ISO/TC 211 N 1757

2005-01-12

Number of pages: 50

ISO/TC 211
Geographic information/Geomatics

ISO reference number: 19141

Title: **WD 19141, Geographic information - Schema for moving features**

Source: ISO/TC 211/WI 19141project leader

Expected action: For information

Type of document: Working draft

Note: This draft will be further processed by PT 19141.

Hyperlink: http://www.isotc211.org/protdoc/211n1757/

ISO/TC 211 Secretariat

Standards Norway
Strandveien 18
P.O. Box 242
NO-1326 Lysaker, Norway

Telephone: + 47 67 83 86 71
Telefax: + 47 67 83 86 01

E-mail: bjs@standard.no

URL: http://www.isotc211.org/

ISO TC 211/SC N

Date: 2005-01-06

ISO/WD 19141.2

ISO TC 211/SC /WG

Secretariat: NSF

Geographic information – Schema for moving features

Information géographique — Schéma des entités mobiles

Document type: International Standard
Document subtype:
Document stage: (20) Preparatory
Document language: E

G:\NTS\NTS_Munin\Standard\30 - IT\ISO\ISOtc211\ISO_TC_211_sekretariat\Dokumenter\211n1701-1800\211n1757\ISO 19141 WD2.doc STD Version 2.1

Contents

Foreword

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

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

Introduction

This International Standard specifies a conceptual schema that addresses moving features, i.e., features whose locations change over time. This International Standard defines the 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.

Geographic information – Schema for moving features

1 Scope

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

— The feature moves within any spatial domain as defined by ISO 19107.

— Time moves forward (past to future) along the timeline.

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

— The feature may move according to an intended schedule, but it may deviate from the intended schedule.

— Motion may be influenced by physical forces, including orbital, gravitational, or inertial.

— Motion of a feature may influence or be influenced by other features, e.g.:

  — The moving feature might follow a network (e.g., road) or change at known points (e.g., bus stops, waypoints).

  — 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).

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

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

— Succession of features or their associations,

— Change of non-spatial attributes,

— Deformation of features.

2 Conformance

2.1 Conformance classes and requirements

To be drafted. The clause will contain language adapted from the conformance clause of ISO 19107 to describe the flexibility involved in realizing types. It will specify two conformance classes: one for full object oriented implementations and one for data transfer.

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

ISO/TS 19103:—[1)], Geographic information — Conceptual schema language

ISO 19107:2003, Geographic information — Spatial schema

ISO 19108:2002, Geographic information — Temporal schema

ISO 19109:2004, Geographic information — Rules for application schema

4 Terms, definitions, and abbreviated terms

4.1 Terms and definitions

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

4.1.1
base representation
<Moving features>
representation, using a local origin and local ordinate **vectors**, of a moving 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]

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
feature
abstraction of real world phenomena [ISO 19101]

4.1.4
feature association
relationship that links instances of one **feature** type with instances of the same or a different feature type [ISO 19110]

NOTE Feature associations include aggregation of features.

---

1) To be published.

4.1.5
feature attribute
characteristic of a **feature** [ISO 19101]

4.1.6
feature operation
operation that every instance of a **feature** type may perform [ISO 19110]

4.1.7
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**

4.1.8
geometric primitive
object representing a single, connected, homogeneous element of space [ISO 19107]

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

4.1.9
instant
0-dimensional **geometric primitive** representing position in time [ISO 19108]

4.1.10
leaf
< one parameter set of geometries>
geometry at a particular value of the parameter

4.1.11
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]

4.1.12
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 to a start (origin, source) junction and end (destination, sink) junction [ISO 19133]

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.

4.1.13
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).

4.1.14
period
one-dimensional **geometric primitive** representing extent in time [ISO 19108]

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

4.1.15
point
0-dimensional **geometric primitive**, representing a position [ISO 19107]

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

4.1.16
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.

4.1.17
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]

4.1.18
temporal position
location relative to a **temporal reference system** [ISO 19108]

4.1.19
temporal reference system
reference system against which time is measured [ISO 19108]

4.1.20
trajectory
path of a moving point described by a one parameter set of points

4.1.21
vector
quantity having direction as well as magnitude [ISO 19123]

4.2 Abbreviated terms

LBS Location Based Services

UML Unified Modelling Language

5 Structure of the conceptual schema for moving features

5.1 Package structure

This clause presents a conceptual schema for describing moving features that is specified using the Unified Modelling Language (UML) [ISO 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 three leaf packages (Figure 1), Geometry Types, Rigid Body and Motion Engines, each described in a separate clause below. Four of the classes included in Geometry Types are subclassed from classes included in the Geometry Package specified in ISO 19107. Classes from the packages Basic Types [ISO 19103], Geometry [ISO 19107], Temporal, Objects and Temporal Reference System [ISO 19108] are used as data types in the schema.

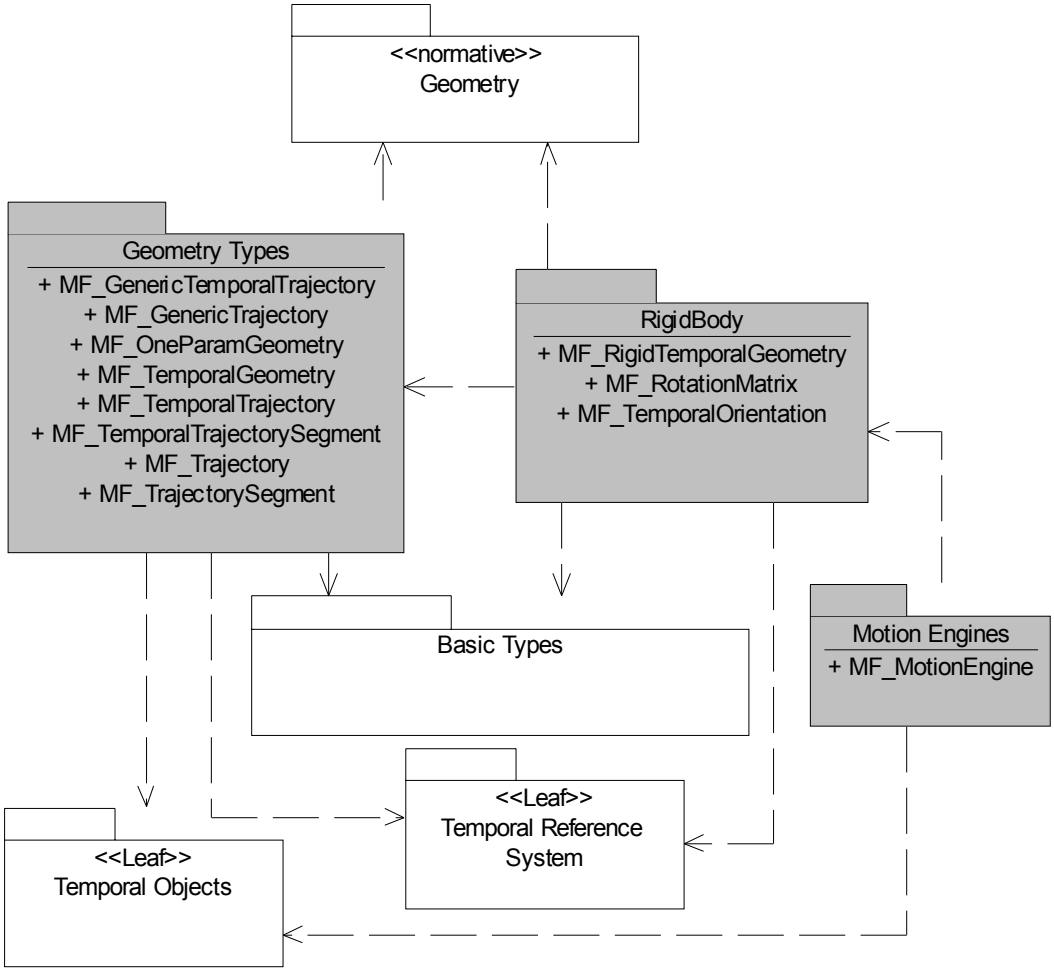

Figure 1 — Moving Feature Package

5.2 Class hierarchy

The classes of the moving features schema form an inheritance hierarchy that has its source in GM_Object and three related classes specified in ISO 19107 (Figure 2). 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 four 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. At the bottom of the hierarchy, MF_RigidTemporalGeometry describes motion of a feature without deformation. Each of the classes is specified fully in a subclause below.

