INTERNATIONAL STANDARD

ISO 19141

First edition
2008-06-01

Geographic information — Schema for moving features

Information géographique — Schéma des entités mobiles

Reference number
ISO 19141:2008(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword ... v

Introduction ... vi

1 Scope ... 1

2 Conformance ... 1
2.1 Conformance classes ... 1
2.2 Requirements ... 2

3 Normative references ... 2

4 Terms, definitions, and abbreviated terms ... 3
4.1 Terms and definitions ... 3
4.2 Abbreviated terms ... 5

5 Package – Moving Features ... 6
5.1 Semantics ... 6
5.2 Package structure ... 7
5.3 Class hierarchy ... 7

6 Package – Geometry Types ... 9
6.1 Package semantics ... 9
6.2 Type – MF_OneParamGeometry ... 9
6.3 Type – MF_TemporalGeometry ... 11
6.4 Type – MF_Trajectory ... 12
6.5 Type – MF_TemporalTrajectory ... 14
6.6 Class – MF_PositionExpression ... 20
6.7 Type – MF_SecondaryOffset ... 20
6.8 Type – MF_MeasureFunction ... 21

7 Package – Prism Geometry ... 22
7.1 Package structure ... 22
7.2 CodeList – MF_GlobalAxisName ... 23
7.3 Type – MF_LocalGeometry ... 25
7.4 Type – MF_PrismGeometry ... 27
7.5 Type – MF_RigidTemporalGeometry ... 28
7.6 Type – MF_RotationMatrix ... 29
7.7 Type – MF_TemporalOrientation ... 30

8 Moving features in application schemas ... 30
8.1 Introduction ... 30
8.2 Representing the spatial characteristics of moving features ... 31
8.3 Associations of moving features ... 31
8.4 Operations of moving features ... 31

Annex A (normative) **Abstract test suite** ... 32
A.1 Application schemas for data transfer ... 32
A.2 Application schemas for data with operations ... 32

Annex B (informative) **UML Notation** ... 34
B.1 Introduction ... 34
B.2 Class ... 34
B.3 Stereotype ... 34
B.4 Attribute ... 35
B.5 Operation ... 35
B.6 Constraint ... 36
B.7 Note ... 36

B.8 Association **36**
B.9 Role name **36**
B.10 Multiplicity **37**
B.11 Navigability **37**
B.12 Aggregation **37**
B.13 Composition **38**
B.14 Dependency **38**
B.15 Generalization **38**
B.16 Realization **39**

Annex C (informative) **Interpolating between orientations** **40**
C.1 Introduction **40**
C.2 Euler rotations and gimbal lock **40**
C.3 Interpolating between two orientation matrices **42**
C.4 Interpolating between other orientation representations **44**
C.5 Sample interpolation **45**

Bibliography **49**

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 document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 19141 was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

Introduction

This International Standard specifies a conceptual schema that addresses moving features, i.e., features whose locations change over time. This schema includes classes, attributes, associations and operations that provide a common conceptual framework that can be implemented to support various application areas that deal with moving features, including:

— Location Based Services,

— Intelligent Transportation Systems,

— Tracking and navigation (land-based, marine, or space), and

— Modeling and simulation.

The schema specifies mechanisms to describe motion consisting of translation and/or rotation of the feature, but not including deformation of the feature. The schema is based on the concept of a one parameter set of geometries that may be viewed as a set of leaves or a set of trajectories, where a leaf represents the geometry of the moving feature at a particular value of the parameter (e.g., a point in time) and a trajectory is a curve that represents the path of a point in the geometry of the moving feature as it moves with respect to the parameter.

Geographic information — Schema for moving features

1 Scope

This International Standard defines a method to describe the geometry of a feature that moves as a rigid body. Such movement has the following characteristics.

a) The feature moves within any domain composed of spatial objects as specified in ISO 19107.

b) The feature may move along a planned route, but it may deviate from the planned route.

c) Motion may be influenced by physical forces, such as orbital, gravitational, or inertial forces.

d) Motion of a feature may influence or be influenced by other features, for example:

   1) The moving feature might follow a predefined route (e.g. road), perhaps part of a network, and might change routes at known points (e.g. bus stops, waypoints).

   2) Two or more moving features may be "pulled" together or pushed apart (e.g. an airplane will be refuelled during flight, a predator detects and tracks a prey, refugee groups join forces).

   3) Two or more moving features may be constrained to maintain a given spatial relationship for some period (e.g. tractor and trailer, convoy).

This International Standard does not address other types of change to the feature. Examples of changes that are not adressed include the following:

— The deformation of features.

— The succession of either features or their associations.

— The change of non-spatial attributes of features.

— The feature's geometric representation cannot be embedded in a geometric complex that contains the geometric representations of other features, since this would require the other features' representations to be updated as the feature moves.

Because this International Standard is concerned with the geometric description of feature movement, it does not specify a mechanism for describing feature motion in terms of geographic identifiers. This is done, in part, in ISO 19133.

2 Conformance

2.1 Conformance classes

2.1.1 Introduction

This International Standard specifies four conformance classes (Table 1). They are differentiated on the basis of two criteria: purpose and level of complexity.

2.1.2 Purpose

This International Standard may be used in support of data transfer. Operations defined for objects are irrelevant to data transfer, which requires only descriptions of the state of the objects at the time of transfer. Thus, two conformance classes require only the implementation of attributes and associations of the classes specified in the schema. The other two conformance classes support the object-oriented implementation of systems or interfaces; they require implementation of operations as well as implementation of attributes and associations.

2.1.3 Complexity

Many applications do not need a complete description of the geometry of a feature and its orientation at any point in time. Their requirements are satisfied by describing the movement of a single reference point on the feature using its trajectory as specified in Clause 6. One pair of conformance classes supports these simple applications.

Other applications need knowledge of the positions at each time of all points or a significant subset of the points on a moving feature. They require the full description provided by the prism geometry specified in Clause 7.

Table 1 — Conformance classes

Complexity	Purpose	
	Data Transfer	Data with operations
Trajectory	**A.1.1**	**A.2.1**
Prism Geometry	**A.1.2**	**A.2.2**

2.2 Requirements

To conform to this International Standard, an application schema shall satisfy the requirements of the Abstract Test Suite in Annex A.

3 Normative references

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

ISO/TS 19103, *Geographic information — Conceptual schema language*

ISO 19107, *Geographic information — Spatial schema*

ISO 19108, *Geographic information — Temporal schema*

ISO 19109, *Geographic information — Rules for application schema*

ISO 19133, *Geographic information — Location-based services — Tracking and navigation*

4 Terms, definitions, and abbreviated terms

4.1 Terms and definitions

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

4.1.1
base representation
⟨moving features⟩ representation, using a local origin and local ordinate **vectors**, of a **geometric object** at a given reference time

NOTE 1 A rigid geometric object may undergo translation or rotation, but remains congruent with its base representation.

NOTE 2 The local origin and ordinate vectors establish an engineering coordinate reference system (ISO 19111), also called a local frame or a local Euclidean coordinate system.

4.1.2
curve
1-dimensional **geometric primitive**, representing the continuous image of a line

[ISO 19107:2003, definition 4.23]

NOTE The boundary of a curve is the set of **points** at either end of the curve. If the curve is a cycle, the two ends are identical, and the curve (if topologically closed) is considered to not have a boundary. The first point is called the start point, and the last is the end point. Connectivity of the curve is guaranteed by the "continuous image of a line" clause. A topological theorem states that a continuous image of a connected set is connected.

4.1.3
design coordinate reference system
engineering coordinate reference system in which the **base representation** of a moving object is specified

4.1.4
feature
abstraction of real world phenomena

[ISO 19101:2002, definition 4.11]

NOTE A feature may occur as a type or an instance. Feature type or feature instance shall be used when only one is meant.

4.1.5
feature association
relationship that links instances of one **feature** type with instances of the same or a different feature type

[ISO 19110:2004, definition 4.2]

NOTE Feature associations include aggregation of features.

4.1.6
feature attribute
characteristic of a **feature**

[ISO 19101:2002, definition 4.12]

4.1.7
feature operation
operation that every instance of a **feature** type may perform

[ISO 19110:2004, definition 4.5]

4.1.8
foliation
one parameter set of geometries such that each point in the **prism** of the set is in one and only one **trajectory** and in one and only one **leaf**

4.1.9
geometric object
spatial object representing a geometric set

[ISO 19107:2003, definition 4.47]

4.1.10
geometric primitive
geometric object representing a single, connected, homogeneous element of space

[ISO 19107:2003, definition 4.48]

NOTE Geometric primitives are non-decomposed objects that present information about geometric configuration. They include points, curves, surfaces, and solids.

4.1.11
instant
0-dimensional **geometric primitive** representing position in time

[ISO 19108:2002, definition 4.1.17]

4.1.12
leaf
⟨one parameter set of geometries⟩ geometry at a particular value of the parameter

4.1.13
location-based service
LBS
service whose return or other property is dependent on the location of the client requesting the service or of some other thing, object or person

[ISO 19133:2005, definition 4.11]

4.1.14
network
abstract structure consisting of a set of 0-dimensional objects called junctions, and a set of 1-dimensional objects called links that connect the junctions, each link being associated with a start (origin, source) junction and end (destination, sink) junction

[ISO 19133:2005, definition 4.17]

NOTE The network is essentially the universe of discourse for the navigation problem. Networks are a variety of 1-dimensional topological complex. In this light, junction and topological node are synonyms, as are link and directed edge.

4.1.15
one parameter set of geometries
function f from an interval $t \in [a, b]$ such that f(t) is a geometry and for each **point** $P \in f(a)$ there is a one parameter set of points (called the trajectory of P) $P(t) : [a, b] \rightarrow P(t)$ such that $P(t) \in f(t)$

EXAMPLE A curve C with constructive parameter t is a one parameter set of points c(t).

4.1.16
period
one-dimensional **geometric primitive** representing extent in time

[ISO 19108:2002, definition 4.1.27]

NOTE A period is bounded by two different **temporal positions**.

4.1.17
point
0-dimensional **geometric primitive**, representing a position

[ISO 19107:2003, definition 4.61]

NOTE The boundary of a **point** is the empty set.

4.1.18
prism
⟨one parameter set of geometries⟩ set of points in the union of the geometries (or the union of the **trajectories**) of a **one parameter set of geometries**

NOTE This is a generalization of the concept of a geometric prism that is the convex hull of two congruent polygons in 3D-space. Such polyhedrons can be viewed as a **foliation** of congruent polygons.

4.1.19
temporal coordinate system
temporal reference system based on an interval scale on which distance is measured as a multiple of a single unit of time

[ISO 19108:2002, definition 4.1.31]

4.1.20
temporal position
location relative to a **temporal reference system**

[ISO 19108:2002, definition 4.1.34]

4.1.21
temporal reference system
reference system against which time is measured

[ISO 19108:2002, definition 4.1.35]

4.1.22
trajectory
path of a moving **point** described by a one parameter set of points

4.1.23
vector
quantity having direction as well as magnitude

[ISO 19123:2005, definition 4.1.43]

4.2 Abbreviated terms

CRS Coordinate Reference System (ISO 19111)

SLERP Spherical Linear Interpolation

LRS Linear Referencing System (ISO 19133)

OCL Object Constraint Language (ISO/IEC 19501)

UML Unified Modelling Language (ISO/IEC 19501)

5 Package – Moving Features

5.1 Semantics

A moving feature can be modelled as a combination of movements. The overall motion can be expressed as the temporal path or trajectory of some reference point on the object (the “origin”), such as its center of gravity. Once the origin’s trajectory has been established, the position along the trajectory can be described using a linear reference system (as defined in ISO 19133). The “parameterization by length” for curves (as defined in ISO 19107) can be used as a simple linear reference if no other is available. The relationship between time (t) and measure value (m) can be represented as the graph of the t→ m function in a plane with coordinates (t, m). This separation of the geometry of the path and the actual “time to position” function allows the moving feature to be tracked along existing geometry.

Figure 1 illustrates how the concepts of foliation, prism, trajectory, and leaf relate to one another. In this illustration, a 2D rectangle moves and rotates. Each representation of the rectangle at a given time is a leaf. The path traced by each corner point of the rectangle (and by each of its other points) is a trajectory. The set of points contained in all of the leaves, and in all of the trajectories, forms a prism. The set of leaves also forms a foliation.

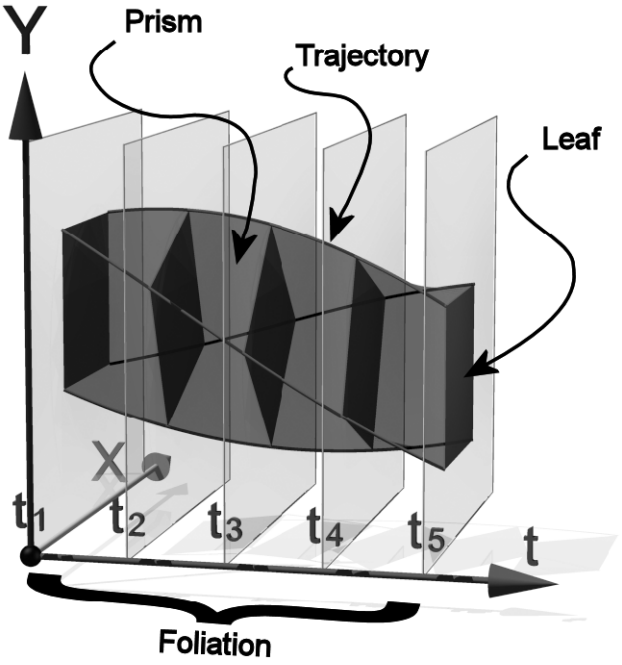

Figure 1 — Feature movement as foliation

These two object representations, of the path and the position along that path, give the general position of the moving feature. The other variable in describing the position of the feature is the rotation about the chosen reference point. To describe this, a local engineering coordinate system is established using the object reference point as its origin. The geometry of the feature is described in the engineering coordinate system and the real-world orientation of the feature is given by mapping of the local coordinate axes to the global coordinate system (the CRS of the trajectory of the reference point). This can be given as a matrix that maps the unit vectors of the local coordinate system to vectors in the global CRS.

If the global CRS and local CRS have the same dimension, then each point within the local CRS can be traced in time through the global CRS by combinations of these various mappings. The map would trace from time (t) to the measure (m) to a position on the reference point's path using the LRS. Then using the rotation matrix, the calculated offset from this point gives a direct position in the global CRS.

This means that the ‘prism’ of the moving feature (defined as all the points which part of the feature passes through) can be viewed (and calculated to whatever degree of accuracy needed) as a bundle of trajectories of

points on the local engineering representation of the feature's geometry. If viewed in a 4 dimensional spatio-temporal coordinate system, the points on the feature at different times are different points. Then the pre-image of the prism (points on the trajectories augmented by a time coordinate) is a foliation, meaning that there is a complete and separate representation of the geometry of the feature for each specific time (called a “leaf”). These names come from a 3D metaphor of a book, where each page or leaf is a slice of time in the “folio”.

This might form the basis for an extension of this standard to non-rigid, mutable objects. Each leaf in the 4D foliation is a separate representation of the object, and by creating methods to describe the change through time of the shape and form of the feature, the existing machinery in this International Standard can be used to place those representations in positions with respect to the global coordinate system.

5.2 Package structure

This clause presents a conceptual schema for describing moving features that is specified using the Unified Modelling Language (UML) [ISO/IEC 19501], following the guidance of ISO/TS 19103. Annex B describes UML notation as used in this International Standard.

