cells are found by searching cells with the same face index (there should be at most two cells with the same face index).

Additional information about the mesh can be stored as meta-data. This data can include e.g. cell type information for tetrahedral mesh. Some tools may operate more efficiently if the cell type is known beforehand and it does not have to be checked for every cell.

4.3.2 Structured mesh

For structured mesh only mesh point coordinates are needed in addition with information of index dimensions I , J and K in different topological dimensions. Point coordinates for structured mesh have to be listed in order of I , J , K , meaning the fastest changing point index is I , then J and finally K . Cells are defined so that points are listed first to lower K face in counter-clockwise (points 0, 1, 4, and 3 for cell 0 in Figure 11) and then the upper K face in counter-clockwise (points 6, 7, 10, and 9 for cell 0 in Figure 11). Ghost cells can be defined implicitly using expanded indexing, because the exact shape and locations of ghost cells' node points are not relevant.

In the semantic database, the index dimensions are stored as attributes. The semantic database can contain also the additional information for unstructured mesh representation purposes, like explicit face and cell lists. This data is optional for structured mesh and thus can simply be ignored.

4.3.3 Boundary and initial conditions

Boundary condition data is represented based on named boundaries. A boundary is a set of faces represented using face list indexes (a list of face indexes for the given boundary). In addition, a boundary has a type and, based on that, parameters and values, which may be constant or vary in time.

Initial conditions are represented based on either mesh or cell data. If initial condition for some specific quantity is constant for the whole mesh (e.g. temperature for the whole mesh is $T = 293.15$ K), it can be represented mesh-based giving just one value. If the initial condition values vary in space, it is represented cell-based by referring to the cell list and giving the quantity value for each individual cell.

4.3.4 Results data representation

Results data can be represented either point-based or cell-based. This means local data is calculated in a point location in the computational domain or a local value is in principle calculated in a cell centre point (this point does not usually exist in the mesh representation). The results data for each quantity is represented by referring to node list (if the data is computed in nodes) or to cell list (if data is computed in cells).

4.3.5 Semantic mesh data model

The data model for a mesh is shown in Figure 16. The mesh model includes representations for structured and unstructured mesh types, for boundary and initial conditions, and for results data representation. The graph in Figure 17 represents a case of a flow container with an inlet and an outlet channel. In Figure 18 is represented an example of composing a computation case with the designed ontology.

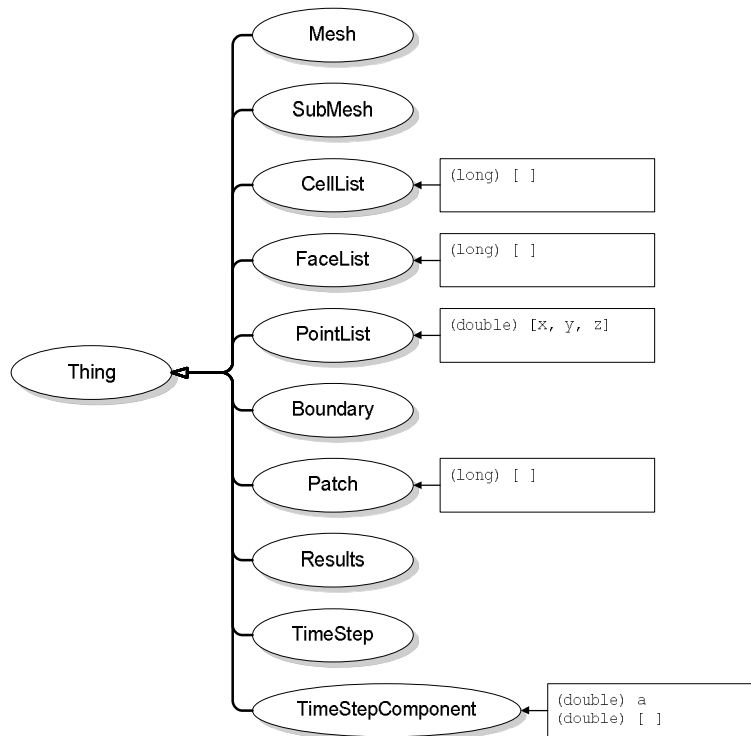


Figure 16: The mesh ontology for spatially discretised data.