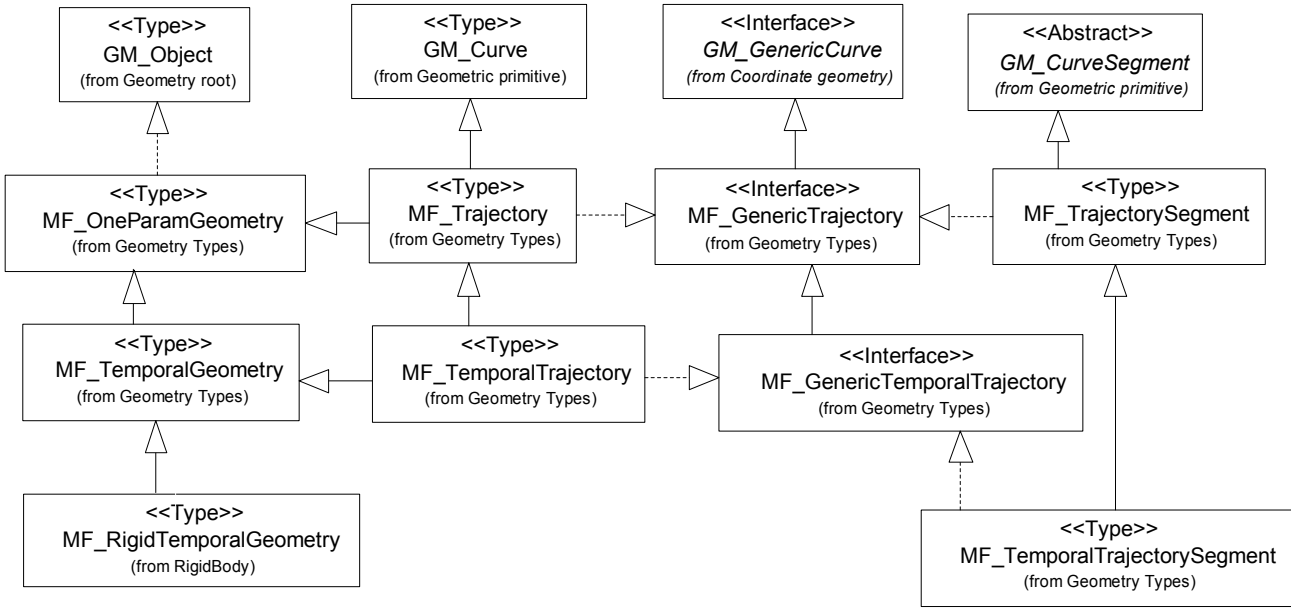

Figure 2 — Moving Feature Class Hierarchy

6 Package – Geometry Types

6.1 Introduction

The Geometry Type package contains the 8 classes shown in Figure 3. Two of these are stereotyped as interfaces while the other six are types. Four classes – MF_OneParamGeometry, MF_GenericTrajectory, MF_Trajectory, and MF_TrajectorySegment – specify a one-parameter geometry based on the geometric objects specified in ISO 19107. The other four – MF_TemporalGeometry, MF_GenericTemporalTrajectory, MF_TemporalTrajectory, and MF_TemporalTrajectorySegment – specialize the first four classes in order to specify a one parameter geometry in which the key parameter is time. Each of the classes is specified fully in a subclause below.

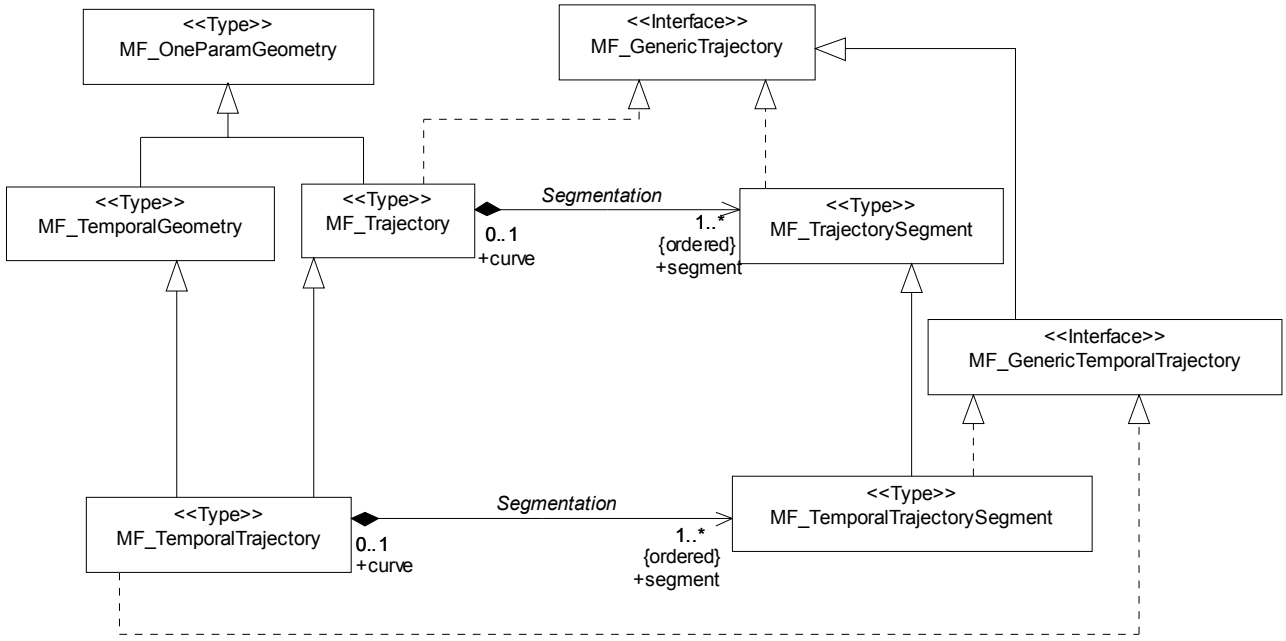

Figure 3 — Components of the Geometry Types Package

6.2 MF_OneParamGeometry

6.2.1 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 $P(t) \in f(t)$.

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

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.

A foliation is a 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.

The class MF_OneParamGeometry (Figure 4) is a realization of the type GM_Object. As such, it may support any of the operations specified for that type in ISO 19107, as well as the association Coordinate Reference System from GM_Object to a coordinate reference system (SC_CRS) as specified in IS 19111. In addition, it has the two attributes and three operations specified below (6.2.2 - 6.2.6).

This international Standard specifies two subclasses of MF_OneParamGeometry. They are MF_Trajectory (6.4) and MF_TemporalGeometry (6.6).

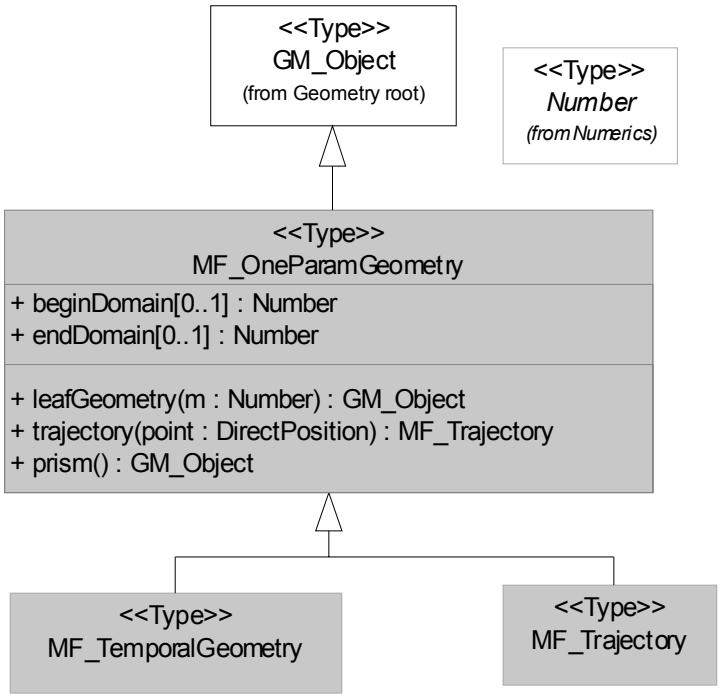

Figure 4 — Context Diagram: MF_OneParamGeometry

6.2.2 Attribute - beginDomain

The optional 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[0..1]: Number
```

6.2.3 Attribute - endDomain

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

```
MF_OneParamGeometry::endDomain[0..1]: Number
```

6.2.4 Operation - leafGeometry

The operation *leafGeometry* shall accept a value of the parameter as input and return the leaf associated with that position in the domain as an instance of GM_Object. GM_Object is specified in ISO 19107.

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

6.2.5 Operation - trajectory

The operation *trajectory* shall accept the position of a point on a leaf of the one parameter set of geometries and return the trajectory of that point.