The schema is contained in the UML package Moving Features. Names of classes included in this package carry the prefix "MF_". The package is subdivided into two leaf packages (Figure 2), Geometry Types and Prism Geometry. The classes in these two packages are derived from classes included in the Geometry Package specified in ISO 19107. Classes from the packages Basic Types [ISO/TS 19103], Geometry [ISO 19107], Temporal Objects, and Temporal Reference System [ISO 19108] are used as data types in the schema.

5.3 Class hierarchy

The classes of the moving features schema form an inheritance hierarchy that has its source in the classes GM_Object and GM_Curve specified in ISO 19107 (Figure 3). This allows the subclasses specific to this schema to be used as feature attributes in compliance with the General Feature Model specified in ISO 19109. The second level of the hierarchy consists of a set of classes that describe a one-parameter geometry. These might be used to describe the movement of a feature with respect to any single variable such as pressure, temperature, or time. The third level specializes these classes to describe motion in time. The classes are specified fully in Clauses 6 and 7.

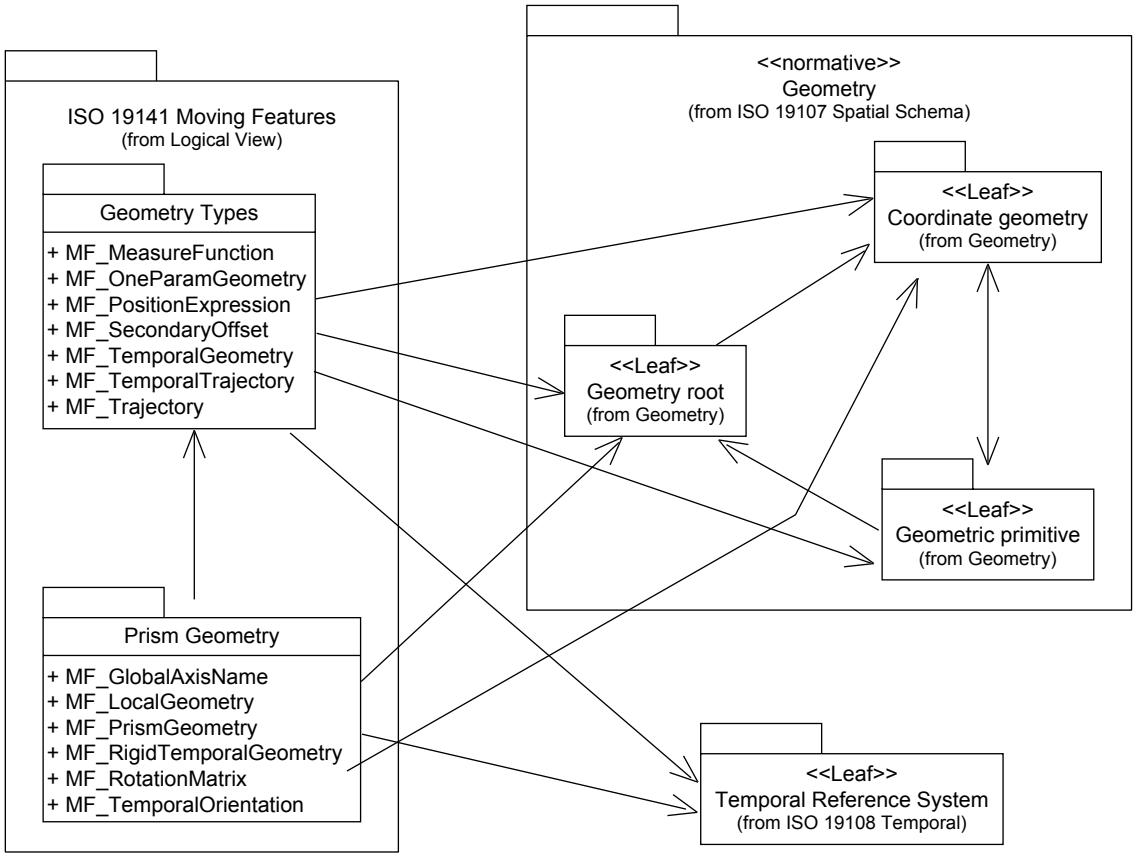

Figure 2 — Moving Feature Package

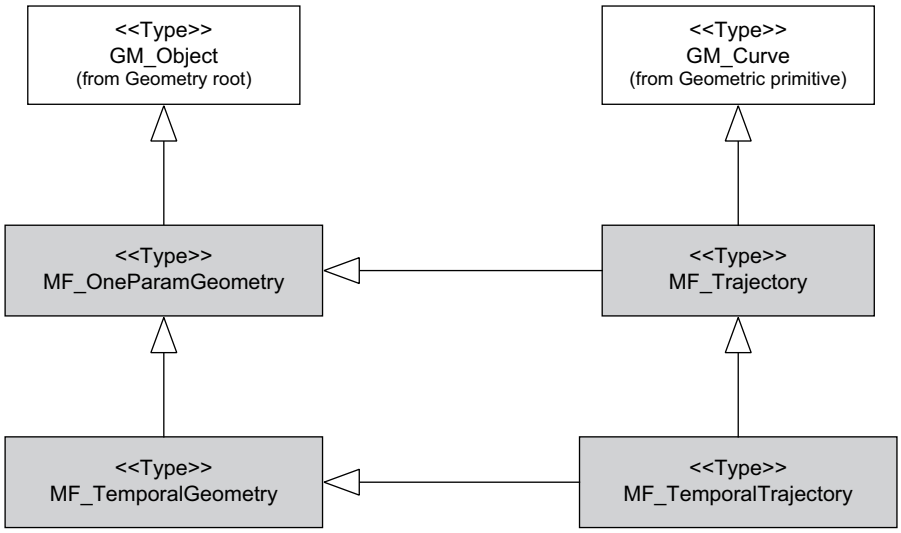

Figure 3 — Components of the Geometry Types Package

6 Package – Geometry Types

6.1 Package semantics

The Geometry Types package contains seven types. Two classes – MF_OneParamGeometry and MF_Trajectory – specify one-parameter geometry types based on the geometric objects specified in ISO 19107 (see Figure 3). Two other classes – MF_TemporalGeometry and MF_TemporalTrajectory – specialize the first classes in order to specify a one-parameter set of geometries in which the parameter is time. The other three classes – MF_MeasureFunction, MF_SecondaryOffset and MF_PositionExpression (Figure 4) – are used to extend the concepts of linear reference systems as defined in ISO 19133. Description of movement in terms of geographic identifiers is out of scope, and is partly covered in ISO 19133.

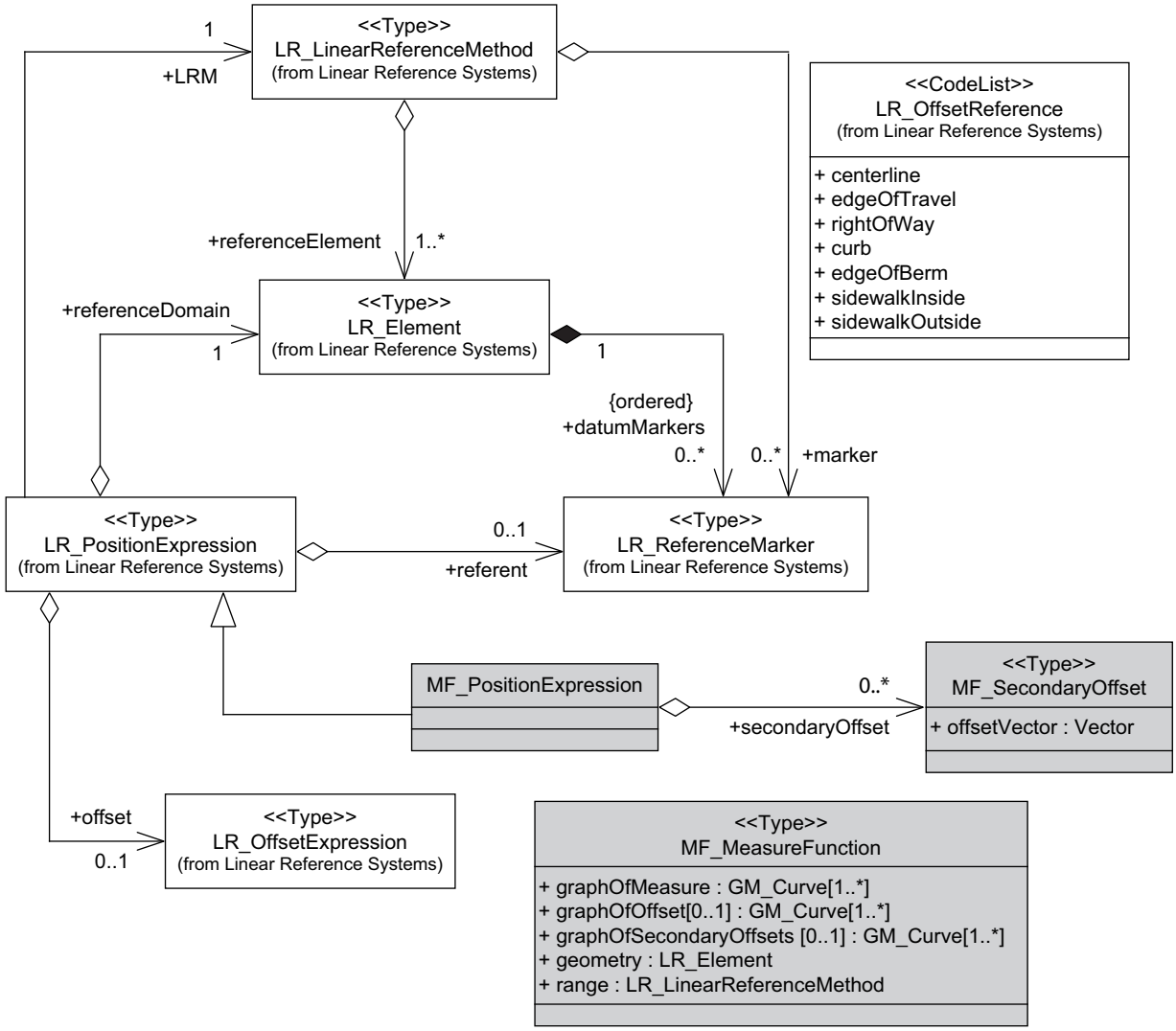

Figure 4 — Use of Linear Reference System by Moving Features

6.2 Type – MF_OneParamGeometry

6.2.1 Class semantics

A one parameter set of geometries is a function f from an interval $t \in [a, b]$ such that f(t) is a geometry and for each point $P \in f(a)$ there is a one parameter set of points (called the trajectory of P) $P(t) : [a, b] \rightarrow P(t)$ such that $P(t) \in f(t)$. A leaf of a one parameter set of geometries is the geometry f(t) at a particular value of the

parameter. The set of geometries forms a prism that is the set of points in the union of the geometries (or the union of the trajectories).

EXAMPLE A curve C with constructive parameter t is a one parameter set of points c(t).

6.2.2 Inheritance from GM_Object

The type "MF_OneParamGeometry" (Figure 5) inherits from the type "GM_Object." As such it shall implement all attributes, operations and associations inherited from that type as specified in ISO 19107, as well as those specified in this subclause.

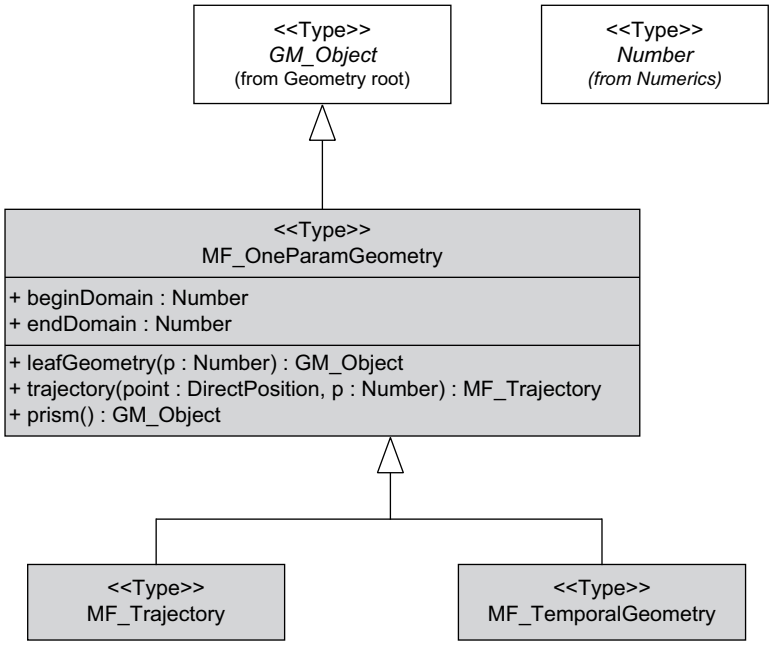

Figure 5 — Context Diagram: MF_OneParamGeometry

6.2.3 Attribute – beginDomain

The attribute "beginDomain" shall contain the value of the parameter at the start of the domain of the one-parameter geometry. The data type Number is specified in ISO/TS 19103.

```
MF_OneParamGeometry::beginDomain: Number
```

6.2.4 Attribute – endDomain

The attribute "endDomain" shall contain the value of the parameter at the end of the domain of the one-parameter geometry.

```
MF_OneParamGeometry::endDomain: Number
```

6.2.5 Operation – leafGeometry

The operation leafGeometry shall accept a value of the parameter as input and return the leaf associated with that value as an instance of GM_Object.

```
MF_OneParamGeometry::leafGeometry( p: Number ): GM_Object
```

6.2.6 Operation – trajectory

The operation trajectory shall accept the position of a point on a leaf (identified by a value of the parameter p) of the one parameter set of geometries and return the trajectory of that point.

```
MF_OneParamGeometry::trajectory( point: DirectPosition, p: Number ):
    MF_Trajectory
```

6.2.7 Operation – prism

The operation prism shall return an instance of GM_Object that is the prism formed by the union of all the leaves of this instance of MF_OneParamGeometry.

```
MF_OneParamGeometry::prism(  ): GM_Object
```

6.3 Type – MF_TemporalGeometry

6.3.1 Class semantics

MF_TemporalGeometry (Figure 6) is a specialization of MF_OneParamGeometry in which the parameter is time as expressed by TM_Coordinate. TM_Coordinate is specified in ISO 19108; it expresses time as a multiple of a single unit of measure such as year, day, or second.