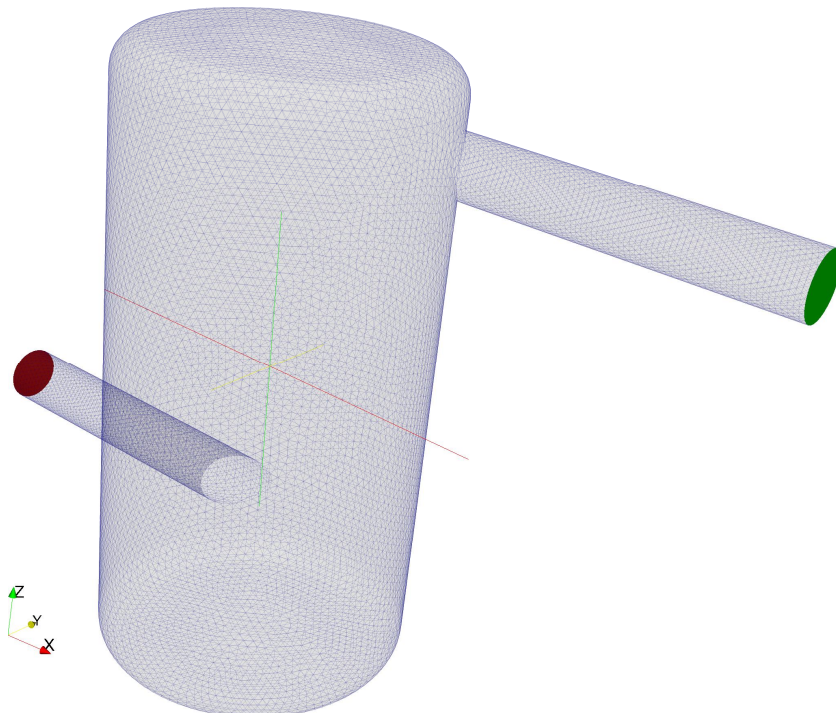


Figure 17: An example case of a flow container with an inlet and an outlet channel. The mesh is divided into three sub-meshes.