```
MF_OneParamGeometry::trajectory(point:DirectPosition): MF_Trajectory
```

6.2.6 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 MF_GenericTrajectory

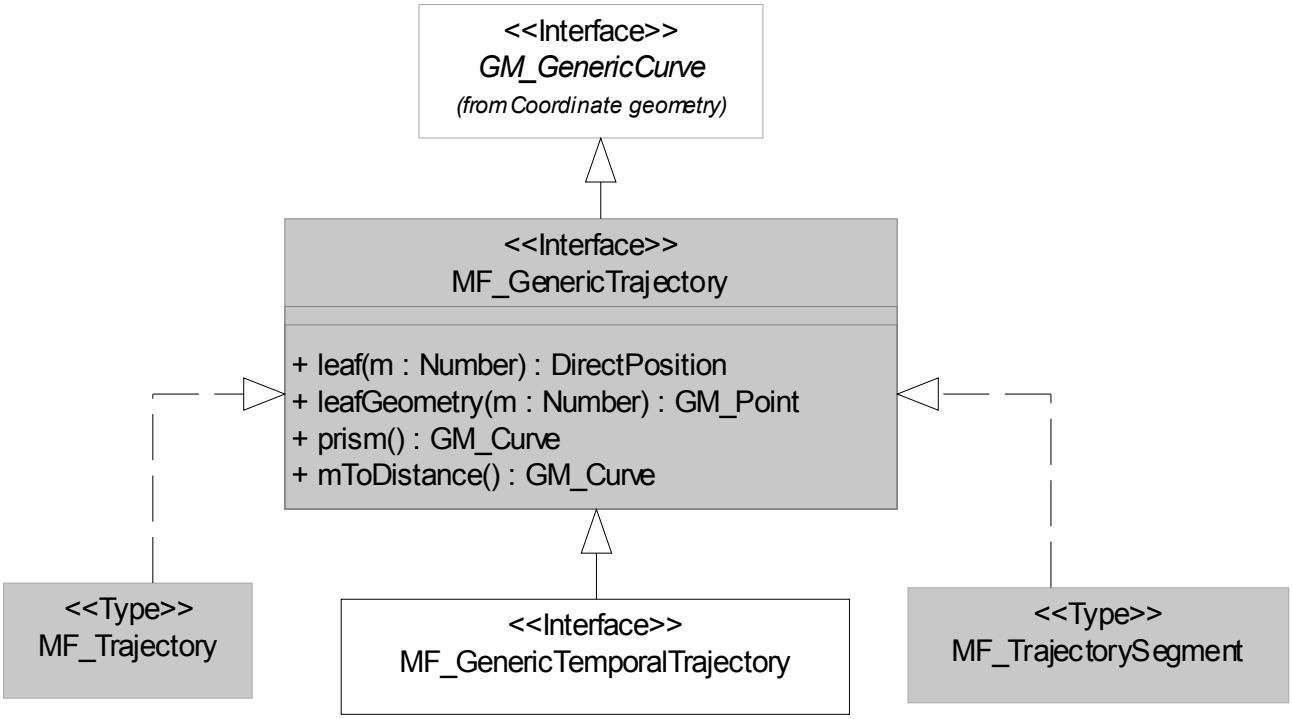

Figure 5 — Context Diagram: MF_GenericTrajectory

6.3.1 Semantics

MF_GenericTrajectory (Figure 5) is an interface that specifies a number of operations that may be supported by any trajectory. As a subtype of the interface GM_GenericCurve specified in ISO 19107, it inherits all of the operations specified there. In addition, it has the following operations (6.3.2 - 6.3.5).

6.3.2 Operation - leaf

The operation *leaf* shall accept a value of the parameter as input and return the DirectPosition through which the trajectory passes at that value of the parameter.

```
MF_GenericTrajectory::leaf(m:Number):DirectPosition
```

6.3.3 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_GenericTrajectory::leafGeometry(m:Number):GM_Point
```

6.3.4 Operation - prism

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

```
MF_GenericTrajectory::prism():GM_Curve
     MF_GenericTrajectory::mToDistance():GM_Curve
```

6.3.5 Operation - mToDistance

The operation *mToDistance* shall return an instance of GM_Curve that describes the relationship between the parameter and distance measured along the trajectory. The GM_Curve's coordinate reference system is the Cartesian product of the trajectory parameter and distance.

```
MF_GenericTrajectory::mToDistance():GM_Curve
```

6.4 MF_Trajectory

6.4.1 Semantics

MF_Trajectory (Figure 6) describes a one-parameter geometry whose cross section is a point. As a subclass of MF_OneParamGeometry (6.2) and a subclass of GM_Curve (ISO 19107), it inherits a set of attributes, associations, and operations from those two classes. It also realizes the interface MF_GenericTrajectory (6.3) and shall support the operations defined for that interface. 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)}
```

An instance of MF_Trajectory may be an aggregation of instances of MF_TrajectorySegment (6.4.2).

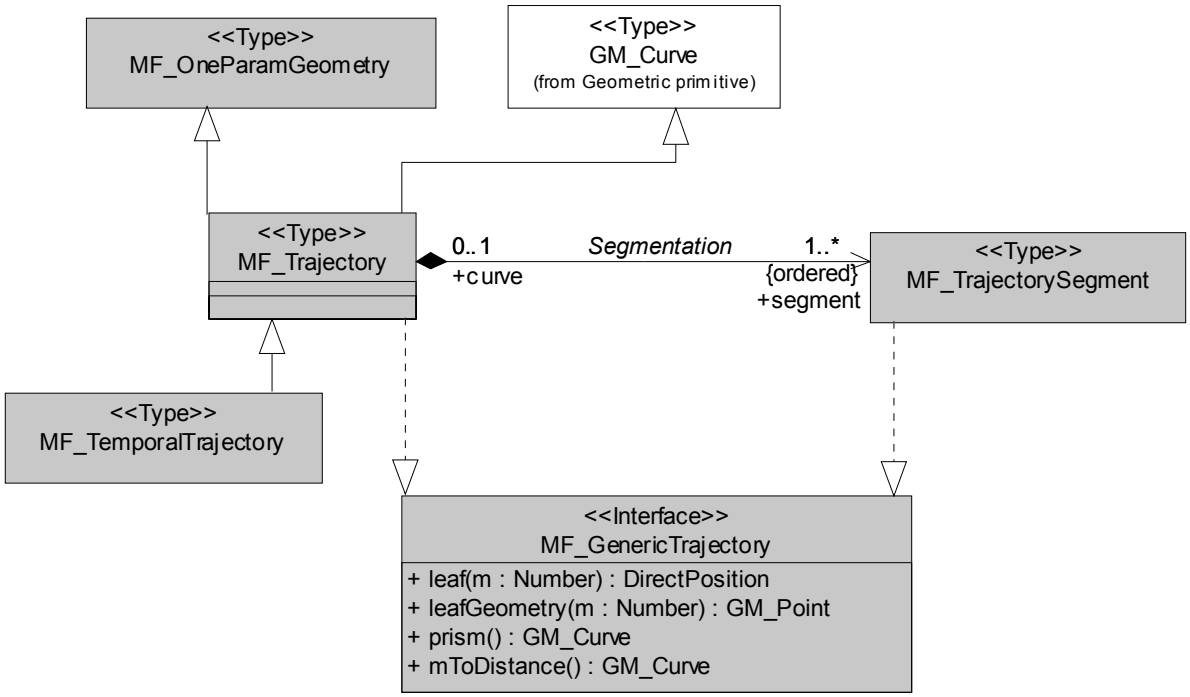

Figure 6 — Context Diagram: MF_Trajectory

6.4.2 Association Role - Segmentation::segment

The aggregation association Segmentation relates MF_Trajectory to a set of instances of MF_TrajectorySegment identified by the role name *segment* which taken together make up this instance of MF_Trajectory.