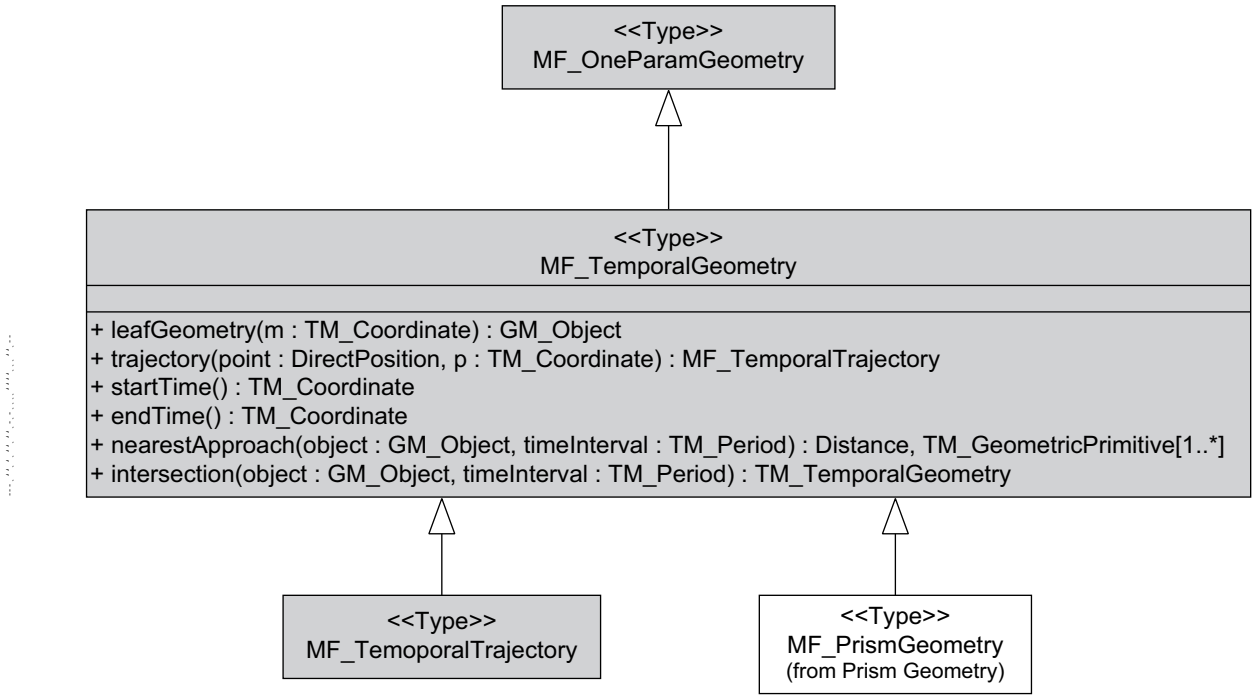

Figure 6 — Context Diagram: MF_TemporalGeometry

6.3.2 Inheritance from MF_OneParamGeometry

The type "MF_TemporalGeometry" inherits from the type "MF_OneParamGeometry". As such it shall implement all inherited attributes, operations and associations.

6.3.3 Operation – leafGeometry

The operation leafGeometry shall accept a time as input and return the instance of GM_Object that describes the leaf of the temporal geometry at that time.

```
MF_TemporalGeometry::leafGeometry( m: TM_Coordinate ): GM_Object
```

6.3.4 Operation – trajectory

The operation trajectory shall accept the position of a point on a leaf of the MF_TemporalGeometry at a specified time and return the temporal trajectory of that point.

```
MF_TemporalGeometry::trajectory( point: DirectPosition, p: TM_Coordinate ):
    MF_TemporalTrajectory
```

6.3.5 Operation – startTime

The operation startTime shall return the time at which the temporal geometry begins. This shall correspond to the value of "beginDomain".

```
MF_TemporalGeometry::startTime(  ): TM_Coordinate
```

6.3.6 Operation – endTime

The operation endTime shall return the time at which the temporal geometry ends. This shall correspond to the value of "endDomain".

```
MF_TemporalGeometry::endTime(  ): TM_Coordinate
```

6.3.7 Operation – nearestApproach

The operation "nearestApproach" shall return the distance and time of the nearest approach of the temporal geometry to any other geometric object. If the other geometric object is also a temporal geometry, then this operation is symmetric. The parameter "timeInterval" shall restrict the search to a particular period of time.

```
MF_TemporalGeometry::nearestApproach( object: GM_Object, timeInterval:
    TM_Period ): Distance, TM_GeometricPrimitive[1..*]
```

6.3.8 Operation – intersection

The operation "intersection" shall return the temporal geometry of the intersection of the temporal geometry to any other geometric object. If the other geometric object is also a temporal geometry, then this operation is symmetric. The parameter "timeInterval" shall restrict the search to a particular period of time.

```
MF_TemporalGeometry::intersection( object: GM_Object, timeInterval:
    TM_Period ): MF_TemporalGeometry
```

6.4 Type – MF_Trajectory

6.4.1 Class semantics

MF_Trajectory (Figure 7) describes a one-parameter geometry whose cross section is a point. The class is subject to the constraint that the position of the GM_Point returned by the *leafGeometry* operation equals the position returned by the *leaf* operation for the same value of the parameter *m*. This is expressed by the OCL:

```
{leafGeometry(m).position = leaf(m)}
```

The attributes of the class are derived using inherited operations as well as those specified for the class.

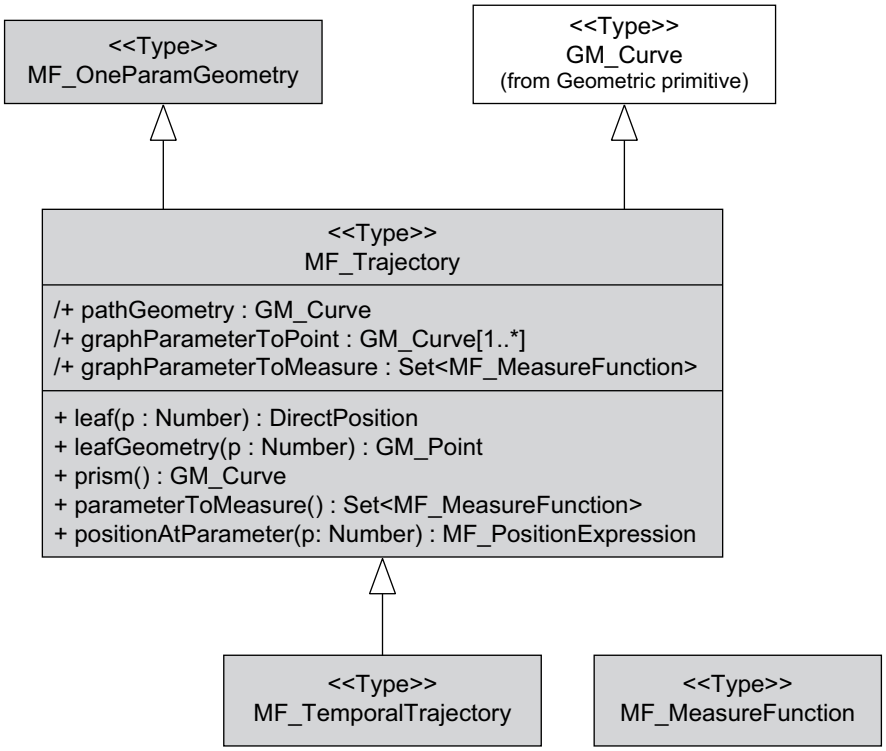

Figure 7 — Context Diagram: MF_Trajectory

6.4.2 Inheritance from MF_OneParamGeometry

The type "MF_Trajectory" inherits from the type "MF_OneParamGeometry". As such it shall implement all inherited attributes, operations and associations.

6.4.3 Inheritance from GM_Curve

The type "MF_Trajectory" inherits from the type "GM_Curve". As such it shall implement all inherited attributes, operations and associations. GM_Curve is described using both spatial and temporal coordinates.

6.4.4 Attribute – pathGeometry

The derived attribute "pathGeometry" of a trajectory is a spatial curve that is the projection of the trajectory over time. No relationship to time or path orientation remains. Repeated traversal of the same place at different times is not reflected in the resultant geometry.

```
MF_Trajectory::pathGeometry: GM_Curve
```

6.4.5 Attribute – graphParameterToPoint

The derived attribute "graphParameterToPoint" is the graph of the parameter to point function, expressed as a set of curves in a Euclidean space. Each curve is in 2D Euclidean space mapping the parameter of the trajectory to the "pathGeometry" curve's "parameterization by arc length" as defined in ISO 19107. Thus, the Euclidean space of the resultant graph curves is the set of points (p, s) where "p" is the parameter of the trajectory and "s" is the parameter of the underlying "pathGeometry".

```
MF_Trajectory::graphParameterToPoint: GM_Curve[1..*]
```

6.4.6 Attribute – graphParameterToMeasure

The derived attribute "graphParameterToMeasure" is the graph of the parameter to point function expressed as a set of curves, each part of an MF_MeasureFunction (6.8), in a Euclidean space. Each curve is in 2D Euclidean space mapping the parameter of the trajectory to the "pathGeometry" curves' linear reference measure as defined in ISO 19133.

```
MF_Trajectory::graphParameterToMeasure: Set<MF_MeasureFunction>
```

6.4.7 Operation – leaf

The operation "leaf" shall accept a value of the parameter as input and return the DirectPosition (from ISO 19107) through which the trajectory passes at that value of the trajectory parameter.

```
MF_Trajectory::leaf( p: Number ): DirectPosition
```

6.4.8 Operation – leafGeometry

The operation "leafGeometry" shall accept a value of the parameter as input and return the GM_Point that is at that position on the trajectory.

```
MF_Trajectory::leafGeometry( p: Number ): GM_Point
```

6.4.9 Operation – prism

The operation "prism" shall return the instance of GM_Curve that corresponds to the geometry of the trajectory.

```
MF_Trajectory::prism(  ): GM_Curve
```

6.4.10 Operation – parameterToMeasure

The operation "parameterToMeasure" shall return an instance of GM_Curve that describes the relationship between the parameter and measure of one or more specified LRS's along the trajectory, using the type MF_MeasureFunction (6.8). Each GM_Curve's coordinate reference system is the Cartesian product of the trajectory parameter and the measure associated to its containing MF_MeasureFunction.

```
MF_Trajectory::parameterToMeasure(  ): Set<MF_MeasureFunction>
```

6.4.11 Operation – positionAtParameter

The operation "positionAtParameter" shall return an instance of MF_PositionExpression that describes the position of the moving feature along the trajectory. The use of MF_PositionExpression allows the trajectory to following existing geometry (such as a road centreline) and use offset to specify variations in any direction. This allows the position expression to carry information about lanes in the case of land vehicles, or a watercraft's attitude or drift in the case of marine vehicles.

```
MF_Trajectory::positionAtParameter( p: Number ): MF_PositionExpression
```

6.5 Type – MF_TemporalTrajectory

6.5.1 Class semantics

An instance of MF_TemporalTrajectory (Figure 8) is a trajectory whose parameter is time.

6.5.2 Inheritance from MF_Trajectory

The type "MF_TemporalTrajectory" inherits from the type "MF_Trajectory". As such it shall implement all inherited attributes, operations and associations.

6.5.3 Inheritance from MF_TemporalGeometry

The type "MF_TemporalTrajectory" inherits from the type "MF_TemporalGeometry". As such it shall implement all inherited attributes, operations and associations.

6.5.4 Attribute – beginDomain

The attribute "beginDomain" shall contain the value of the time parameter at the start of the domain of the one-parameter geometry.

```
MF_TemporalTrajectory::beginDomain: TM_Coordinate
```

NOTE This overrides the same attribute in MF_OneParamGeometry (6.2.3). Normally this would require TM_Coordinate to be a subtype of Number, but TM_Coordinate can be expressed as a measure (which is a number, with either an implicit or explicit unit). The specification of a particular temporal coordinate system allows a purely numeric representation for TM_Coordinate.

6.5.5 Attribute – endDomain

The attribute "endDomain" shall contain the value of the time parameter at the end of the domain for the trajectory.

```
MF_TemporalTrajectory::endDomain: TM_Coordinate
```

NOTE This overrides the same attribute in MF_OneParamGeometry (6.2.4).

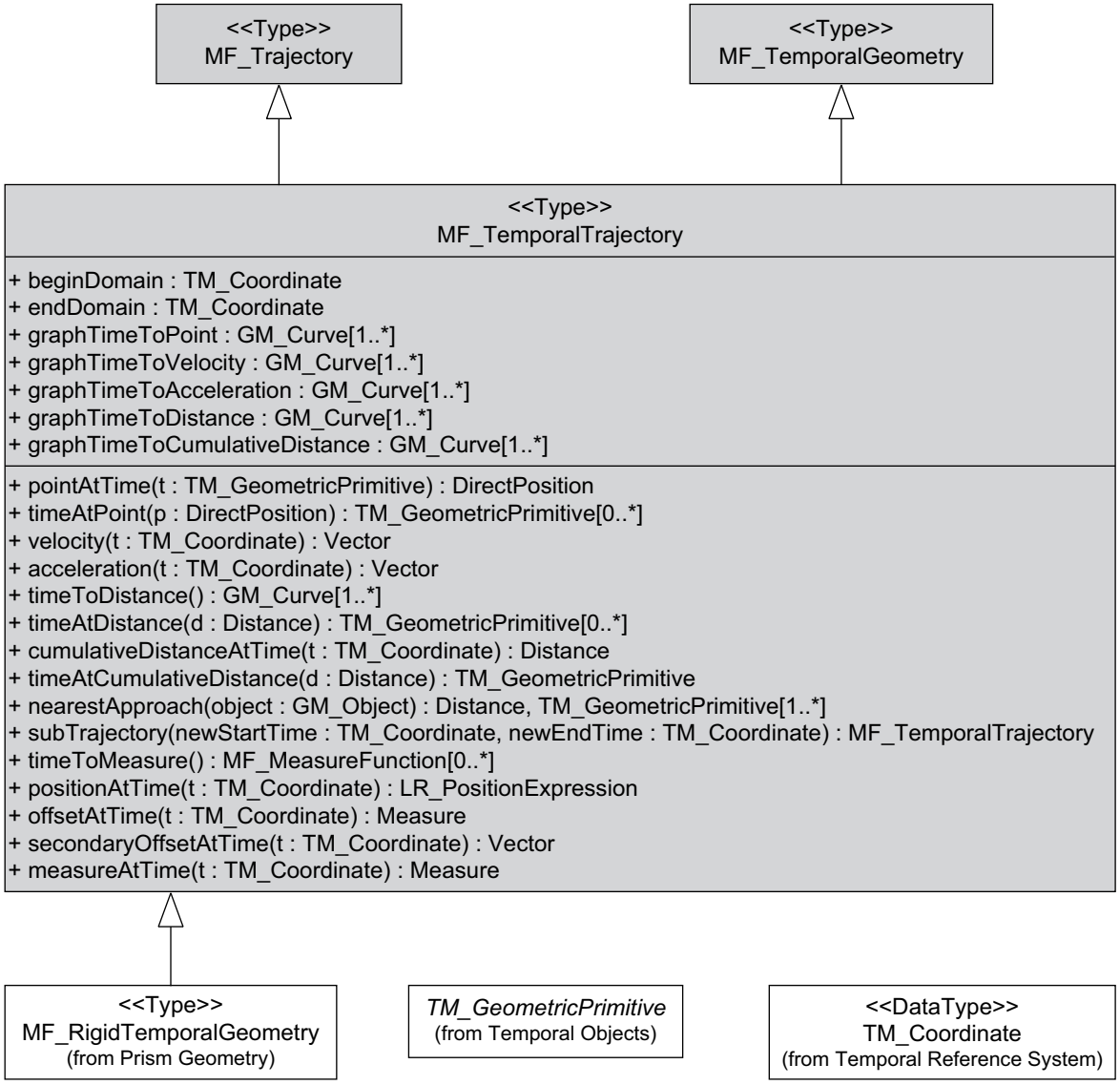

Figure 8 — Context Diagram: MF_TemporalTrajectory

6.5.6 Attribute – graphTimeToPoint

The attribute "graphTimeToPoint" is the graph of the time to distance function, expressed as a set of curves in a Euclidean space. Each curve is in a 2D Euclidean space mapping time along the trajectory to the curve's "parameterization by arc length" as defined in ISO 19107. Since time is the parameter of a temporal trajectory, this is the same as "graphParameterToPoint" (6.4.5).

```
MF_TemporalTrajectory::graphTimeToPoint: GM_Curve[1..*] =
    graphParameterToPoint
```