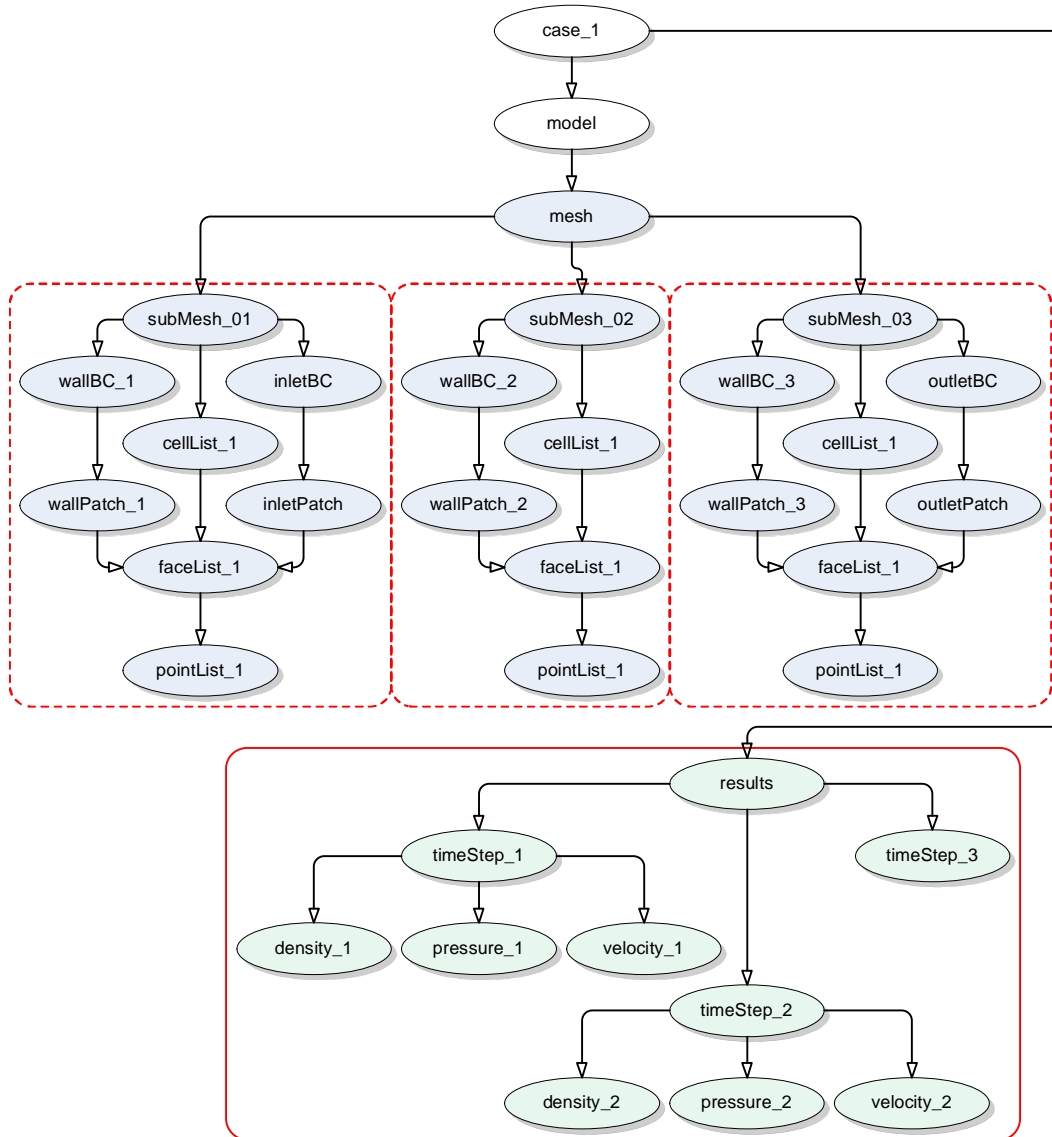


Figure 18: The graph representation for the example case shown in Figure 17. The mesh is divided into three sub-meshes. Case results contain values for fluid density (scalar), pressure (scalar), and velocity (vector).

4.4 General semantic data model for FEM models

Our final goal is to enable the use of a common semantic database for different tools like FEM and CVM programs. Though they appear many ways similar, they have so many differences that they both need their own semantic data models. In previous sections, mainly CVM has been discussed. However, the general observations made above apply also to FEM.

The objective is to create a semantic data model that can store all data for a FEM mesh and everything that is needed to run an analysis. More specifically the objective is to be able to store the FEM model and write it out from the database exactly as it was in the original input file.

Next a general semantic data model for FEM models is proposed. For the moment the description covers mainly the basic components of a FEM model, nodes, and elements. Also the properties that can be attached to them are included. Boundary conditions are defined as node

properties. Also point forces can be given the same way as the boundary conditions. However, the general description of loads is much more complicated and was not defined in this project.

FEM models can nowadays include millions of nodes and elements. The fully semantic mesh representation is out of the question because of the poor efficiency as discussed in chapter 4.1.4. Instead the list and table based meta-level semantic representation (see chapter 4.1.5.) has to be used.

FEM models need also another exception to the pure triplet based semantic representation. Many FEM programs attach properties to nodes and elements sequentially e.g. the previous value is overwritten by the next attachment. For example, the thickness of all shell elements is at first set to 5 mm, but then the thickness of an element group is changed to 9 mm. This sequence of the actions must be preserved and it is naturally preserved with the list based meta-level semantic model. However, not all FEM programs use the sequential method. So, occasionally, when a FEM model is transformed from one program format to another, the sequential attachments have to be interpreted.