```
MF_Trajectory::segment[1..*]:MF_TrajectorySegment
```

6.5 MF_TrajectorySegment

6.5.1 Semantics

MF_TrajectorySegment (Figure 7) represents a part of a trajectory that is distinguished from other parts for some reason pertinent to the application, most likely dealing with its internal interpolation or parameterization.

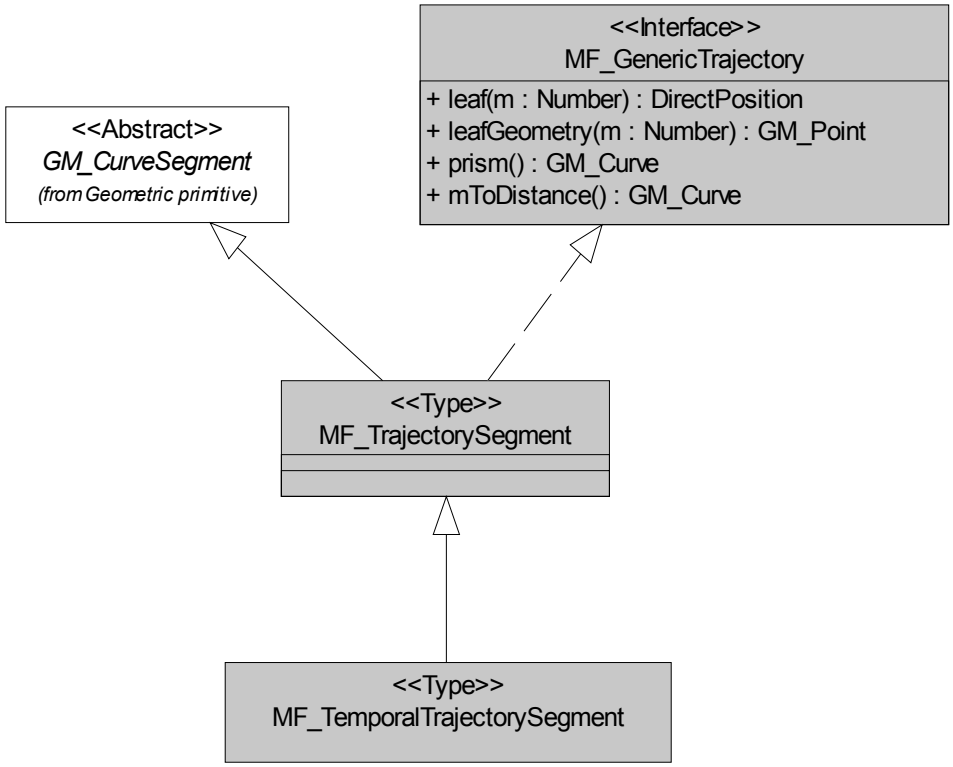

Figure 7 — Context Diagram: MF_TrajectorySegment

6.6 MF_TemporalGeometry

6.6.1 Semantics

MF_TemporalGeometry (Figure 8) 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.

6.6.2 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
```

6.6.3 Operation - trajectory

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

```
MF_TemporalGeometry::trajectory(point:DirectPosition):MF_TemporalTraj
     ectory
```

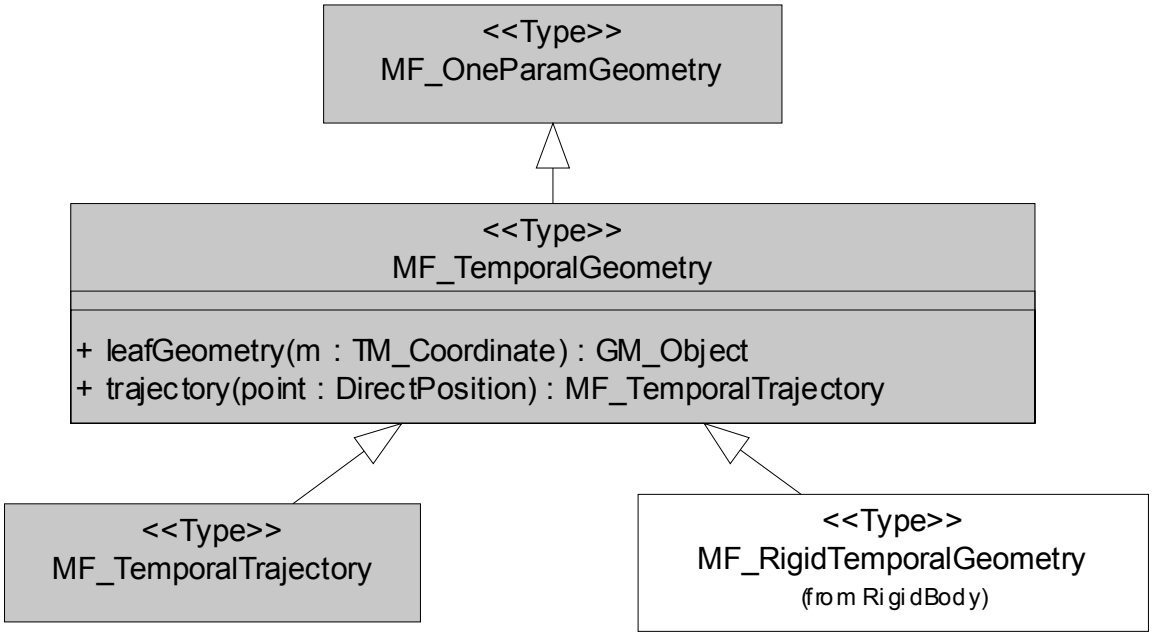

Figure 8 — Context Diagram: MF_TemporalGeometry

6.7 MF_GenericTemporalTrajectory

6.7.1 Semantics

MF_GenericTemporalTrajectory (Figure 9) is an interface that specifies a number of operations that may be supported by a temporal trajectory. As a subclass of MF_GenericTrajectory (6.3), MF_GenericTemporalTrajectory inherits the operations specified for that class. In addition, it has the 17 operations specified below (6.7.2 - 6.7.18).

6.7.2 Operation - leafGeometry

The operation *leafGeometry* shall accept a time as input and return the instance of GM_Point that is on the trajectory at that time.

```
MF_GenericTemporalTrajectory::leafGeometry(t:TM_Coordinate): GM_Point
```

6.7.3 Operation - startTime

The operation *startTime* shall return the time at which the temporal trajectory begins.

```
MF_GenericTemporalTrajectory::startTime():TM_Coordinate
```

6.7.4 Operation - endTime

The operation *endTime* shall return the time at which the temporal trajectory ends.

```
MF_GenericTemporalTrajectory::endTime():TM_Coordinate
```

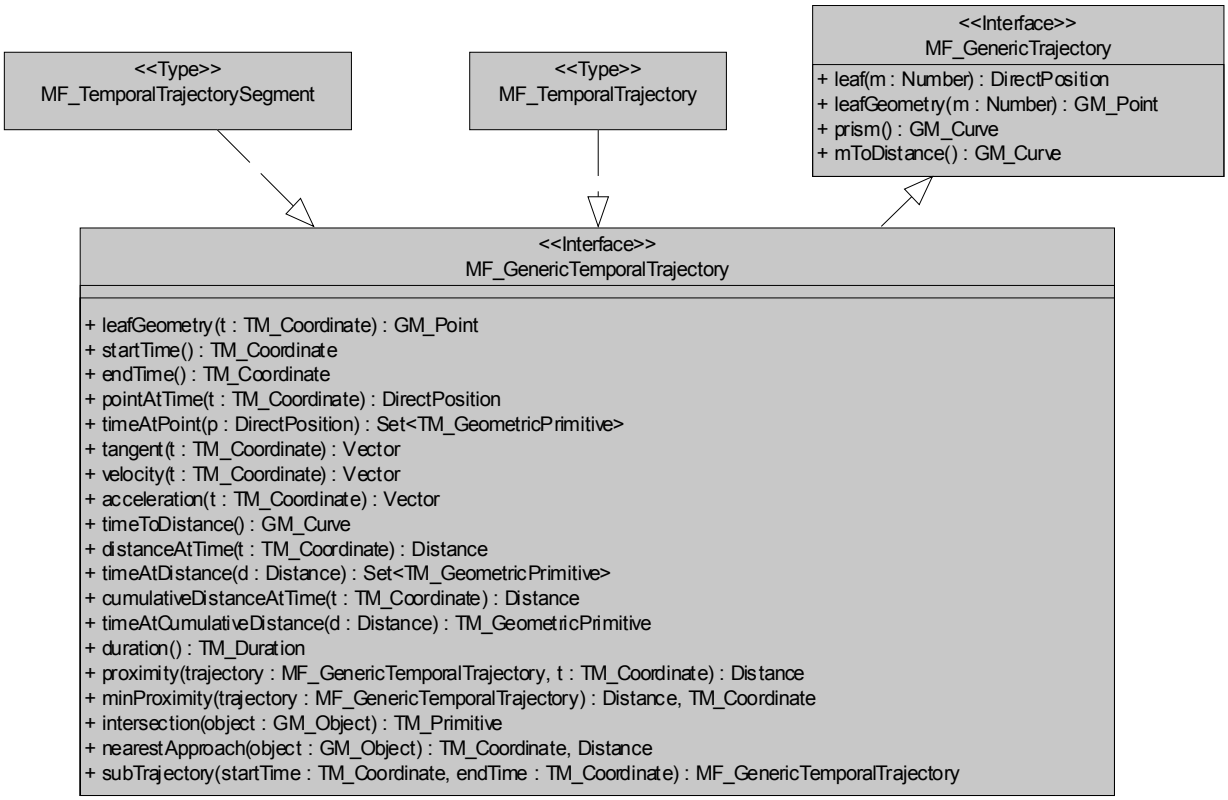

Figure 9 — Context Diagram: MF_GenericTemporalTrajectory

6.7.5 Operation - pointAtTime

The operation *pointAtTime* shall accept a time (TM_Coordinate) as input and return the DirectPosition through which the trajectory passes at that time.

```
MF_GenericTemporalTrajectory::pointAtTime(t:TM_Coordinate):
     DirectPosition
```

6.7.6 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_GenericTemporalTrajectory::timeAtPoint(p:DirectPosition):
     Set<TM_GeometricPrimitive>
```

6.7.7 Operation - tangent

The operation *tangent* shall accept a time (TM_Coordinate) as input and return the tangent to the trajectory at that time. The data type Vector is specified in ISO/TS 19103.