6.5.7 Attribute – graphTimeToVelocity

The attribute "graphTimeToVelocity" is the graph of the time to velocity function expressed as a set of curves in a Euclidean space. Each curve is in a (n+1)−D (where n is the coordinate dimension of the trajectory's velocity vector) Euclidean space mapping time along the trajectory to a velocity vector. The distance measures to be associated to the velocity vector shall be the same as those associated to the trajectory coordinate reference system.

```
MF_TemporalTrajectory::graphTimeToVelocity: GM_Curve[1..*]
```

6.5.8 Attribute – graphTimeToAcceleration

The attribute "graphTimeToAcceleration" is the graph of the time to acceleration function, expressed as a set of curves in a Euclidean space. Each curve is in a (n+1)−D (where n is the coordinate dimension of the trajectory's velocity vector) Euclidean space mapping time along the trajectory to an acceleration vector. The distance measures to be associated to the acceleration vector shall be the same as those associated to the trajectory coordinate reference system.

```
MF_TemporalTrajectory::graphTimeToAcceleration: GM_Curve[1..*]
```

6.5.9 Attribute – graphTimeToDistance

The attribute "graphTimeToDistance" is the graph of the time to distance, expressed as a set of curves in a Euclidean space. Each curve is in a 2D Euclidean space mapping time along the trajectory to distance along the trajectory's "pathGeometry".

```
MF_TemporalTrajectory::graphTimeToDistance: GM_Curve[1..*]
```

6.5.10 Attribute – graphTimeToCumulativeDistance

The attribute "graphTimeToCumulativeDistance" is the graph of the time to cumulative distance, expressed as a set of curves in a Euclidean space. Each curve is in a 2D Euclidean space mapping time along the trajectory to distance along the trajectory's "pathGeometry".

```
MF_TemporalTrajectory::graphTimeToCumulativeDistance: GM_Curve[1..*]
```

6.5.11 Operation – pointAtTime

The operation "pointAtTime" shall accept a time in the domain of the trajectory as input, and shall return the direct position of the trajectory at that time.

```
MF_TemporalTrajectory::pointAtTime( t: TM_GeometricPrimitive ): DirectPosition
```

6.5.12 Operation – timeAtPoint

The operation "timeAtPoint" shall accept a DirectPosition as input and return the set of times at which the trajectory passes through that DirectPosition. Each value in the set returned shall be an instance of one of the subclasses of TM_GeometricPrimitive, either a TM_Instant or a TM_Period. The use of TM_Period allows for the description of periods of time during which the moving object remains stationary. If the point is not in the prism of the trajectory, then the operation returns an empty set.

```
MF_TemporalTrajectory::timeAtPoint( p: DirectPosition ):
    Set<TM_GeometricPrimitive>
```

6.5.13 Operation – velocity

The operation "velocity" shall accept a time (TM_Coordinate) as input and return the velocity as a vector at that time.

```
MF_TemporalTrajectory::velocity( t: TM_Coordinate ): Vector
```

6.5.14 Operation – acceleration

The operation "acceleration" shall accept a time (TM_Coordinate) as input and return the acceleration as a vector at that time.

```
MF_TemporalTrajectory::acceleration( t: TM_Coordinate ): Vector
```

6.5.15 Operation – timeToDistance

The operation "timeToDistance" returns a graph of the time to distance function as a set of curves in the Euclidean space consisting of coordinate pairs of time, distance.

```
MF_TemporalTrajectory::timeToDistance(  ): GM_Curve[1..*]
```

EXAMPLE In Figure 9 the curve at "A" represents the time to distance function for an object that accelerates from t_0 to t_1, moves at constant velocity from t_1 until t_2, and then decelerates to a stop at t_3. The curve at "B" represents the time to distance function for an object that travels 3 times around a closed trajectory at constant velocity; the solid line represents the distance from the origin of the trajectory, while the dashed line represents the cumulative distance travelled.

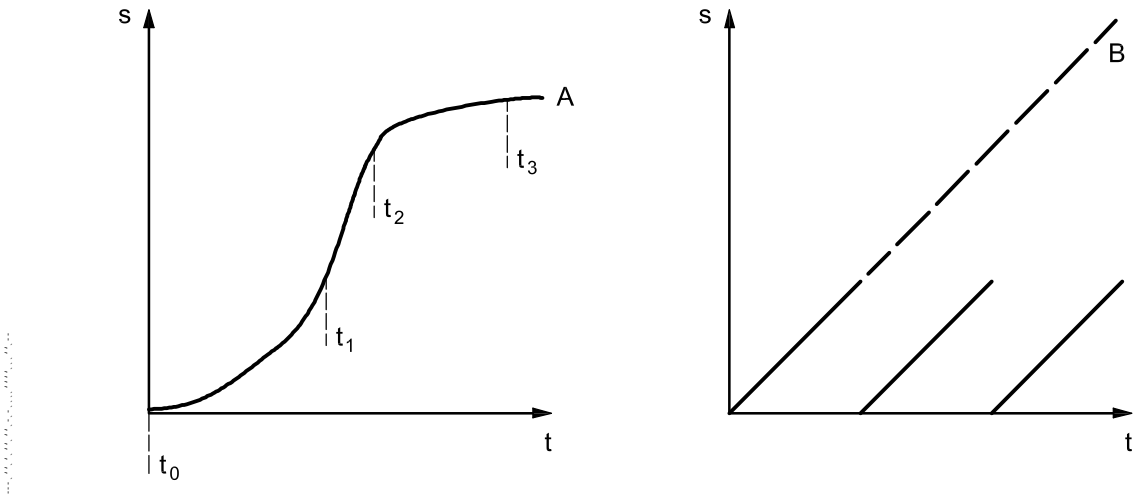

Figure 9 — Examples of time to distance curves

6.5.16 Operation – timeAtDistance

The operation "timeAtDistance" shall return an array of TM_GeometricPrimitive that lists in ascending order the time or times a particular point (determined by the Set<Distance> in the trajectory's GM_GenericCurve::paramForPoint(p:DirectPosition) : Set<Distance>, DirectPosition) is reached.

```
MF_TemporalTrajectory:: timeAtDistance(d : Distance) :
    TM_GeometricPrimitive[0..*]
```

For a point p:DirectPosition (in the global CRS) the trajectory passes through at time t, then for each d a distance in the trajectory's paramForPoint(p:DirectPosition).Set<Distance>, timeAtDistance(d) contains t in the union of its primitives.

6.5.17 Operation – cumulativeDistanceAtTime

The operation "cumulativeDistanceAtTime" shall accept a time as input and return the cumulative distance travelled (including all movements forward and retrograde as positive travel distance) from the beginning of the trajectory at that time "t".

```
MF_TemporalTrajectory::cumulativeDistanceAtTime( t: TM_Coordinate ): Distance
```

6.5.18 Operation – timeAtCumulativeDistance

The operation "timeAtCumulativeDistance" shall accept a distance as input and return the time at which the trajectory's total length (including all movements forward and retrograde as positive travel distance) reaches that cumulative travel distance.

```
MF_TemporalTrajectory::timeAtCumulativeDistance( d: Distance ):
    TM_GeometricPrimitive
```

6.5.19 Operation – nearestApproach

The operation "nearestApproach" shall return the time and distance of the nearest approach of this trajectory to another GM_Object. If the other GM_Object is also a temporal geometry, the distance returned will be the distance between the leaves of the two objects at the returned time.

```
MF_TemporalGeometry::nearestApproach( object: GM_Object ): Distance,
    TM_GeometricPrimitive[1..*]
```

6.5.20 Operation – subTrajectory

The operation "subTrajectory" shall accept two times in the domain of the trajectory and return a trajectory that is a subset of the given trajectory for the specified time interval. The operation allows all temporal trajectory operations to be restricted to particular time periods.

```
MF_TemporalTrajectory::subTrajectory( newStartTime: TM_Coordinate, newEndTime:
    TM_Coordinate ): MF_TemporalTrajectory
```

6.5.21 Operation – timeToMeasure

The operation "timeToMeasure" shall return a set of MF_MeasureFunctions that associate time to position within a linear reference system.

```
MF_TemporalTrajectory::timeToMeasure(  ): Set<MF_MeasureFunction>
```

6.5.22 Operation – positionAtTime

The operation "positionAtTime" shall accept a time in the domain of the trajectory and return the position of the moving feature on the trajectory at that time, expressed as a linear reference system position.

```
MF_TemporalTrajectory::positionAtTime( t: TM_Coordinate ):
    LR_PositionExpression
```

6.5.23 Operation – offsetAtTime

The operation "offsetAtTime" shall accept a time in the domain of the trajectory and return the offset position on the trajectory at that time, expressed as the offset measure for a linear reference system.

```
MF_TemporalTrajectory::offsetAtTime( t: TM_Coordinate ): Measure
```

6.5.24 Operation – secondaryOffsetAtTime

The operation "secondaryOffsetAtTime" shall accept a time in the domain of the trajectory and return the secondary offset position of the trajectory at that time, expressed as vector of measures.

```
MF_TemporalTrajectory::secondaryOffsetAtTime( t: TM_coordinate ): Vector
```

6.5.25 Operation – measureAtTime

The operation "measureAtTime" shall accept a time in the domain of the trajectory and return the position of the trajectory at that time, expressed as a linear reference system measure.

```
MF_TemporalTrajectory::measureAtTime( t: TM_Coordinate ): Measure
```

6.6 Class – MF_PositionExpression

6.6.1 Class semantics

The type "MF_PositionExpression" (Figure 10) extends LR_PositionExpression to allow for use of existing geometry for moving features that may vary from the "default course" by drifting around the nominal route by potentially complex, temporally varying offsets.

Figure 10 — Context Diagram: MF_PositionExpression

6.6.2 Inheritance from LR_PositionExpression

The type "MF_PositionExpression" inherits from the type "LR_PositionExpression". As such it shall implement all attributes, operations and associations inherited from that type as specified in ISO 19133.

6.6.3 Association Role – secondaryOffset

The optional association role "secondaryOffset" allows the MF_PositionExpression instances to express complex offsets in any dimension, as opposed to the offset of normal LRS's which are usually restricted to left-right offsets in a 2-dimensional space.

```
MF_PositionExpression::secondaryOffset[0..*]: MF_SecondaryOffset
```

6.7 Type – MF_SecondaryOffset

6.7.1 Class semantics

The type "MF_SecondaryOffset" (Figure 11) describes an offset in any direction from the geometry identified as the "default course" of the moving object.

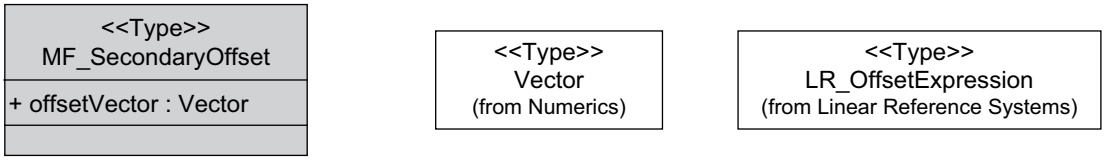

Figure 11 — Context Diagram: MF_SecondaryOffset

6.7.2 Attribute – offsetVector

The attribute "offsetVector" describes the magnitude and direction of the offset.

```
MF_SecondaryOffset::offsetVector: Vector
```

6.8 Type – MF_MeasureFunction

6.8.1 Class semantics

The type "MF_MeasureFunction" (Figure 12) is used by a temporal trajectory to express position of a moving feature by use of a linear reference system, as defined in 19133 and extended by MF_PositionExpression in 6.6.

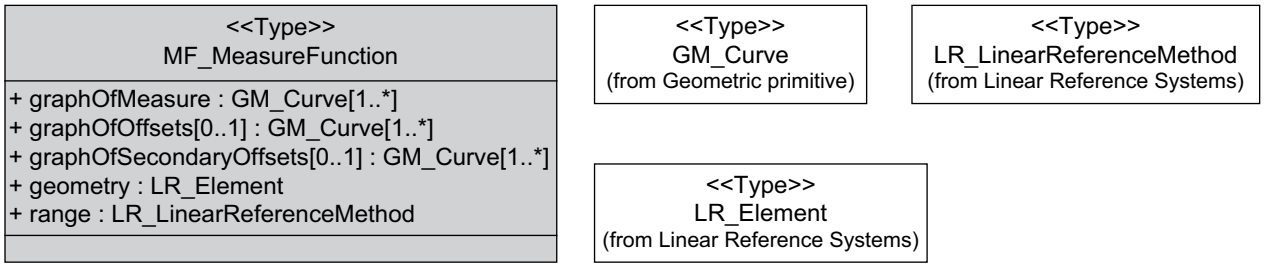

Figure 12 — Context Diagram: MF_MeasureFunction

6.8.2 Attribute – graphOfMeasure

The attribute "graphOfMeasure" is the graph of the time to measure as a set of curves in a Euclidean space. Each curve is in a 2D Euclidean space mapping time along the trajectory to measure along the MF_MeasureFunction "pathGeometry".

```
MF_MeasureFunction::graphOfMeasure: GM_Curve[1..*]
```

6.8.3 Optional attribute – graphOfOffset

The optional attribute "graphOfOffset" is a set of curves that represent the left-right deviations of the trajectory from the default course.

```
MF_MeasureFunction::graphOfOffset[0..1]: GM_Curve[1..*]
```

6.8.4 Optional attribute – graphOfSecondaryOffsets

The optional attribute "graphOfSecondaryOffsets is a set of curves that represent the deviations in any direction of the trajectory from the default course.

```
MF_MeasureFunction::graphOfSecondaryOffsets[0..1]: GM_Curve[1..*]
```

EXAMPLE The planned route for an aircraft might be a geodesic curve at constant elevation. Because of air turbulence, the actual path travelled by the aircraft will deviate at random from the planned route. The graphOfSecondaryOffsets will record those deviations.

6.8.5 Attribute – geometry

The attribute "geometry" contains the underlying curvilinear geometry upon which the measures in the linear reference system are taken.

```
MF_MeasureFunction::geometry: LR_Element
```

6.8.6 Attribute – range

The attribute "range" contains the linear reference system used by a temporal trajectory to express position of a moving feature.

```
MF_MeasureFunction::range: LR_LinearReferenceMethod
```

7 Package – Prism Geometry

7.1 Package structure

The package Prism Geometry (Figure 13) contains five types used to describe the prism of a moving geometric object. MF_RigidTemporalGeometry (Figure 19) specializes MF_PrismGeometry, which in turn specializes MF_TemporalGeometry for the case of an object that moves without deformation. The remaining classes support description of the possible rotation of such an object.