Input files of several FEM programs (Abaqus, Ansys, Adina, Code_Astér, Elmer, I-Deas, Nastran) were studied to ensure the generality of the developed semantic data model.

4.4.1 Description of general semantic data model for FEM meshes

In this section, a general semantic data model for FEM meshes is described. The data model was not designed just for a passive “store-once” database, but for an active database, where data can be efficiently and easily updated and modified. To achieve this goal the data model is not as direct description of a FEM model as possible. For instance, node numbers (*nid*) are not used to refer to a node but its index (*nind*) in *NodeList*, and node sets are not accessed by their name (*nsname*) but indirectly by their index (*nsind*) in *NodeSetList*, which contains pointers (*nsref*) to the node sets.

4.4.1.1 Header data

The proposed general semantic data model includes two lists of header or meta data.

HeaderList is a list of lines (e.g. strings) that contains information of the stored model. This information is only for the user and can any kind of description of the model. It is recommended to begin with a name for the FEM model, the name of program, which was used to create it, creation date, and the author. The FEM program and the version of the program for which the FEM model has been created is stored with elements.

UnitList is a list of keywords, names of units and scaling factors. By multiplying a value with the assigned scaling factor gives the corresponding value in SI units. The units are not listed in any defined order. The quantities of identified by keywords that correspond the names of quantities defined in ISO 80000 standard. If a quantity is not defined in *UnitList*, it is assumed to be given in SI units. Since there is only one scaling factor per quantity, all values of the same quantity must be given with the same unit. For example, quantity length, if shell thicknesses are given in mm, also node coordinates must be in mm. However, density can still be in kg/m^3 and pressure in MPa.

4.4.1.2 General semantic data model for nodes

The proposed general semantic data model for nodes is presented in Figure 19. It is a meta-level semantic representation of the node data. All the data is represented as lists. The lists are shown as coloured boxes with the header to identify the contents of the list. The coloured

lower part is the actual list. The white columns with numbers on the left side of some lists are not part of the list. They have been added to illustrate the use of indices to refer to items in those lists.

CoordinateSystemList is a list of coordinate systems used in the FEM model. They are identified with a name (*csname*) by the user. The type of the coordinate system is defined with a keyword (*cstype*) like ‘cylindrical’. How much other data is needed depends on the type of the coordinate system.

NodeList contains the basic data of the nodes: three coordinate values and the node identification (*nid*) for the user, a node number or a label string. Since the node coordinates can be given in different coordinate systems, the used coordinate system is identified with the reference *csind*.

NodeSet is a list of node indices (*nind*) identified by its name (*nsname*) by the user. A node set can include also other node sets by giving a negative *nid* e.g. $-nsind$. Node set are used in FEM programs to attach properties to nodes, to define boundary conditions etc.

NodeSetList is a list of references to node sets (*nsref*). It is a switchboard by which the node sets are used. It was added to make the database more efficient and easier to modify.

NodePropertyList is a list of node set indices (*nsind*) and property values like thickness. There is a list for every property that is used in the FEM model. In Figure 19, two examples are given *NodeColorList* and *NodeThicknessList*. The basic function of a *NodePropertyList* is to attach the given property value to all nodes in the referenced node set. However, the property “value” can also be a list. Then the nodes in the node set are attached to the corresponding items in the property value list. If the value list is shorter than the node set, the last value in the list is attached to the rest of the nodes.