```
MF_GenericTemporalTrajectory::tangent(t:TM_Coordinate): Vector
```

6.7.8 Operation - velocity

The operation *velocity* shall accept a time (TM_Coordinate) as input and return the velocity of the object moving along the trajectory at that time.

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

6.7.9 Operation - acceleration

The operation *acceleration* shall accept a time (TM_Coordinate) as input and return the acceleration of the object moving along the trajectory at that time.

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

6.7.10 Operation - timeToDistance

The operation *timeToDistance* shall return an instance of GM_Curve, in a 2 dimensional coordinate space (t, s) where t is the time from the *startTime* of the trajectory, and s is the distance parameter for the underlying curve of the trajectory for the position of the leaf at time "t". This GM_Curve describes the relationship between time and distance measured along the trajectory.

```
MF_GenericTemporalTrajectory::timeToDistance():GM_Curve
```

EXAMPLE In Figure 10, 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.

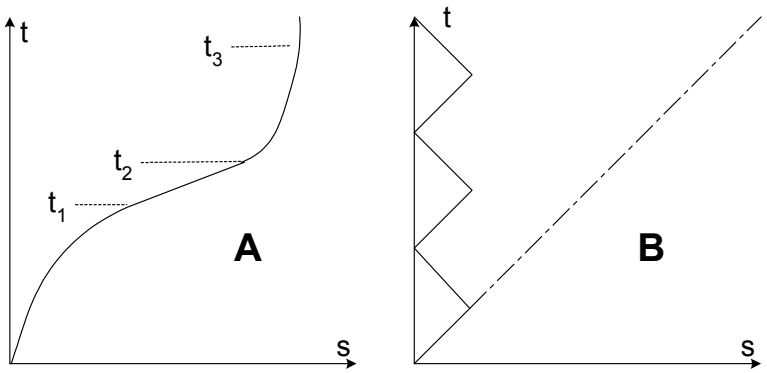

Figure 10 — Examples of time to distance curves

6.7.11 Operation - distanceAtTime

The operation "*distanceAtTime*" shall accept a time (TM_Coordinate) as input and return the geometric distance from the startPoint to the position on the trajectory at that time.

```
MF_GenericTemporalTrajectory::distanceAtTime(t:TM_Coordinate):
     Distance
```

6.7.12 Operation - timeAtDistance

The operation *timeAtDistance* shall accept a distance as input and return the time at which the moving object reaches that distance along the trajectory's underlying curve as measured from the *startPoint*. 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.

```
MF_GenericTemporalTrajectory::timeAtDistance(d:Distance):
     Set<TM_GeometricPrimitive>
```

6.7.13 Operation - cumulativeDistanceAtTime

The operation *cumulativeDistanceAtTime* shall accept a time as input and return the cumulative distance travelled by the object at that time.

```
MF_GenericTemporalTrajectory::cumulativeDistanceAtTime(t:TM_Coordinat
     e): Distance
```

6.7.14 Operation - timeAtCumulativeDistance

The operation *timeAtCumulativeDistance* shall accept a distance as input and return the time at which the object was at that cumulative distance.

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

6.7.15 Operation - duration

The operation "*duration*" shall return the elapsed time from the time when the trajectory begins to the time when the trajectory ends. The TM_Duration data type is defined in ISO 19108.

```
MF_GenericTemporalTrajectory::duration():TM_Duration
```

6.7.16 Operation - proximity

The operation *proximity* shall accept another trajectory as input, together with a time, and shall return the distance from the point associated with the input time on this trajectory to the point associated with the input time on the other trajectory.

```
MF_GenericTemporalTrajectory::proximity(
     trajectory:MF_GenericTemporalTrajectory, t:TM_Coordinate):
     Distance
```

6.7.17 Operation - minProximity

The operation *minProximity* shall accept another trajectory as input and shall return the instant at which the two trajectories are closest, and the distance between the two points associated with that time.

```
MF_GenericTemporalTrajectory::minProximity(trajectory ;
     MF_GenericTemporalTrajectory:):Distance, TM_Coordinate
```

6.7.18 Operation – nearestApproach

The operation *nearestApproach* shall accept a GM_Object as input and return the time at which the trajectory is closest to that object and the distance from the object at that time.

```
MF_GenericTemporalTrajectory::nearestApproach(object:GM_Object):
     TM_Coordinate, Distance
```

6.7.19 Operation – intersection

The operation *intersection* shall accept a GM_Object as input and return the time during which the temporal trajectory intersects that GM_Object. This operation depends upon the intersection operations specified for GM_Object in ISO 19107. Depending upon the type of intersection, the operation may return a TM_Instant or a TM_Period.

```
MF_GenericTemporalTrajectory::intersection(object:GM_Object):
     TM_Primitive
```

6.7.20 Operation - subTrajectory

The operation *subTrajectory* shall accept two times as input and return a sub-trajectory that begins at the first input time and ends at the second.

```
MF_GenericTemporalTrajectory::subTrajectory(startTime : TM_Coordinate,
     endTime:TM_Coordinate) : MF_GenericTemporalTrajectory
```

6.8 MF_TemporalTrajectory

6.8.1 Semantics

MF_TemporalTrajectory (Figure 11) is a subclass of both MF_TemporalGeometry and MF_Trajectory. As such, it inherits the attributes, associations, and operations of both classes.

The parameter "m : time" as used in the leaf() operation inherited from MF_Trajectory is numerically equivalent to the constructive parameter "cp : Real" that is the input to the constrParam() operation of GM_Curve (ISO 19107). This is expressed by the OCL constraint:

```
OCL::MF_TemporalTrajectory
[MF_TemporalGeometry::leafGeometry(t).position =
     MF_Trajectory::leaf(t)]
```

The class realizes the interface MF_GenericTemporalTrajectory.

An instance of MF_TemporalTrajectory is composed of a set of instances of MF_TemporalTrajectorySegment.

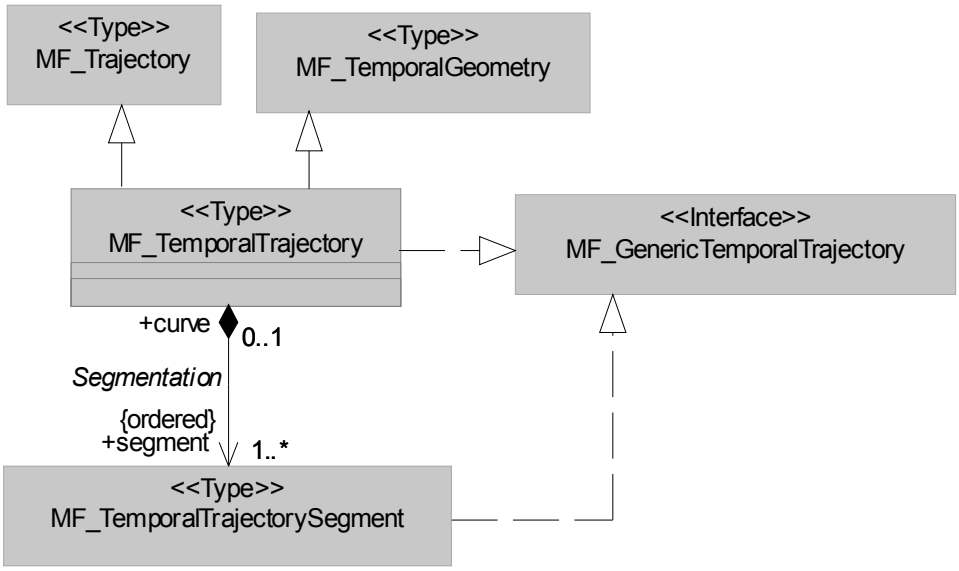

Figure 11 — Context Diagram: MF_TemporalTrajectory

6.8.2 Association role – Segmentation::segment

The aggregation association Segmentation relates MF_TemporalTrajectory to the set of instances of MF_TemporalTrajectorySegment identified by the role name *segment* which taken together make up this instance of MF_TemporalTrajectory.