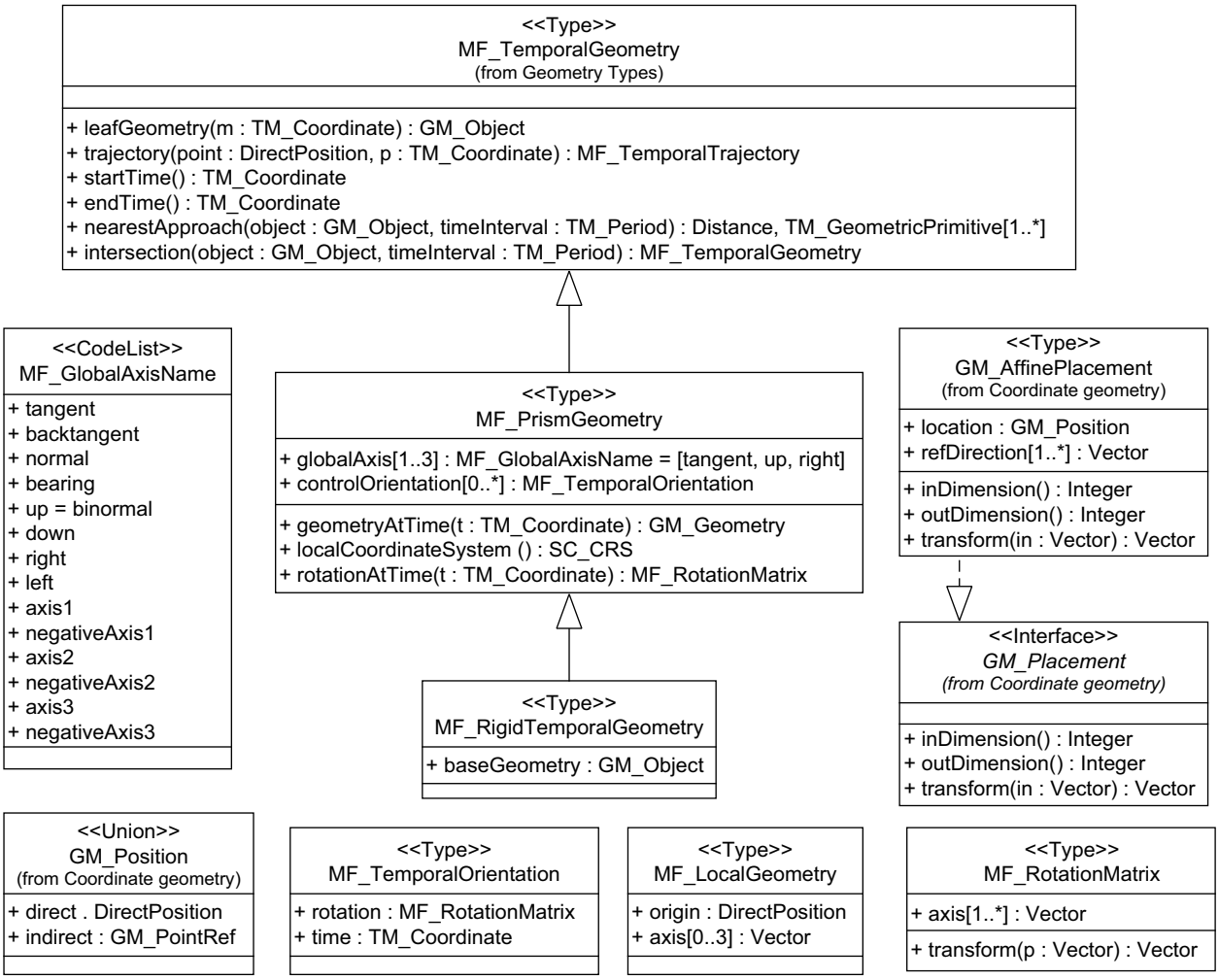

Figure 13 — Classes of the Prism Geometry package

7.2 CodeList – MF_GlobalAxisName

7.2.1 Class semantics

The code list "MF_GlobalAxisName" (Figure14) names the usual global axes. They are normally either in terms of the moving frame of the curve, or in terms of the external CRS in which the geometry of the curve is defined (Figure 15). The moving frame of the curve is a right-handed set of independent vectors that are a basis for a local Euclidean vector space equal to the tangent space of the CRS (3D) at the points along the curve. It shall consist of a tangent to the curve, a normal vector pointing to the left of this tangent, and their cross product, which is a "near" upward normal to the curve. If the original CRS is 2D, this third vector is the local up of the underlying CRS surface. If only the first axis is given, it must be horizontal, with the third axis vector shall be assumed to be the local "up" and the second axis vector shall be the cross product of the other two (v2 = v3 x v1). The external CRS is usually an earth-fixed geographic CRS, but this International Standard does not require that it be such. Steerable objects may work best with the moving frame: front (which usually maps to the tangent), up, and right. Non-steered objects may work best with the tangents to the coordinate curves of the external CRS.

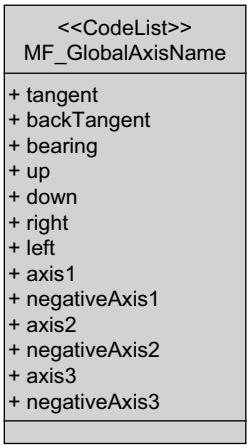

Figure 14 — Context Diagram: MF_GlobalAxisName

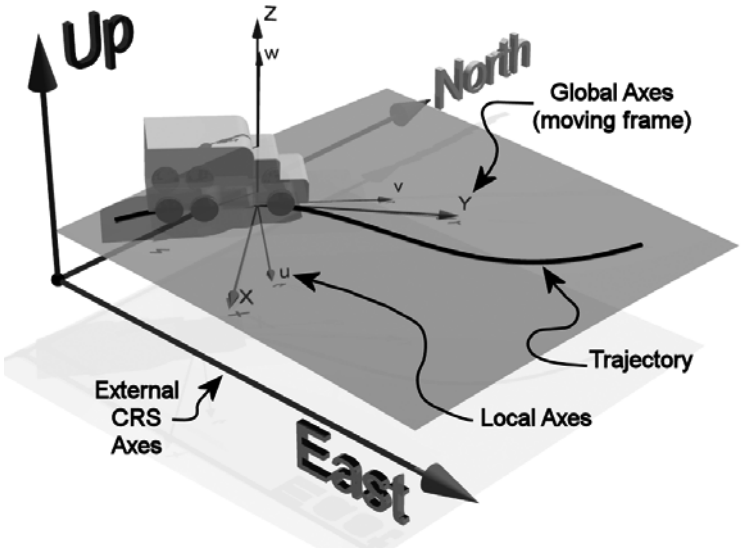

Figure 15 — Global and local axes

7.2.2 Attribute – tangent

The "tangent" vector represents the geometric unit tangent to the curve. It always points in the forward direction of the curve's underlying "parameterization by length" as defined in ISO 19107. This is not the tangent to the motion of the object. If the object backtracks the curve, the forward unit tangent of the object will be the negative of the curve's tangent.

```
MF_GlobalAxisName::tangent
```

7.2.3 Attribute – backTangent

The "backTangent" vector is the negative of the tangent to the curve.

```
MF_GlobalAxisName::backtangent
```

7.2.4 Attribute – bearing

The bearing is the projection of the tangent to the curve onto the horizontal plane of the external CRS. If the curve is perfectly vertical, the bearing is undefined. Curves that go perfectly vertical should not use bearing.

```
MF_GlobalAxisName::bearing
```

7.2.5 Attribute – up

This "up" vector is the local up as in elevation. The vector up is the cross product of the tangent and left. It represents the up for the moving feature, which may be the up for the external coordinate reference system if the curve is locally on a surface of constant elevation.

```
MF_GlobalAxisName::up = binormal
```

7.2.6 Attribute – down

This "down" vector is the negative of "up".

```
MF_GlobalAxisName::down
```

7.2.7 Attribute – right

This "right" vector is the cross product of "tangent" and "up" and the negative of "left".

```
MF_GlobalAxisName::right
```

7.2.8 Attribute – left

The "left" vector is the unit vector point to the left of the curve. As such the left is the cross product of "up" and the "tangent". "Left" is the negative of "right".

```
MF_GlobalAxisName::left
```

7.2.9 Attribute – axis1

The vector "axis1" is the first coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by increasing the first coordinate while keeping the others constant.

```
MF_GlobalAxisName::axis1
```

7.2.10 Attribute – negativeAxis1

The vector "negativeAxis1" is negative of the first coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by decreasing the first coordinate while keeping the others constant.

```
MF_GlobalAxisName::negativeAxis1
```

7.2.11 Attribute – axis2

The vector "axis2" is the second coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by increasing the second coordinate while keeping the others constant.

```
MF_GlobalAxisName::axis2
```

7.2.12 Attribute – negativeAxis2

The vector "negativeAxis2" is negative of the second coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by decreasing the second coordinate while keeping the others constant.

```
MF_GlobalAxisName::negativeAxis2
```

7.2.13 Attribute – axis3

The vector "axis3" is the third coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by increasing the third coordinate while keeping the others constant. If the CRS is 2D, then this is the upward unit vector perpendicular to the defining surface of the 2D CRS. It should be used when placing 3D moving feature icons on a 2D map.

```
MF_GlobalAxisName::axis3
```

7.2.14 Attribute – negativeAxis3

The vector "negativeAxis3" is negative of the third coordinate axis of the external CRS. In other words, it is the unit tangent to the curve created by decreasing the third coordinate while keeping the others constant. If the CRS is 2D, then this is the downward unit vector perpendicular to the defining surface of the 2D CRS. It should be used when placing 3D moving feature icons on a 2D map.

```
MF_GlobalAxisName::negativeAxis3
```

7.3 Type – MF_LocalGeometry

7.3.1 Class semantics

The type "MF_LocalGeometry" (Figure 16) is a geometric object in a design coordinate reference system (usually 3D). Instances of this type are used to define the local geometry of the moving object (Figure 17). For "iconized" representations, a 2D design coordinate reference system might be used. The design coordinate reference system is accessible through the Coordinate Reference System association inherited from GM_Object.

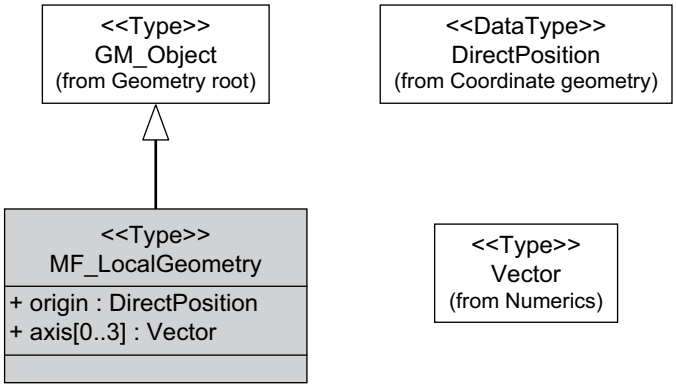

Figure 16 — Context Diagram: MF_LocalGeometry

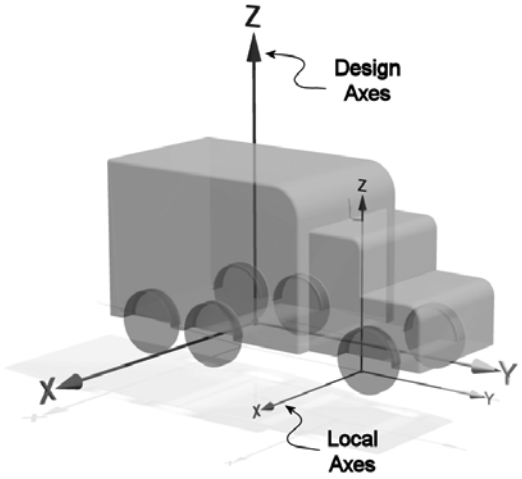

Figure 17 — Local geometry

7.3.2 Inheritance from GM_Object

The type "MF_LocalGeometry" inherits from the type "GM_Object". As such it shall implement all inherited attributes, operations and associations.

7.3.3 Attribute – origin

The attribute "origin" describes a point on the local geometry that will act as a placement point into geographic space. This point is chosen by the application usually for ease of calculations. Common choices for an origin might be centre of gravity or centre of footprint. The returned direct position is in the design coordinate reference system of the MF_LocalGeometry.

```
MF_LocalGeometry::origin: DirectPosition
```

7.3.4 Optional attribute – axis

The attribute "axis" lists up to three local axes that will be used in describing the local object's orientation (and scale) when it is embedded in geographic space. These axes shall be specified as vectors centred on the "origin" as described in 7.3.3 and are referenced to the design coordinate reference system of the MF_LocalGeometry. When not given, the axis array shall be assumed to be the coordinate axis of the design coordinate reference system.

```
MF_LocalGeometry::axis[0..3]: Vector
```

7.4 Type – MF_PrismGeometry

7.4.1 Class semantics

The type "MF_PrismGeometry" (Figure 18) represents the movement of an object through geographic space. This International Standard only considers the case where the operation "geometryAtTime" is a constant (that is, where the object's basic shape is immutable over time).

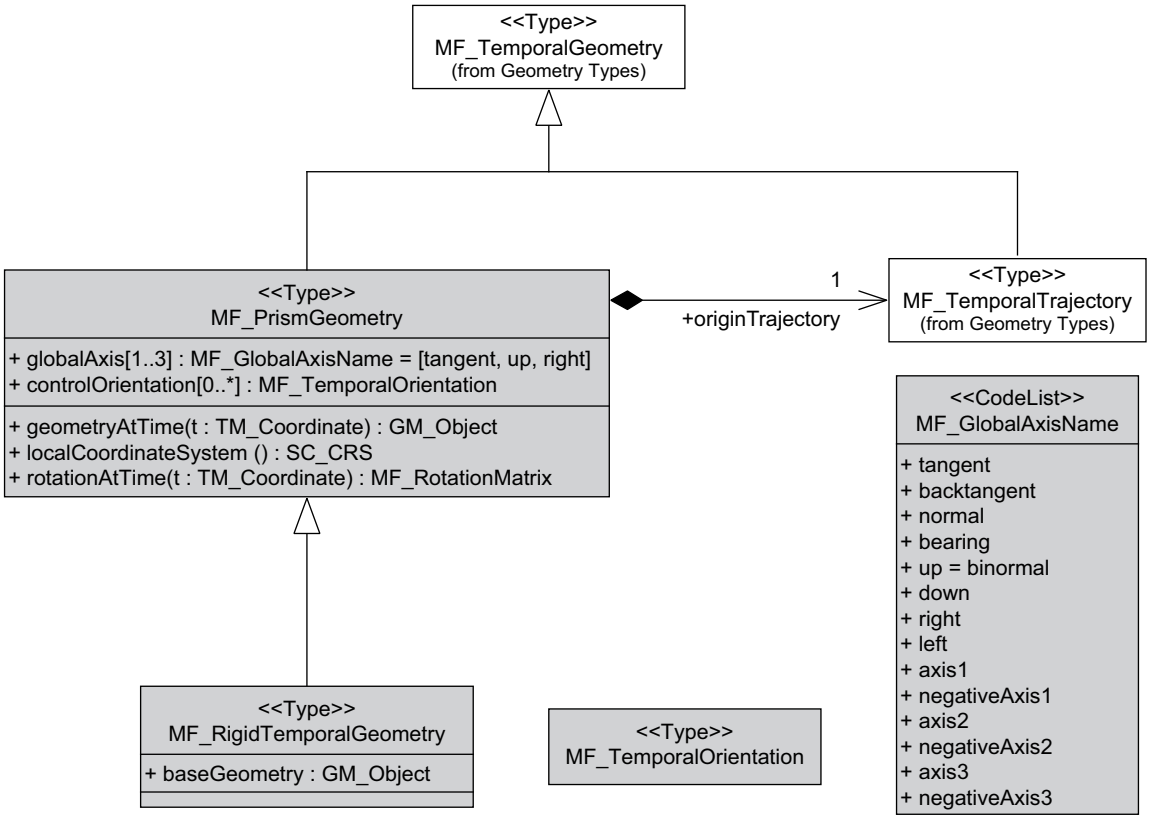

Figure 18 — Context Diagram: MF_PrismGeometry

7.4.2 Inheritance from MF_TemporalGeometry

The type "MF_PrismGeometry" inherits from the type "MF_TemporalGeometry". As such it shall implement all inherited attributes, operations and associations.