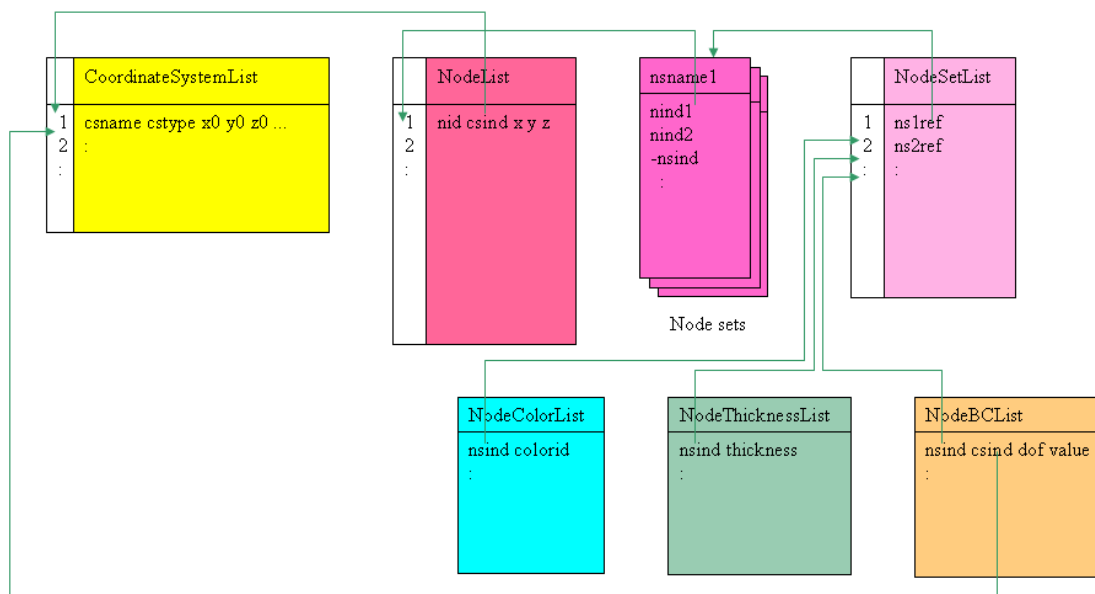


Figure 19: The general data model for nodes in a FEM mesh.

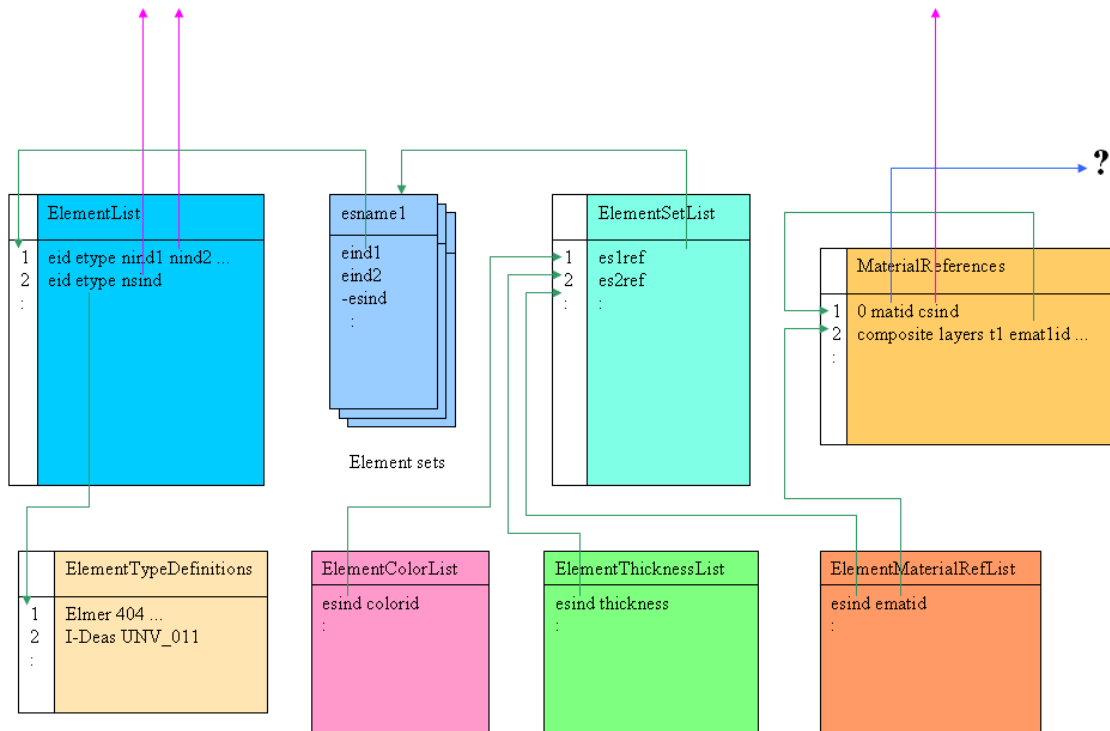


Figure 20: The general data model for elements in a FEM mesh.