```
MF_TemporalTrajectory::segment[1..*]:MF_TemporalTrajectorySegment
```

6.9 MF_TemporalTrajectorySegment

6.9.1 Semantics

MF_TemporalTrajectorySegment () is a subclass of MF_TrajectorySegment specialized so that the parameter is time. The semantics of MF_TemporalTrajectorySegment is otherwise identical to that of MF_TrajectorySegment.

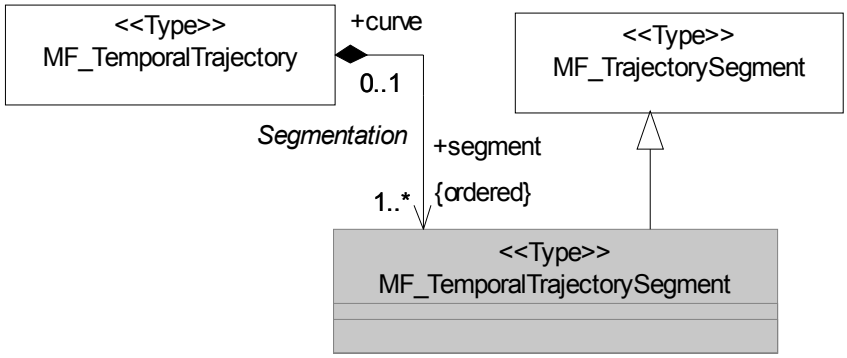

Figure 12 — Context Diagram: MF_TemporalTrajectorySegment

6.9.2 Association role – Segmentation::curve

The aggregation association Segmentation relates MF_TemporalTrajectorySegment to the instance of MF_TemporalTrajectory identified by the role name *curve* of which this instance of MF_TemporalTrajectorySegment is a part.

```
MF_TemporalTrajectorySegment::curve[0..1]:MF_TemporalTrajectory
```

7 Package – Rigid Body

7.1 MF_RigidTemporalGeometry

7.1.1 Semantics

MF_RigidTemporalGeometry () 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.

7.1.2 Attribute - baseGeometry

The baseGeometry is the geometry of the moving object in a local rectangular coordinate system based on the axes of the object. The object should have a natural up, front and right (cross product of up and front).

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

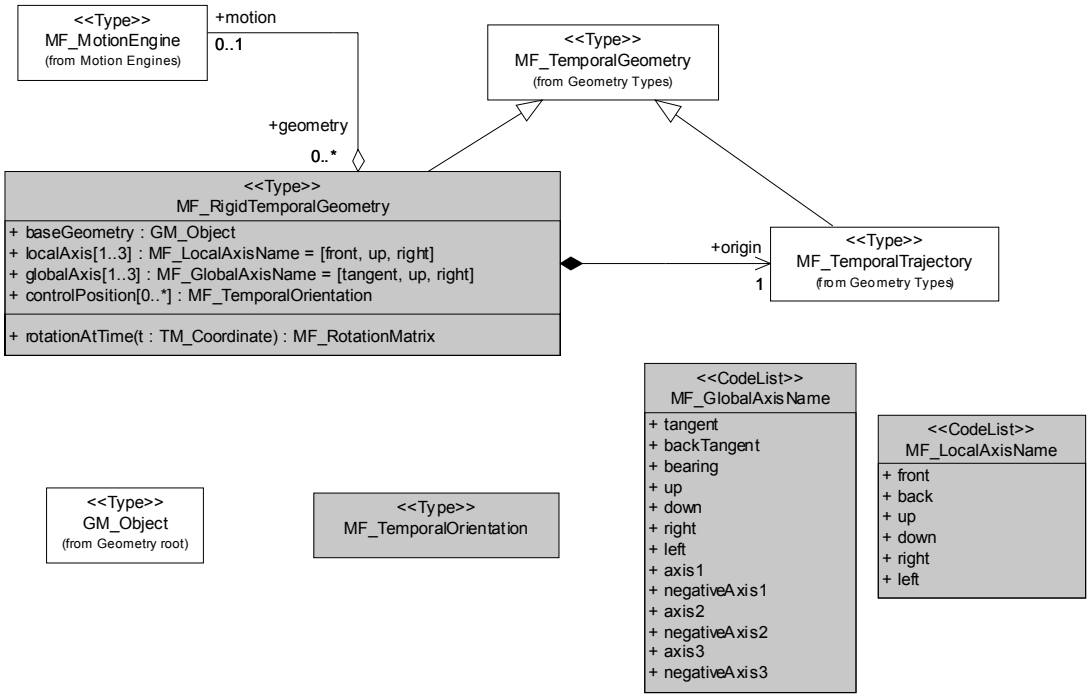

Figure 13 — Context Diagram: MF_RigidTemporalGeometry

7.1.3 Attribute – localAxis

The attribute *localAxis* specifies the local geometric vector frame (Figure 14) in which the base geometry is defined, a right-handed local Euclidean space. If the third vector is not given, it is assumed to be the cross product of the other two.

```
MF_RigidTemporalGeometry::localAxis[1..3]: MF_LocalAxisName=[front,
     up, right]
```

7.1.4 Attribute – globalAxis

The attribute *globalAxis* specifies the global, moving, geometric coordinate frame (represented as three axes with respect to points along the trajectory curve) in which the rotation of the base geometry is defined. It is a right-handed local Euclidean vector space, containing or equal to the tangent space of the CRS at the points along the curve. If the third vector is not given, it is assumed to be the cross product of the first two. If only the first is given, it is assumed to be a horizontal vector, with the second being up and the third being the cross product of the first two.

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

7.1.5 Attribute - controlPosition

The *controlPosition* array contains some number of rotational positions distributed along the curve by time. The motion engine is use to interpolate using the time parameter between these orientations.

```
MF_RigidTemporalGeometry::controlPosition[0..*] :
     MF_TemporalOrientation
```

7.1.6 Association role - origin

The *origin* is the trajectory curve of the origin of the local coordinate system (localAxis) used to describe the rigid body.

```
MF_RigidTemporalGeometry::origin: MF_TemporalTrajectory
```

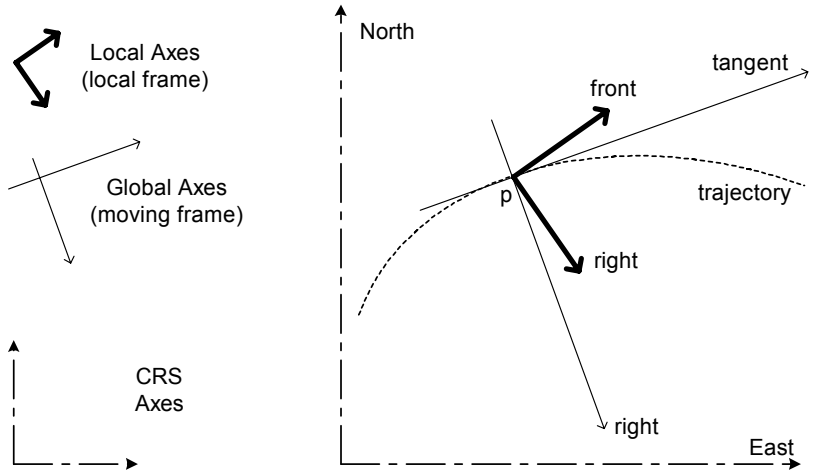

Figure 14 — 2-D example of relevant coordinate axes

7.1.7 Association role - motion

The *motion* engine shall provide the mechanism for calculating the orientation of the object as it moves along the trajectory curve.

```
MF_RigidTemporalGeometry::motion: MF_MotionEngine
```

7.1.8 Operation - rotationAtTime

The operation *rotationAtTime* shall accept a TM_Coordinate as input and return the rotation of the rigid body at that time. . The value returned should agree with the *controlPosition* array at common values of time. The operation shall return an error message if the input time is not within the domain.