7.4.3 Association Role – originTrajectory

The association role "originTrajectory" is the trajectory curve of the origin of the local geometry (7.3.3).

```
MF_PrismGeometry::originTrajectory: MF_TemporalTrajectory
```

7.4.4 Attribute – globalAxis

The attribute "globalAxis" is the global, possibly moving, geometric coordinate frame (7.2) in which the rotation and scaling of the base geometry is defined. The default values are the axes of the moving frame of the curve.

```
MF_PrismGeometry::globalAxis[1..3]: MF_GlobalAxisName =
    [tangent, up, right]
```

7.4.5 Optional attribute – controlOrientation

The attribute "controlOrientation" array contains some number of rotational positions distributed along the curve by time. A method of interpolation used between these orientations is described in Annex C.

```
MF_PrismGeometry::controlOrientation[0..*]: MF_TemporalOrientation
```

7.4.6 Operation – geometryAtTime

The operation "geometryAtTime" shall accept a time in the domain of the prism geometry and return the geometry of the moving feature, as it is at a given time in the global coordinate reference system. This shape might be a realistic rendition of the object, or it may be an iconized rendition of the type of object, as needed by the application. For example, in a simulation a truck might be represented as an icon as opposed to a photorealistic rendition. This allows the application to use the local geometry to convey information such as certainty of identification or feature status through the use of appropriate icons and other portrayal parameters.

```
MF_PrismGeometry::geometryAtTime( t: TM_Coordinate ): GM_Object
```

7.4.7 Operation – localCoordinateSystem

The operation "localCoordinateSystem" shall return a SC_CRS for the design coordinate reference system in which the moving feature's shape is defined.

```
MF_PrismGeometry::localCoordinateSystem(  ): SC_CRS
```

7.4.8 Operation – rotationAtTime

The operation "rotationAtTime" shall accept a time in the domain of the prism geometry and return the rotation matrix that embeds the local geometry into geographic space at a given time (TM_Coordinate). The vectors of the rotation matrix allow the feature to be aligned and scaled as appropriate to the vectors of the global "map" coordinate reference system. Because scale may change in complex ways as a feature moves with respect to a projected coordinate reference system, the rotation matrix may also contain scale factors. These should be used to prevent apparent deformation of the moving feature. The operation shall return an error message if the input time is not within the domain.

```
MF_PrismGeometry::rotationAtTime( t: TM_Coordinate ): MF_RotationMatrix
```

7.5 Type – MF_RigidTemporalGeometry

7.5.1 Class semantics

The type "MF_RigidTemporalGeometry" (Figure 19) describes the motion of a rigid body, one that may be translated or rotated, but which does not change shape – it remains congruent to its base representation. In OCL this would say:

```
{baseGeometry = geometryAtTime(beginDomain)}
```

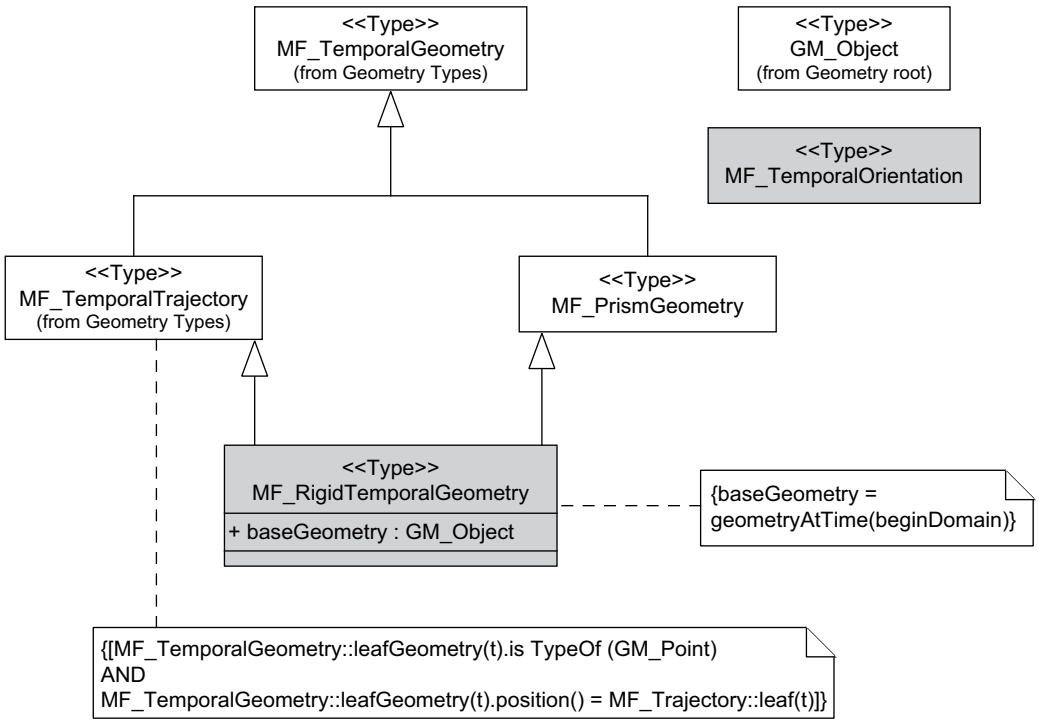

Figure 19 — Context Diagram: MF_RigidTemporalGeometry

7.5.2 Inheritance from MF_PrismGeometry

The type "MF_RigidTemporalGeometry" inherits from the type "MF_PrismGeometry". As such it shall implement all inherited attributes, operations and associations (7.4).

7.5.3 Inheritance from MF_TemporalTrajectory

The type "MF_RigidTemporalGeometry" inherits from the type "MF_TemporalTrajectory". As such it shall implement all inherited attributes, operations and associations (6.5).

7.5.4 Attribute – baseGeometry

The attribute "baseGeometry" is the geometry of the moving object in a local rectangular coordinate reference system based on the axis of the object. The object should have a natural up, front and right (cross product of up and front). If the representation to be used is not "centred" on its origin, the application should use an MF_LocalGeometry (7.3) subclass to create a local origin.

```
MF_RigidTemporalGeometry::baseGeometry: GM_Object
```

7.6 Type – MF_RotationMatrix

7.6.1 Class semantics

The type "MF_RotationMatrix" (Figure 20) is designed to capture the changing orientation of the local frame of the object in terms of the global frame. The frames are defined in the MF_PrismGeometry (7.4) type, by the "localCoordinateSystem" operation and the "globalAxis" attributes. In this International Standard, this matrix includes both rotation and scale factors to compensate for the differences in the scale of the local engineering coordinate reference system and the global geographic coordinate reference system.

Figure 20 — Context Diagram: MF_RotationMatrix

7.6.2 Attribute – axis

The attribute "axis" shall contain the coordinates of the localAxis array in terms of the globalAxis array as an orthogonal matrix. The first two vectors should be orthogonal (have a zero vector dot product). The third axis should be the cross product of the first two. If not all axes are present, these facts or some other default mechanism should be able to calculate them. The dimension of the matrix is limited to three in most moving object applications, but the type can be extended to higher dimension if other data (velocities for example) are captured.

```
MF_RotationMatrix::axis[1..*]:Vector
```

7.7 Type – MF_TemporalOrientation

7.7.1 Class semantics

The type "MF_TemporalOrientation" (Figure 21) is designed to capture the rotational motion of the object as it passes along its trajectory. As such, it contains information on scale and orientation at a particular time. The "controlOrientation" (7.4.5) array in MF_PrismGeometry aggregates these orientations for interpolation by the application.

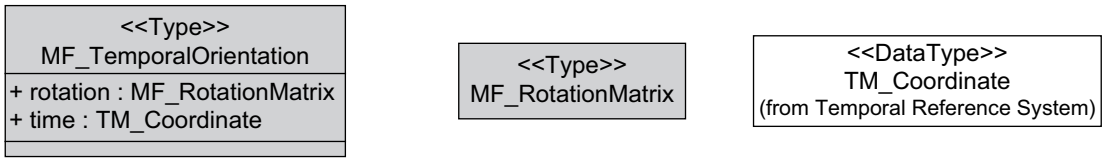

Figure 21 — Context Diagram: MF_TemporalOrientation

7.7.2 Attribute – rotation

The attribute "rotation" shall contain a rotational, and potentially scaling, matrix or its equivalent for a particular time.

```
MF_TemporalOrientation::rotation: MF_RotationMatrix
```

7.7.3 Attribute – time

The attribute "time" shall contain the time at which the rotation matrix is valid.

```
MF_TemporalOrientation::time: TM_Coordinate
```

8 Moving features in application schemas

8.1 Introduction

ISO 19109 specifies rules for developing schemas to specify the feature types, characteristics, and relationships needed to support particular applications. Application schemas are to be built upon a framework of concepts specified in base standards such as this International Standard. ISO 19109 was published before

work began on this International Standards, so it provides no specific rules for specifying moving features in application schemas. This clause states some requirements for doing so.

8.2 Representing the spatial characteristics of moving features

ISO 19109 requires that the spatial characteristics of a feature be represented by a GM_Object or a TP_Object used as a data type for an appropriately defined feature attribute. The principal types specified in this International Standard are subclassed from GM_Object (Figure 3). An application schema that includes moving feature types shall use realizations of the types specified in this International Standard to represent the spatial characteristics of such feature types.

8.3 Associations of moving features

Associations may influence or depend upon movement of features. Such associations may be modelled at the feature level, or at the level of the class that represents the spatial characteristics of the moving feature.

Movement of a feature may be constrained by its associations with other feature instances or feature types. In such cases, the application schema shall specify an association between the trajectory of the moving feature and the GM_Object that represents the spatial characteristics of the constraining feature and specify an appropriate constraint.

EXAMPLE 1 Motion of a vehicle type might be constrained to a road network. This could be modelled as an association between the trajectory of the vehicle and the GM_Complex representing the road network, with a constraint stating that the GM_Curve underlying the trajectory shall equal a GM_Curve (most likely a composite curve containing links and partial links in the road network, representable by an NT_Route, see ISO 19133) within the road network (an implementation of 19133:NT_Network).

Moving features may participate in associations with other moving features as well as with immobile features. In this case, an application schema shall specify an association between the feature types as an association class with a temporal attribute that contains the duration of the association.

EXAMPLE 2 Trucks in a convoy participate in a set of associations with the other trucks in that convoy.

8.4 Operations of moving features

Application schemas will often specify operations for particular moving feature types that involve the spatial and temporal relationships of the moving feature to other features. Such operations shall make use of the results of the basic geometric operations specified in this International Standard and in ISO 19107 and ISO 19108. Documentation of such feature operations shall include a description of the dependencies upon the basic geometric operations.

EXAMPLE Consider an aircraft route planning application that has a requirement to determine the times at which an aircraft following a planned route will be under the control of different air traffic control centres. If the route is modelled as a realization of MF_TemporalTrajectory and air traffic control zones are modelled as realizations of GM_Solid, an operation defined for the planned route might make use of the 'union' operation specified in ISO 19107 to determine the sub-trajectory of the route that lies within a given air traffic control zone, and then apply the 'startTime' and 'endTime' operations specified in this International Standard to arrive at the range of time during which the aircraft is within that air traffic control zone.

Annex A
(normative)

Abstract test suite

A.1 Application schemas for data transfer

A.1.1 Transfer of trajectory data

a) Test Purpose: Verify that an application schema for transfer of the trajectories of moving features satisfies the minimum requirements for specifying the trajectory of each moving feature.

b) Test Method: Inspect the application schema to ensure that the specification of each moving feature type includes an association to a realization of MF_TemporalTrajectory with the attributes beginDomain, endDomain, and graphTimeToDistance as well as the attribute pathGeometry inherited from MF_Trajectory. If the trajectory is constrained to follow a linear feature that has a specified LRS, ensure that the realization of MF_TemporalTrajectory also includes the attribute graphParameterToMeasure inherited from MF_Trajectory.

c) Reference: 6.4, 6.5, 6.8

d) Test Type: Capability

A.1.2 Transfer of prism geometry data

a) Test Purpose: Verify that an application schema for transfer of the prism geometry of moving features satisfies the minimum requirements for specifying the prism of each moving feature.

b) Test Method: Inspect the application schema to ensure that the specification of each moving feature type satisfies the requirements of A.1.1 and also includes an association to a realization of MF_RigidTemporalGeometry with the attribute baseGeometry as well as the attributes globalAxis and controlOrientation inherited from MF_PrismGeometry.

c) Reference: 7.4, 7.5, 7.6, 7.7

d) Test Type: Capability

A.2 Application schemas for data with operations

A.2.1 Data with operations on trajectories

a) Test Purpose: Verify that an application schema that supports operations on the trajectories of moving features satisfies the minimum requirements for specifying the trajectory of each moving feature.

b) Test Method: Inspect the application schema to ensure that the specification of each moving feature type includes an association to a realization of MF_TemporalTrajectory that includes all attributes and associations and supports all operations specified for that type and inherited from its supertypes.

c) Reference: Clause 7

d) Test Type: Capability

A.2.2 Data with operations on prism geometry

a) Test Purpose: Verify that an application schema that supports operations on the prisms of moving features satisfies the minimum requirements for specifying the prism of each moving feature.

b) Test Method: Inspect the application schema to ensure that the specification of each moving feature type includes an association to a realization of MF_RigidBody that includes all attributes and associations and supports all operations specified for that type and inherited from its supertypes.

c) Reference: Clause 8

d) Test Type: Capability

Annex B
(informative)

UML Notation

B.1 Introduction

This annex provides a brief description of UML notation as used in the UML diagrams in this International Standard.

B.2 Class

A UML class (Figure B.1) represents a concept within the system being modelled. It is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines. The top name compartment holds the class name and other general properties of the class (including stereotype); the middle list compartment holds a list of attributes; the bottom list compartment holds a list of operations. The attribute and operation compartments may be suppressed to simplify a diagram. Suppression does not indicate that there are no attributes or operations.

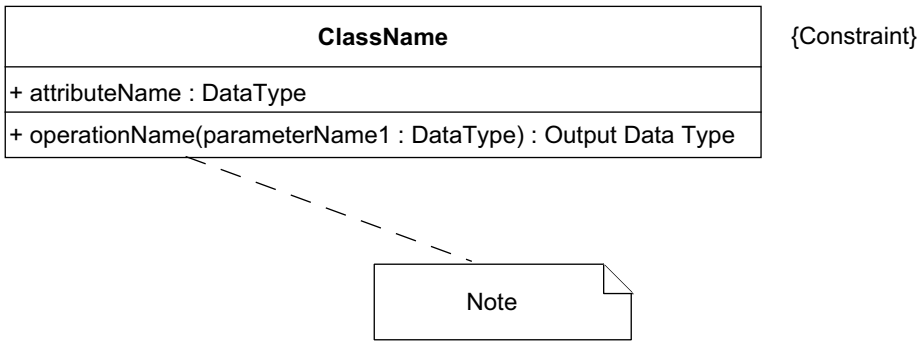

Figure B.1 — UML Class

ISO/TS 19103 specifies that a class name shall include no blank spaces and that individual words in the name shall begin with capital letters.