NodeBCList defines the boundary conditions for the FEM mesh. It attaches the given boundary condition to nodes in the node set (*nsind*). Since the boundary conditions can be given in different coordinate systems, the used coordinate system is identified with the reference *csind*. Also, the degrees of freedom (*dof*) that are affected must be defined. Numbers from 1 to 6 refer to a single degree of freedom, but also keywords like ‘*xsymm*’, which refer to a set of degrees of freedom, can be used. Usually the nodes at the boundary are fixed, e.g. the *value* is 0. However, sometimes the displacements on the boundary are known and then they can be given with *value*.

4.4.1.3 General semantic data model for elements

The proposed general semantic data model for elements is presented in Figure 20. It is a meta-level semantic representation of the element data. All the data is represented as lists. The lists are shown as coloured boxes with the header to identify the contents of the list. The coloured lower part is the actual list. The white columns with numbers on the left side of some lists are not part of the lists. They have been added to illustrate the use of indices to refer to items in those lists.

ElementList contains the basic data of the elements: the type (*etype*) and the topology e.g. the list of nodes that define the geometry of the element. Instead of node numbers, the nodes are identified with *ninds* that refer to *NodeList* in Figure 19. Some special elements may have even thousands of nodes. They can be given with a node set (*nsind*). The element number of label is used by the user to identify the element (*eid*).

ElementTypeDefintions is simply a list of lines (e.g. strings) that contain the name and the version of the FEM program for which the model was targeted, and the element type as defined by the target program. When the element type is defined this way, FEM models from different programs can be combined and the original types of the elements are still preserved.

ElementSet is a list of element indices (*esind*) identified by its name (*esname*) by the user. An element set can include also other element sets by giving a negative *esind* e.g. *-esind*. Element sets are used in FEM programs to attach properties to elements, to define surfaces etc.

ElementSetList is a list of references to element sets (*esref*). It is a switchboard by which the element sets are used. It was added to make the database more efficient and easier to modify.

ElementPropertyList is a list of element set indices (*esind*) and property values like thickness. There is a list for every element property that is used in the FEM model. In Figure 20 a couple examples are given (*ElementColorList*, *ElementThicknessList*, and *ElementMaterialRefList*). The basic function of an *ElementPropertyList* is to attach the given property value to all elements in the referenced element set. However, the property “value” can also be a list. Then the elements in the element set are attached to the corresponding items in the property value list. If the value list is shorter than the element set, the last value in the list is attached to the rest of the elements.

MaterialReferences is a list of references to local or external material libraries. It is not properly defined, but rather illustrates what is required of such list: coordinate systems must be defined (*csind*) for orthotropic materials, composite materials must be built, etc. Materials are so complicated and big issue that they need their own data model, especially when material nonlinearities are taken into account.

4.4.1.4 Practical experience with Simantics

The semantic data model for FEM meshes described above was implemented in *Simantics* [1]. A graphical user interface was programmed for the Universal File Format (see chapter 3.3) to interact with the Simantics FEM ontology. The tested FEM models were created with I-Deas pre-processor, from which the models could be exported in Universal File Format. An example of a test FEM model shown in the Simantics graphical user interface is shown in Figure 21.