```
MF_RigidTemporalGeometry::rotationAtTime(t:TM_Coordinate):MF_Rot
ationMatrix
```

7.2 MF_TemporalOrientation

7.2.1 Semantics

The MF_TemporalOrientation type (Figure) is designed to capture the rotational motion of the object as it passes along its trajectory. As such, it contains information on orientation at a particular time. The *controlPosition* array in MF_RigidTemporalGeometry aggregates these positions for interpolation by a motion engine.

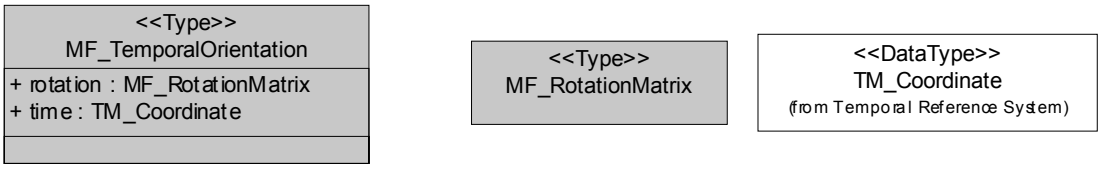

Figure 15 — Context Diagram: MF_TemporalOrientation

7.2.2 Attribute - rotation

The attribute *rotation* shall contain a rotation matrix or its equivalent for a particular time.

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

7.2.3 Attribute - time

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

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

7.3 MF_RotationMatrix

7.3.1 Semantics

The MF_RotationMatrix is designed to capture the moving frame of the object in terms of the global frame being used. The frames are defined in the MF_RigidTemporalGeometry type, by the localAxis and globalAxis attributes.

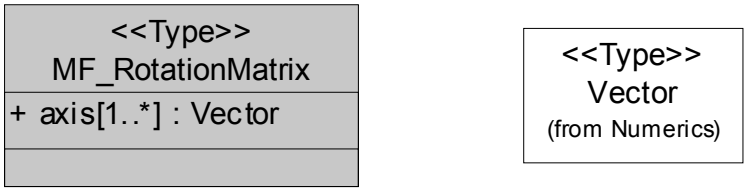

Figure 16 — Context Diagram: MF_RotationMatrix

7.3.2 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
```

7.4 CodeList - MF_LocalAxisName

7.4.1 Class semantics

The code list “MF_LocalAxisName” contains choices for the local coordinate axes for the base geometry of the moving object.

7.4.2 Attribute - front

The front is the vector towards the front of the object from its center.

`MF_LocalAxisName::front`

7.4.3 Attribute - back

The back vector is the negative of front.

`MF_LocalAxisName::back`

7.4.4 Attribute - up

The up vector is towards the nominal top of the object.

`MF_LocalAxisName::up`

7.4.5 Attribute - down

The down vector is the negative of up.

`MF_LocalAxisName::down`

7.4.6 Attribute - right

The right is the cross product of front and up.

`MF_LocalAxisName::right`

7.4.7 Attribute - left

The left is the cross product of up and front, the negative of right.

`MF_LocalAxisName::left`

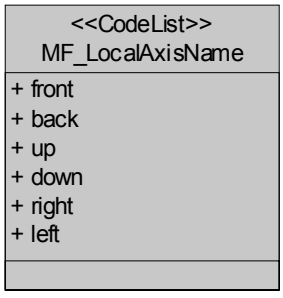

Figure 17: Context Diagram: MF_LocalAxisName

7.5 CodeList - MF_GlobalAxisName

7.5.1 Class semantics

This code list names the usual global axes. They are normally either in terms of the moving frame of the curve, or in terms of the coordinate curve tangents. Steerable objects may work best with the moving frame (front usually maps to the tangent, up to up), and non-steered object may work best with the coordinate curve tangents.

7.5.2 Attribute - tangent

This is the vector tangent to base curve with respect to the geometry, not to the motion of the object. If the object backtracks the curve, the forward tangent of the object will be the negative of the curve's tangent.

```
MF_GlobalAxisName::tangent
```

7.5.3 Attribute - backTangent

This vector is the negative of the tangent to the curve.

```
MF_GlobalAxisName::backTangent
```

7.5.4 Attribute - bearing

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

```
MF_GlobalAxisName::bearing
```

7.5.5 Attribute - up

This vector is the local up as in elevation.

```
MF_GlobalAxisName::up
```

7.5.6 Attribute - down

This vector is the negative of up.

```
MF_GlobalAxisName::down
```

7.5.7 Attribute - right

This vector is the cross product of tangent and up.

```
MF_GlobalAxisName::right
```

7.5.8 Attribute - left

This vector is the cross product of up and tangent.

```
MF_GlobalAxisName::left
```

7.5.9 Attribute - axis1

This is the first coordinate axis of the CRS.

```
MF_GlobalAxisName::axis1
```

7.5.10 Attribute - negativeAxis1

This is negative of the first coordinate axis of the CRS.

`MF_GlobalAxisName::negativeAxis1`

7.5.11 Attribute - axis2

This is the second coordinate axis of the CRS.

`MF_GlobalAxisName::axis2`

7.5.12 Attribute - negativeAxis2

This is the negative of the second coordinate axis of the CRS.

`MF_GlobalAxisName::negativeAxis2`

7.5.13 Attribute - axis3

This is the third coordinate axis of the CRS.

`MF_GlobalAxisName::axis3`

7.5.14 Attribute - negativeAxis3

This is the negative of the third coordinate axis of the CRS.

`MF_GlobalAxisName::negativeAxis3`

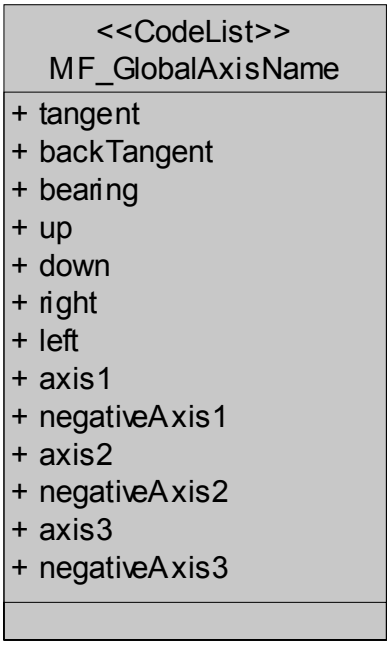

Figure 18: Context Diagram: MF_GlobalAxisName

8 Package – Motion Engines

8.1 Introduction

8.2 Class – MF_MotionEngine

The MF_MotionEngine type is designed to be able to interpolate rotation matrices for a particular set of moving geometric objects.

8.2.1 Association role – geometry

The *geometry* role associates the motion engine to an instance ofMF_RigidTemporalGeometry to allow it to interrogate it for information needed in interpolation. Depending on the implementation, this association role may not be needed as the motion engine receives the geometry's identity in the interpolation call. Having it allows application developers to better control threading and other runtime resources. This is an implementation consideration and, therefore, navigability of this association from the MF_MotionEngine to the MF_RigidTemporalGeometry is not mandatory for all applications.

```
MF_MotionEngine::geometry[0..*]: MF_RigidTemporalGeometry
```

8.2.2 Operation – interpolate

The *interpolate* function is the operation protocol to allow the associated geometric objects to interpolate their rotational motion over time. Annex C describes one method for interpolating rotational motion.

```
MF_MotionEngine::interpolate(g: MF_RigidTemporalGeometry, t:
     TM_Coordinate): MF_RotationMatrix
```

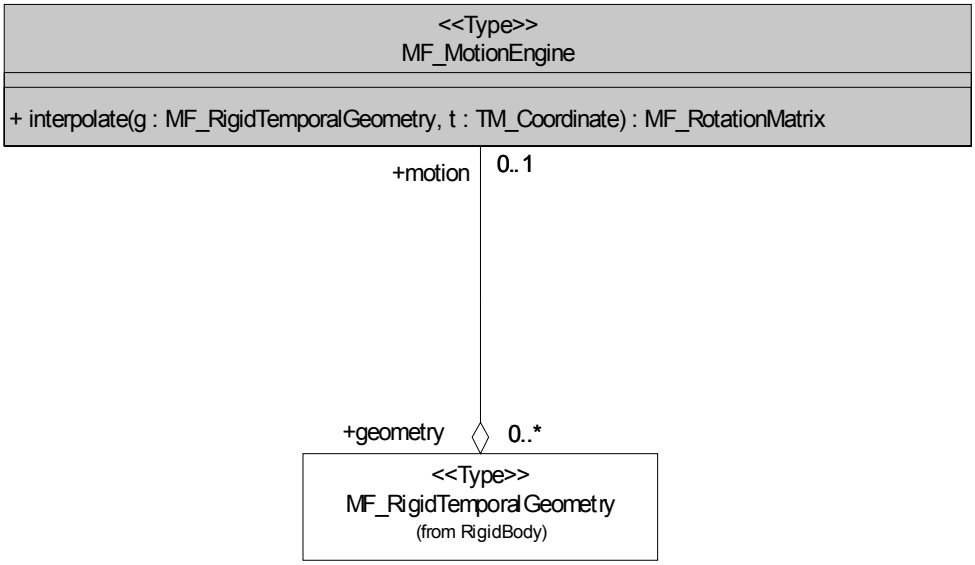

Figure 19 - Context diagram for MF_MotionEngine

9 Moving features in application schemas

9.1 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 one. 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.

9.2 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 (5.2). 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.

9.3 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 within the GM_Complex.

Associations between feature instances may develop as a result of movement of one or both of the associated features. In this case, the application schema shall specify an association between the feature types as an association class with a temporal attribute of the association to contain the duration of the association. See the discussion in ISO 19108 of modelling of feature associations with temporal characteristics.

EXAMPLE 2 A moving aircraft participates in a sequence of associations with individual air traffic control centres.