B.3 Stereotype

Stereotypes extend the semantics, but not the structure of pre-existing types and classes. Class level stereotypes used in this International Standard include:

<<Type>> is a stereotype of class defined by ISO/IEC 19501. A Type is used to specify a domain of objects together with operations applicable to the objects without defining the physical implementation of those objects. It may also have attributes and associations that are defined solely for the purpose of specifying the behaviour of the type's operations and do not represent any actual implementation of state data.

<<Interface>> is a stereotype of class defined by ISO/IEC 19501. An Interface contains a set of operations that together define a service offered by a class realizing the interface. A class may realize several Interfaces, and several classes may realize the same Interface. Interfaces may not have attributes, associations, or methods. An Interface may participate in an association provided the Interface cannot see the association;

that is, a class (other than another Interface) may have an association to an interface that is navigable from the class but not from the Interface.

<<DataType>> is a descriptor of a set of values that lack identity (independent existence and the possibility of side effects). Data types include primitive predefined types and user-definable types. A DataType is thus a class with few or no operations whose primary purpose is to hold the abstract state of another class for transmittal, storage, encoding or persistent storage.

<<Enumeration>> is a data type whose instances form a list of named literal values. Both the enumeration name and its literal values are declared. Enumeration means a short list of well-understood potential values within a class. Classic examples are Boolean that has only 2 (or 3) potential values TRUE, FALSE (and NULL). Most enumerations will be encoded as a sequential set of Integers, unless specified otherwise. The actual encoding is normally only of use to the programming language compilers.

<<CodeList>> defined in ISO/TS 19103 is a flexible enumeration that uses string values through a binding of the Dictionary type key and returns values as string types; e.g. Dictionary (String, String). Code lists are useful for expressing a long list of potential values. If the elements of the list are completely known, an enumeration shall be used; if the only likely values of the elements are known, a code list shall be used. Enumerated code lists may be encoded according to a standard, such as ISO 3166-1. Code lists are more likely to have their values exposed to the user, and are therefore often mnemonic. Different implementations are likely to use different encoding schemes (with translation tables back to other encoding schemes available).

<<Union>> defined in ISO 19107, is a type consisting of one and only one of several alternatives (listed as member attributes). This is similar to a discriminated union in many programming languages. In some languages using pointers, this requires a "void" pointer that can be cast to the appropriate type as determined by a discriminator attribute.

B.4 Attribute

An attribute represents a characteristic common to the objects of a class. An attribute is specified by a text string that can be parsed into elements that describe the properties of the attribute:

visibility name [multiplicity]: type-expression = initial-value

where:

visibility may be public (indicated by “+”) or private (indicated by “−”).

name is a character string. ISO/TS 19103 specifies that an attribute name shall include no blank spaces, that it shall begin with a lower case letter, and that individual words in the name, following the first word, shall begin with upper case letters.

multiplicity specifies the number of values that an instance of a class may have for a given attribute. Notation for multiplicity is explained in B.10.

type-expression identifies the data type of the attribute.

initial value specifies the default value for the attribute.

B.5 Operation

An operation represents a service that can be requested from an object. An operation is specified by a text string that can be parsed into elements that describe the properties of the operation:

visibility name (parameters): output parameter(s)

where:

visibility may be public (indicated by "+") or private (indicated by "−").

name is a character string. ISO/TS 19103 specifies that an operation name shall include no blank spaces, that it shall begin with a lower case letter, and that individual words in the name, following the first word, shall begin with upper case letters.

parameters is a list of parameters, each described by a parameter name and data type. These are assumed to be input parameters unless otherwise specified.

output parameter(s) is a list of returned values, each described by a data type.

B.6 Constraint

A constraint specifies a semantic condition or restriction. Although ISO/IEC 19501 specifies and Object Constraint Language for writing constrains, a constraint may be written using any formal notation, or a natural language. A constraint is shown as a text string in braces { }. It is placed near the element to which it applies. If the notation for an element is a text string (such as an attribute), the constraint string may follow the element text string in braces. A constraint included as an element in a list applies to all subsequent elements in the list, down to the next constraint element or the end of the list.

B.7 Note

A note contains textual information. It is shown as a rectangle with a "bent corner" in the upper right corner, attached to zero or more model elements by a dashed line. Notes may be used to contain comments or constraints.

B.8 Association

An association (Figure B.2) is a semantic relationship between classes that specifies connections between their instances. An association is drawn as a solid line connecting to class rectangles. An association may have a name, represented as a character string placed near the line, but not close to either end. ISO/TS 19103 specifies that an attribute name shall include no blank spaces and that individual words in the name shall begin with upper case letters. The association ends are adorned with information pertinent to the class at that end of the association, including multiplicity and role name.

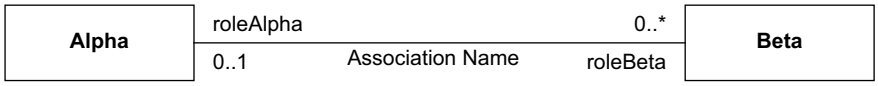

Figure B.2 — UML Associations

B.9 Role name

A role name adorning an association end specifies behaviour of the class at that end with respect to the class at the other end of the association. In Figure B.2, roleAlpha describes the role that the class named Alpha has with respect to the class named Beta. A role name is represented as a Character String. ISO/TS 19103 specifies that a role name shall include no blank spaces, that it shall begin with a lower case letter, and that individual words in the name, following the first word, shall begin with upper case letters.

B.10 Multiplicity

Multiplicity specifies the number of instances of a class that may be associated with a class at the other end of the association. The values shown in Figure B.3 are all valid. They have the following meanings:

a) zero or one instance of Alpha may be associated with one instance of Beta,

b) zero or more instances of Beta may be associated with one instance of Alpha,

c) one and only one instance of Gamma may be associated with one instance of Delta,

d) *n* being an integer number, *n* and only *n* instances of Delta may be associated with one instance of Gamma,

e) *n*1 and *n*2 being integer numbers, with *n*2>*n*1, the number of instances of Epsilon that may be associated with an instance of Phi may be within the range *n*1 to *n*2,

f) *n* being an integer number, n or more instances of Phi may be associated with one instance of Epsilon.

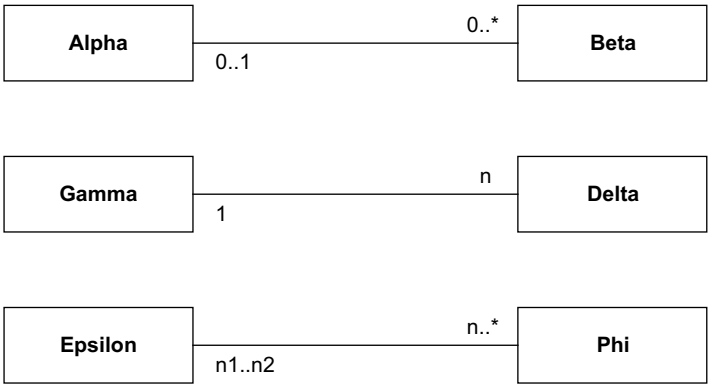

Figure B.3 — UML Multiplicity

B.11 Navigability

An arrow may be attached to the end of an end of an association path to indicate that navigation is supported toward the class attached to the arrow. For example, in Figure B.4, the association is navigable from user to supplier. This means that an instance of the class Phi has access to information held in an instance of the class Epsilon. For example, an operation specified for Phi might use the value of an attribute of Epsilon.

Figure B.4 — UML Navigability

B.12 Aggregation

Associations may be used to show aggregation or composition relationships between classes. An open diamond on an association end indicates that the class at that end of the association is an aggregate of instances of the class at the other end of the association. For example, the class named Gamma, in Figure B.5, is an aggregate of zero or more instances of the class named Delta. Aggregation is considered a weak form of composition. The members of an aggregation may exist independently of the aggregation, and may be members of more than one aggregation.

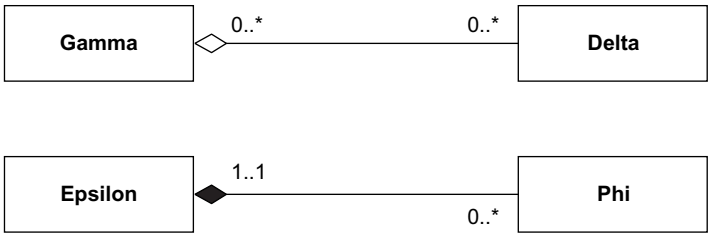

Figure B.5 — Aggregation and Composition

B.13 Composition

A closed diamond on an association end indicates that the class at that end of the association is composed of instances of the class at the other end of the association. For example, the class named Epsilon in Figure B.5 is composed of zero or more instances of the class named Phi. Members of a composite may not exist independently of the composite class, nor may they be members of more than one composite class.

B.14 Dependency

A dependency states that the implementation or functioning of one or more elements requires the presence of one or more other elements. A dependency indicates a semantic relationship between two model elements (or two sets of model elements). It relates the model elements themselves and does not require a set of instances for its meaning. A dependency is shown as a dashed arrow between two model elements. The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The kind of dependency may be indicated by a keyword in guillemets, such as <<import>>, <<refine>>, or <<use>>. In the example of Figure B.6, Epsilon has a <<use>> dependency upon Phi.

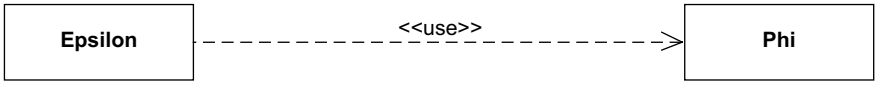

Figure B.6 — Dependency

B.15 Generalization

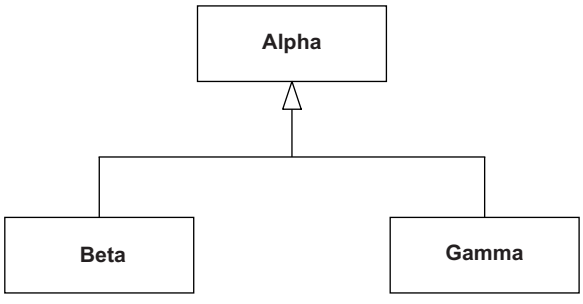

Figure B.7 — UML Generalization

ISO/IEC 19501 defines generalization (Figure B.7) as a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. Generalization is shown as a solid-line path from the child (the more specific element, such as a subclass) to the parent (the more general element, such as a superclass), with a large hollow triangle at the end of the path where it meets the more general element.

B.16 Realization

Realization specifies a realization relationship between a type or interface (the supplier) and a class that implements it (the client). The client is required to support all of the operations declared by the supplier. The implementation of a type or interface by a class is shown as a dashed line with a solid triangular arrowhead (a dashed “generalization arrow”).

Annex C
(informative)

Interpolating between orientations

C.1 Introduction

Given a pair of orientations of a moving feature at two distinct positions along a continuous path, as well as a corresponding pair of times t_0 and t_f at which the moving feature is present at those positions, there arises the problem of determining the feature's orientation at some time t_i which lies between t_0 and t_f. Such an orientation can be calculated by interpolating between the two given orientations via a process known as SLERP, or Spherical Linear intERPolation.

It is important to note that the angle between the feature orientations at t_0 and t_f must be acute. Interpolation will follow, by definition, the shortest path between the orientations. If the angle between the orientations is greater than 180 degrees then the interpolated orientation will be a mirror image of the desired result.

The following discussions will be illustrated according to the convention introduced in Figure C.1.

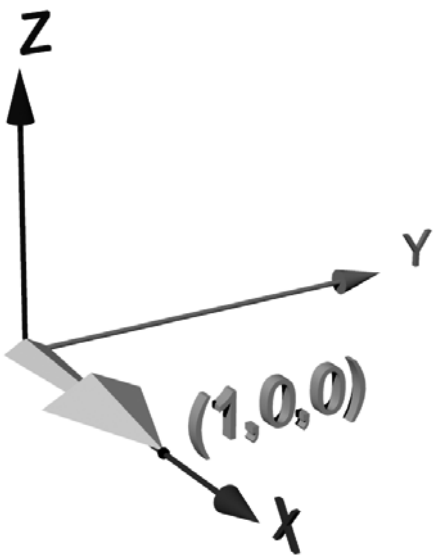

Figure C.1 — Default orientation

Rotations will be performed on the delta shape about the origin of the three main axes. Note the default position, aligned along the X-axis with "up" aligned along the Z-axis, representing the object with no rotations applied. For simplicity the object is exactly one unit in length, and its centre of rotation is the world origin. All subsequent transformations will be applied relative to this "zero" position.

C.2 Euler rotations and gimbal lock

A common method for representing orientation is as a sequence of rotations about each major axis in turn – sometimes called Euler angles. While intuitive to envision, there are some problems inherent in this method. One problem is that the rotations aren't commutative. For example, a rotation described according to the sequence X-Y-Z is not the same as the same angles expressed as Y-Z-X [11] [12]. Another potentially serious shortcoming of using Euler angles is the so-called gimbal-lock problem. For example, define an orientation

using the sequence X-Y-Z (roll-pitch-yaw) where the angles are 45 degrees, −90 degrees, and −25 degrees, respectively. The results of applying the first rotation can be seen in Figure C.2.

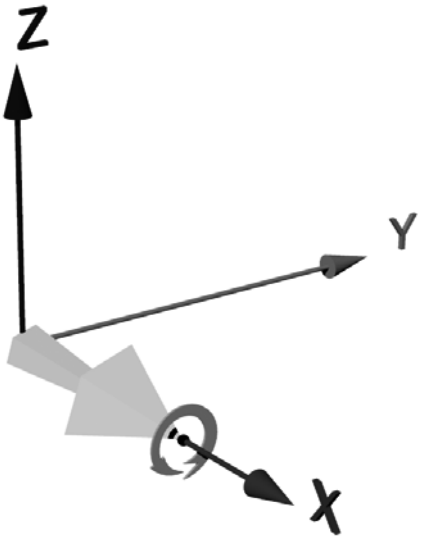

Figure C.2 — X-axis rotation

Now rotate the object −90 degrees about the Y-axis, as in Figure C.3.

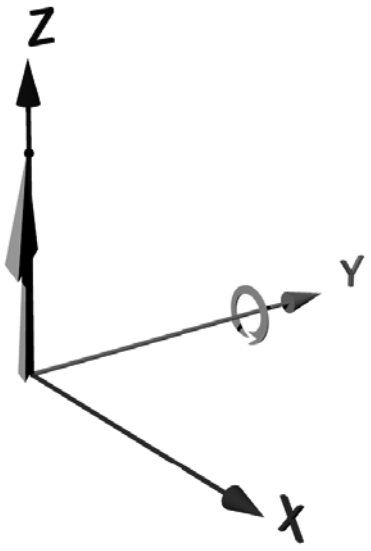

Figure C.3 — Y-axis rotation

As you can see, there is now a problem: when the final rotation about the Z-axis is applied, the object will actually be rotating about its local X-axis.