The database was tested with five FEM models. Four simple test models consisted entirely of 3D solid elements with 20-nodes and the last one was an industrial FEM model consisting of several element types. The biggest test model, 1 000 000 elements, could not be read, because of the too large memory requirement. The other results are presented in Table 4. The reading and writing of the UFF file is about as fast as with I-Deas pre-processor. However, the real test was the performance of the database. The results show that the saving of the data into the database and reading the model from the database is relatively fast except for the largest, 250 000 elements, test model. The slowing down may be due to the large memory requirement for this test model. The results show that the developed semantic data model for FEM meshes can be used as an active database.

Table 4. Time taken by the example FEM models

Number of elements	Reading input file [s]	Creating database [s]	Reading database [s]	Writing out input file [s]
10 000	0.7	0.6	0.1	1.5
80 000	5.6	2.7	0.2	11.8
150 000	14.0	7.0	0.8	24.1
250 000	16.6	40.8	13.5	66.1

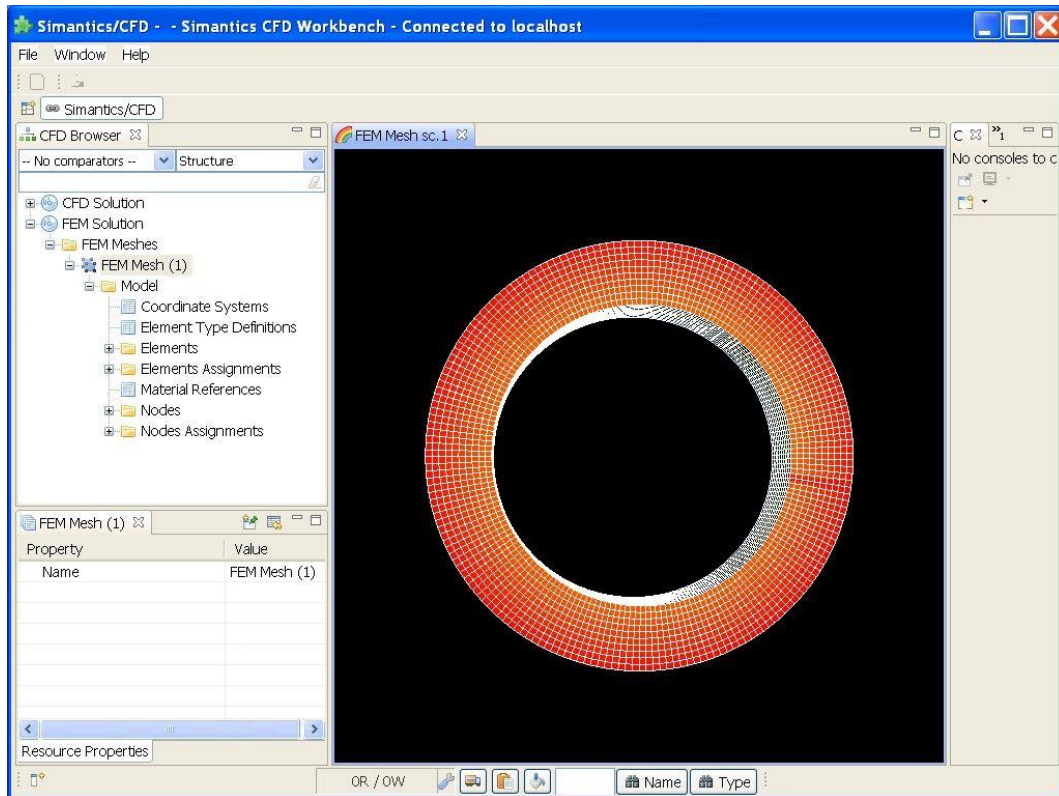


Figure 21: The graphical user interface of the semantic FEM data model in Simantics.

5 Summary

In this report, general features of spatially discretised data representation was introduced and existing definitions for CGNS, OpenFOAM mesh, and Universal File Format were studied to understand the needs and solutions for different domain areas. Different data models for general mesh representations in a semantic database were compared and a generic semantic computational mesh representation definition has been developed based on existing data models. The developed general mesh representation definition was used for developing and implementing a semantic data model for FEM data, based on Universal File Format. The implementation of the data import, storage, and export in the Simantics platform was verified and the performance of the implementation was tested and documented.

This study shows that spatially discretised data can be represented in semantic form and that there are several approaches for representing the data. The fully semantic approach provides the most flexible data model from semantic point of view by allowing semantic reasoning in single element level, due to semantic representation of each of the nodes and elements of the mesh. On the other hand, the fully semantic approach is also the most resource intensive and is probably too inefficient for industrial applications. The fully semantic data model for mesh representation was not implemented in the project. An approach between fully semantic data representation and plain tabular data was selected as the best compromise for the data model. In this approach, the data for nodes and elements and typically represented in tables, is stored in type-specific tables, and relations between different tables, and thus different type components, are represented semantically. The developed approach is still computationally efficient but has still the information about the semantic relations of the data components.

This study shows that developing a general data model for a simulation domain is not always a simple and straightforward task. Different software applications in one simulation domain, such as finite element method, have different methods of representing similar features. In addition, the set of features differ between different software applications, which make the development of a general data model challenging.

References

- [1] Simantics website: <https://www.simantics.org/simantics> (June 16th, 2009).
- [2] CGNS website: CFD General Notation System. <http://cgns.sourceforge.net/> (visited June 16th, 2009).
- [3] CFD General Notation System. *Overview and Entry-Level Document*. Document version 2.0.20. <http://www.grc.nasa.gov/WWW/cgns/overview/overview.pdf> (visited June 16th, 2009).
- [4] Rumsey, C., Poirier, D., Bush, R. & Towne, C. *CFD General Notation System. A User's Guide To CGNS*. Document Version 1.1.12. CGNS Version 2.5. <http://www.grc.nasa.gov/WWW/cgns/user/usersguide.pdf> (visited July 8th, 2009).
- [5] Recommended Practice: *The CFD General Notation System – Standard Interface Data Structures*. AIAA R-101A-2005. American Institute of Aeronautics and Astronautics, 1801 Alexander Bell Drive, Reston, VA 20191. http://www.grc.nasa.gov/WWW/cgns/sids/aiaa/R_101A_2005.pdf (visited June 16th, 2009).
- [6] CFD General Notation System. *Standard Interface Data Structures*. Document version 2.5.2, CGNS version 2.5. <http://www.grc.nasa.gov/WWW/cgns/sids/sids.pdf> (visited June 16th, 2009).
- [7] CFD General Notation System. *SIDS-to-HDF File Mapping Manual*. Document Version 1.1. CGNS Version 2.5. http://www.grc.nasa.gov/WWW/cgns/filemap_hdf/filemap_hdf.pdf.
- [8] Pättikangas T., Manninen M., Ilvonen M., Huhtanen R., & Luukkainen M. *Symbiosis between computational fluid dynamics and plant models*. VTT research report no. VTT-R-085820-6. 32 p. 2006.
- [9] OpenFOAM, the Open Source CFD Toolbox. *Programmer's Guide*. Version 1.6, 24th July 2009. <http://foam.sourceforge.net/doc/Guides-a4/ProgrammersGuide.pdf> (visited November 24th, 2009).
- [10] OpenFOAM, the Open Source CFD Toolbox. *User Guide*. Version 1.6, 24th July 2009. <http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf> (visited November 24th, 2009).
- [11] Website: *Universal File Formats for Modal Analysis Testing*. University of Cincinnati, Department of Mechanical Engineering, 2600 Clifton Avenue, Cincinnati, Ohio 45221. <http://www.sdrl.uc.edu/universal-file-formats-for-modal-analysis-testing-1> (visited June 15th, 2009).
- [12] Manola, F. & Miller, E. *RDF Primer*. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/REC-rdf-syntax/> (visited October 20th, 2009).
- [13] Smith, M., Welty, C. & McGuinness, D.L. *OWL Web Ontology Language Guide*. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/2004/REC-owl-guide-20040210/> (visited October 21th, 2009).