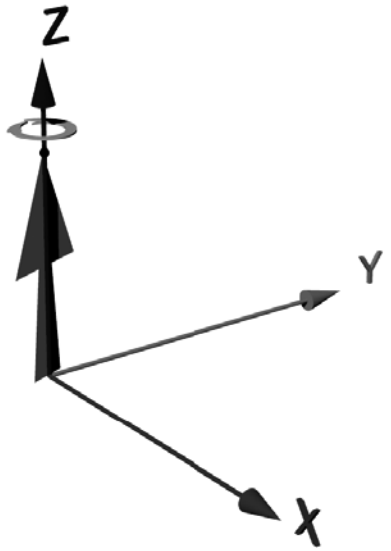

Figure C.4 — Final Z-axis rotation

In essence, the object has lost a degree of freedom – it receives no yaw rotation. In fact, the unintended additional roll actually conflicts with the original degree. One solution might seem to be to postpone the 90-degree rotation (or any rotation that aligns the object with a conflicting major axis) until last but then, since Euler angles aren't commutative, the final result will be inconsistent with the intended order (which was specified as X-Y-Z, or roll-pitch-yaw). Since future rotations could result in further instances of gimbal lock about any of the axes, the rotation order would conceivably need to be modified repeatedly to compensate, in each case coming into conflict with the requirement that all Euler rotations be applied in the same order.

A more effective solution to the gimbal-lock problem is to avoid using Euler rotations in favour of alternate methods, which aren't susceptible to this phenomenon.

C.3 Interpolating between two orientation matrices

The given pair of orientation matrices must first be converted to quaternion representation. (A detailed examination of this necessity is beyond the scope of this document. For more information refer to [13].) Each quaternion will take the form of a vector such that

$$Q =< X, Y, Z, W > \tag{1}$$

where X, Y and Z are coefficients defining an axis of rotation in an imaginary spherical space surrounding the moving feature, and W is a scalar representation of the amount of rotation applied around that axis. At this point it should be noted that the X, Y and Z components of a quaternion are not the same as the components of a three-dimensional unit vector, nor is W a direct representation of a rotation in degrees.

In the ensuing discussion the elements within orientation matrices will be referenced according to the following example:

$$M = \begin{bmatrix} mat[0] & mat[1] & mat[2] & mat[3] \\ mat[4] & mat[5] & mat[6] & mat[7] \\ mat[8] & mat[9] & mat[10] & mat[11] \\ mat[12] & mat[13] & mat[14] & mat[15] \end{bmatrix}. \tag{2}$$

Calculate the trace *T* for each original orientation matrix as follows:

$$T = 1 + mat[0] + mat[5] + mat]10] \tag{3}$$

Test for $T > (0 + \varepsilon)$ where ε represents the limit of a given system's ability to approximate zero within rounding errors (a useful practical value for ε might be 0.00000001). If $T > (0 + \varepsilon)$ holds true, then the components of the quaternion are calculated as follows:

$$X = \frac{mat[6] - mat[9]}{2\sqrt{T}}, \tag{4}$$

$$Y = \frac{mat[8] - mat[2]}{2\sqrt{T}}, \tag{5}$$

$$Z = \frac{mat[1] - mat[4]}{2\sqrt{T}}, \tag{6}$$

$$W = \frac{\sqrt{T}}{2}. \tag{7}$$

In the case where $T = 0$ (within the limits of the ability to approximate zero digitally), the algorithm for deriving *Q* is more involved, but still straightforward. Presented in the C computing language, the procedure is as follows:

```
if ( mat[0] > mat[5] && mat[0] > mat[10] )
     {//intermediate value S to simplify subsequent code
      S = sqrt( 1.0 + mat[0] - mat[5] - mat[10] ) * 2;
      X = 0.25 * S;
      Y = (mat[4] + mat[1] ) / S;
      Z = (mat[2] + mat[8] ) / S;
      W = (mat[6] - mat[9] ) / S;
     }
else if ( mat[5] > mat[10] )
     {
      S = sqrt( 1.0 + mat[5] - mat[0] - mat[10] ) * 2;
      X = (mat[4] + mat[1] ) / S;
      Y = 0.25 * S;
      Z = (mat[9] + mat[6] ) / S;
      W = (mat[8] - mat[2] ) / S;
     }
else
     {
      S = sqrt( 1.0 + mat[10] - mat[0] - mat[5] ) * 2;
      X = (mat[2] + mat[8] ) / S;
      Y = (mat[9] + mat[6] ) / S;
      Z = 0.25 * S;
      W = (mat[1] - mat[4] ) / S;
     }
```

The process of interpolating an intermediate quaternion Q_i, given the initial (Q_0) and final (Q_f) quaternions defining the endpoints of a time interval *t* (on [0..1]), consists of two calculations. The value for θ, defined as one-half the angle between Q_0 and Q_f, is found using dot product:

$$\theta = \arccos(Q_0 \bullet Q_f) \tag{8}$$

where

$$Q_0 \bullet Q_f = X_0 X_f + Y_0 Y_f + Z_o Z_f + W_0 W_f \tag{9}$$

For any desired time t_i in the given interval the components of Q_i may now be found according to the formula:

$$Q_i = Q_0 \sin((1-t_i)\theta) + Q_f \sin(t_i \theta) \tag{10}$$

Recall that a quaternion takes the form of a vector $Q =< X, Y, Z, W >$ (see equation (1)). The final interpolated orientation matrix is then derived as follows:

$$M_i = \begin{bmatrix} 1-(2Y^2+2Z^2) & 2XY+2ZW & 2XZ-2YW & 0 \\ 2XY-2ZW & 1-(2X^2+2Z^2) & 2YZ+2XW & 0 \\ 2XZ+2YW & 2YZ-2XW & 1-(2X^2+2Y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

C.4 Interpolating between other orientation representations

When interpolating between two orientations represented as Euler angles or as an axis and an angle, the process is similar to that outlined above: convert the representations to quaternion form, perform the SLERP operation, then convert the resultant quaternion back to the original form.

To convert Euler angles to quaternion form, define the angle around the X axis (roll) as ψ, the Y axis (pitch) as θ, and the Z axis (yaw) as ϕ. The components of Q can then be calculated as follows [10]:

$$X = \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) - \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \tag{12}$$

$$Y = \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \tag{13}$$

$$Z = \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) \tag{14}$$

$$W = \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right) \tag{15}$$

Converting the interpolated quaternion-form rotation back to Euler angles is based on the tight coupling between Euler angles and their corresponding orientation matrix forms, and is "*very* ill-defined" [13]. Because this conversion introduces numerous potential divide-by-zero situations where yaw and roll become indistinguishable (see gimbal lock discussion above), it is strongly advised to avoid Euler angles entirely and to resort to other rotation methods, such as matrix or axis-angle form.

The process to convert angle-axis representation to quaternion (and back) is quite straightforward. Given a unit axis whose components are defined as

$$\langle A_x \quad A_y \quad A_z \rangle \tag{16}$$

and a rotation θ about that axis, the components of the corresponding quaternion Q are calculated as follows:

$$X = A_x \sin\left(\frac{\theta}{2}\right) \tag{17}$$

$$Y = A_y \sin\left(\frac{\theta}{2}\right) \tag{18}$$

$$Z = A_z \sin\left(\frac{\theta}{2}\right) \tag{19}$$

$$W = \cos\left(\frac{\theta}{2}\right) \tag{20}$$

The axis-angle representation of a quaternion Q is calculated by reversing the above process (with a small substitution to eliminate three inverse trigonometric operations):

$$\theta = 2\arccos(W) \tag{21}$$

$$A_x = \frac{X}{\sqrt{1-W^2}} \tag{22}$$

$$A_y = \frac{Y}{\sqrt{1-W^2}} \tag{23}$$

$$A_z = \frac{Z}{\sqrt{1-W^2}}\,. \tag{24}$$

C.5 Sample interpolation

To illustrate the process of interpolating an intermediate orientation, assume an initial orientation in the default position (no rotation, as per Figure C.1, above) and a final orientation (in Euler angles for ease of visualization) of −45, −45 and 90 degrees in the X, Y and Z axes, respectively. Both orientations are shown in Figure C.5.

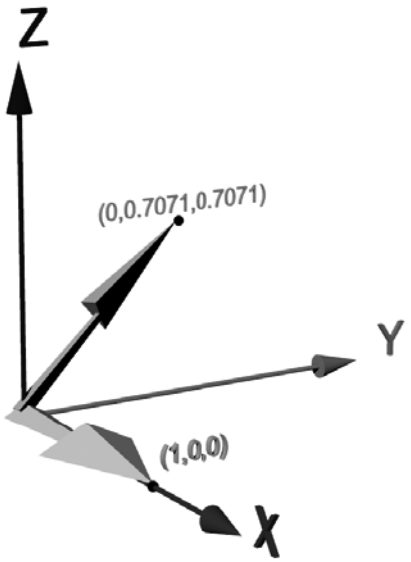

Figure C.5 — Initial and final rotations

The final orientation is:

$$\begin{bmatrix} 0 & 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.5 & -0.5 & 0 \\ -0.707107 & -0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

Alternatively, to help in visualizing the orientation, the axis-angle representation is depicted in Figure C.6.

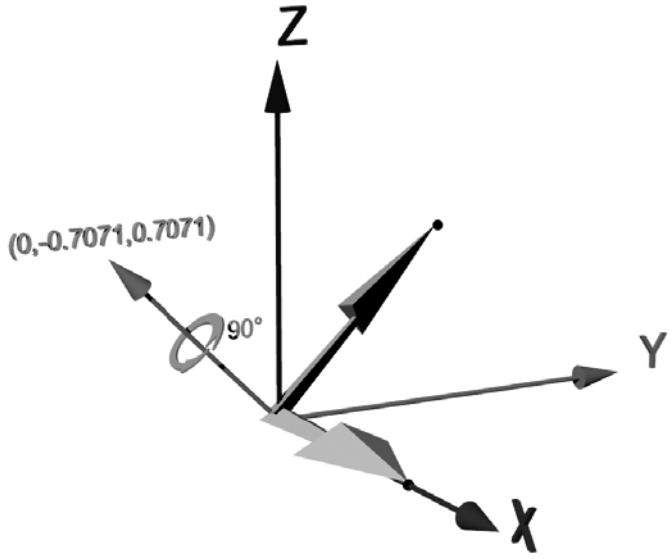

Figure C.6 — Axis-angle representation

The exercise is to interpolate an intermediate orientation for the object, depicting its orientation at a point halfway between the given ones – in other words, when *t* is 0.5.

Converting the initial orientation to quaternion form is trivial. Since the orientation is, in fact, zero, the result is simply the unit quaternion:

$$Q_0 = \langle 0 \quad 0 \quad 0 \quad 1 \rangle \tag{26}$$

Calculating the matrix trace *T* for the second orientation yields:

$$\mathrm{T} = 0 + 0.5 + 0.5 + 1 = 2 \tag{27}$$

Since 2 is easily greater than any reasonable approximation of zero, the components of Q_f are calculated as follows:

$$X = \frac{mat[6] - mat[9]}{2\sqrt{T}} = \frac{-0.5 - (-0.5)}{2\sqrt{2}} = 0 \tag{28}$$

$$Y = \frac{mat[8] - mat[2]}{2\sqrt{T}} = \frac{-0.707107 - 0.707107}{2\sqrt{2}} = -0.5\,, \tag{29}$$

$$Z = \frac{mat[1] - mat[4]}{2\sqrt{T}} = \frac{0.707107 - (-0.707107)}{2\sqrt{2}} = 0.5\,, \tag{30}$$

$$W = \frac{\sqrt{T}}{2} = \frac{\sqrt{2}}{2} = 0.707107\,. \tag{31}$$

Calculating the values needed for the SLERP equation yields:

$$Q_0 \cdot Q_f = 0.707107 \tag{32}$$

$$\theta = \arcsin(0.707107) = \pi/4 = 45^\circ \tag{33}$$

$$Q_{ix} = \frac{Q_{0x}\ \sin((1\text{-}0.5)*45^\circ) + Q_{fx}\sin(0.5*45^\circ)}{\sin(45^\circ)} = 0. \tag{34}$$

Calculating the remaining components of Q_i in similar fashion, the final interpolated quaternion is:

$$Q_i = \langle 0 \quad \text{-}0.2706 \quad 0.2706 \quad 0.9239 \rangle\,. \tag{35}$$

Converted to matrix form, the interpolated orientation matrix is:

$$\begin{bmatrix} 0.707107 & 0.5 & 0.5 & 0 \\ -0.5 & 0.853553 & 0.14645 & 0 \\ -0.5 & -0.14645 & 0.853553 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{36}$$

And, unsurprisingly, the axis-angle form is simply a 45-degree rotation about the same axis as that of the final rotation (Figure C.7).

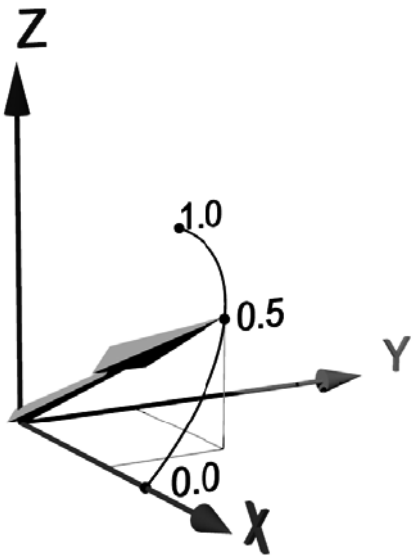

Figure C.7 — Interpolated rotation

It's important at this point to note that, while it may seem reasonable simply to use axis-angle representation and multiply the rotation angle by *t* to get an intermediate rotation, that method is only practical when the initial rotations are zero. Performing linear interpolation between two arbitrary axes of rotation can yield an intermediate axis that is not of unit length, which must them be normalized. In addition, should the axis pass through the origin, a given application must then deal with rotation about an intermediate axis of zero length. No such problems arise when applying the SLERP process, which in most cases is also computationally slightly more efficient than the alternatives.

Bibliography

[1] ISO 19101:2002, *Geographic information — Reference model*

[2] ISO 19110:2004, *Geographic information — Methodology for feature cataloguing*

[3] ISO 19111:2003, *Geographic information — Spatial referencing by coordinates*

[4] ISO 19123:2005, *Geographic information — Schema for coverage geometry and functions*

[5] ISO/IEC 19501:2005, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*

[6] C. S. JENSEN, et al. *A consensus glossary of temporal data base concepts,* ACM SIGMOD Records 1994, Vol. 23 Also available as consGlos.ps from ftp://ftp.cs.arizona.edu/tsql/doc/

[7] Object Management Group, *OMG Unified Modeling Language Specification, version 1*.3 1999, Available from World Wide Web at http://www.omg.org/cgi-bin/doc?ad/99-06-08

[8] LUCA FORLIZZI et al., *A data model and data structures for moving object databases.* Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp. 319 – 330, Available from the World Wide Web at http://portal.acm.org/citation.cfm?id=335426&dl=ACM&coll=portal

[9] OURI WOLFSON et al., *Moving objects databases: issues and solutions.* Proceedings of the 10th International Conference on Scientific and Statistical Database Management, 1998, pp. 111-122

[10] MARTIN ERWIG et al., *A foundation for representing and querying moving objects.* ACM Transactions on Database Systems March 2000, pp. 1-42

[11] BOURG, DAVID, M., *Physics for Game Developers*, O'Reilly & Associates, 2002

[12] HEARN, DONALD and BAKER, M. PAULINE, *Computer Graphics*, Prentiss Hall, 1997

[13] SAVCHENKO, SERGEI, *3D Graphics Programming*, Sams Publishing, 2000

[14] SHOEMAKE, KEN, *Animating Rotations with Quaternion Curves*, ACM SIGGRAPH 1985, Volume 19 Number 3, pp. 245-254

ICS 35.240.70

Price based on 49 pages