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

EXAMPLE 3 Trucks in a convoy participate in a set of associations with the other trucks in that convoy.

9.4 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.

We need to discuss the extent to which the model does or does not specify the basic operations in the table below. The table should be deleted after that discussion.

The table below shows a set of possible operations for moving features. The left column shows inputs, the top row shows outputs, and the body of the table names the commands.

Out-puts / Inputs	Travel time	Abs. location	Trajectory	Travel distance	Location relative to another feature
Travel time		whereAtTime	pathAtTime	distAtTime	relAtTime
Abs. location	whenAtPlace		PathAtPlace	distAtPlace	relAtPlace
Trajectory	whenOnPath			distOnPath	relOfPath
Travel distance	whenAtDist	whereAtDist	pathAtDist		relAtDist
Location relative to another feature	whenIntersect	whereIntersect	pathToIntersect	distToIntersect	

Annex A
(normative)

Abstract test suite

To be revised.

A.1 Application schemas for data transfer

a) Test Purpose: Verify that an application schema for data transfer defines moving features in compliance with specified requirements.

b) Test Method: Inspect the presentation of the moving features included in the application schema to ensure that the definitions satisfy requirements for the use of moving features. …. Verify that required data types are used for values of temporal position.

c) Reference:

d) Test Type: Basic

A.2 Application schemas for data with operations

a) Test Purpose: Verify that an application schema that supports operations on data satisfies the requirements of A.1.1, that it defines moving feature operations, and that it implements operations of moving features in compliance with specified requirements.

b) Test Method: Inspect the presentation of moving feature operations included in the application schema to ensure that they satisfy requirements. Inspect the presentation of the attributes of moving features included in the application schema to ensure that any operations of moving objects are implemented in compliance with requirements.

c) Reference:

d) Test Type: Basic

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.

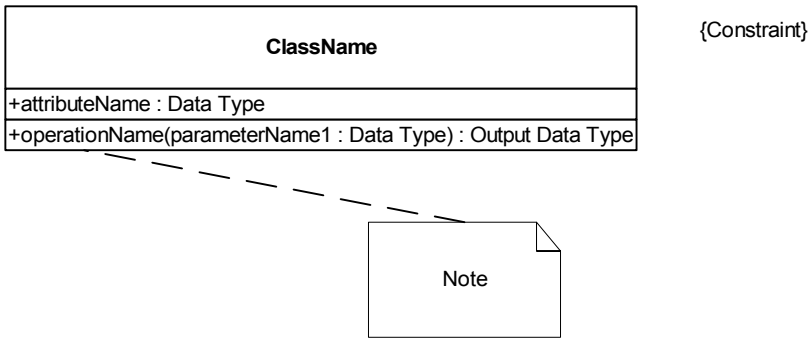

Figure B.1 — UML Class

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.

ISO 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 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 behavior of the type's operations and do not represent any actual implementation of state data.

<<Interface>> is a stereotype of class defined by ISO 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 19103 is a flexible enumeration that uses string values through a binding of the Dictionary type key and return values as string types; e.g. Dictionary (String, String). Code lists are useful to 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 the 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).

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

name is a character string. ISO 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

B.6 Constraint

A constraint specifies a semantic condition or restriction. Although ISO 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.

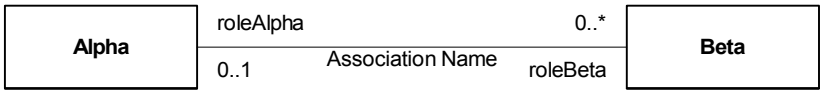

Figure B.2 — UML Associations

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 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.

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 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 B.3 are all valid. They have the following meanings:

- zero or one instance of Alpha may be associated with one instance of Beta,
- zero or more instances of Beta may be associated with one instance of Alpha,
- one and only one instance of Gamma may be associated with one instance of Delta,

- *n* being an integer number, *n* and only *n* instances of Delta may be associated with one instance of Gamma,
- *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,
- *n* being an integer number, n or more instances of Phi may be associated with one instance of Epsilon.

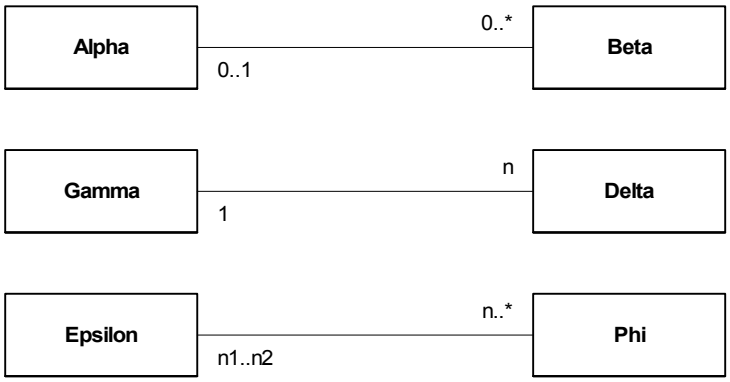

Figure B.3 — UML Mulitiplicity

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 Epsilon might use the vaklue of an attribute of Phi.

Figure B.4 — UML Mulitiplicity

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 can exist independently of the aggregation, and can be members of more than one aggregation.

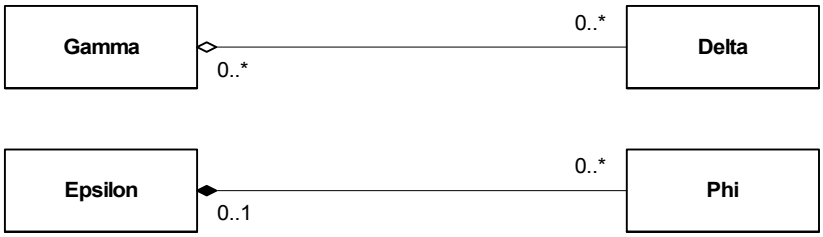

Figure B.5 — Aggregation and Composition

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 cannot exit independently of the composite class, nor can 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 8, Epsilon has a <<use>> dependency upon Phi.

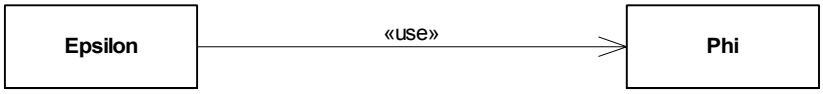

Figure B.6 — Dependency

B.15 Generalization

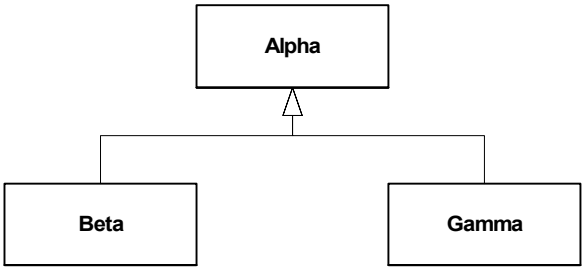

Figure B.7— UML Generalization

ISO 19501 defines generalization (Figure B.4) 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 implement 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.

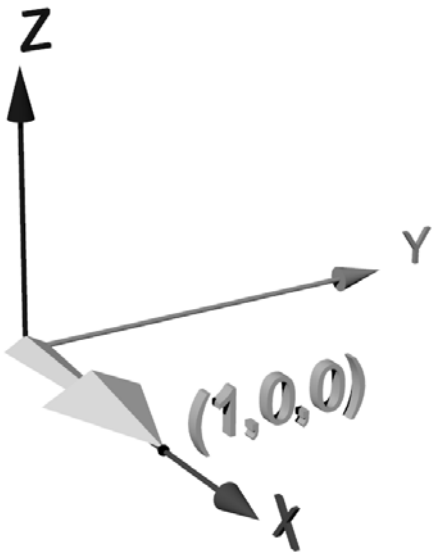

Figure C.1: Default orientation

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 center 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 [Savchen00] [Hearn97]. 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:

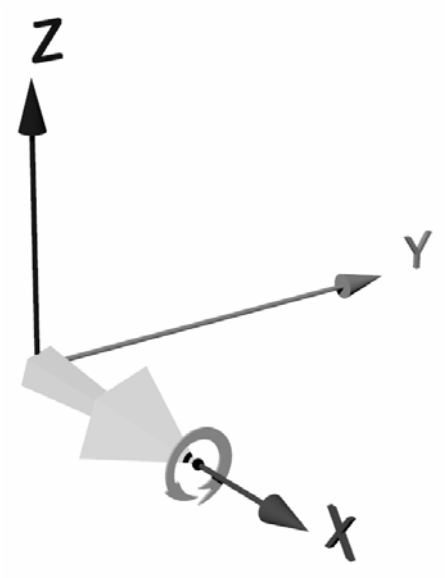

Figure C.2: X-axis rotation

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

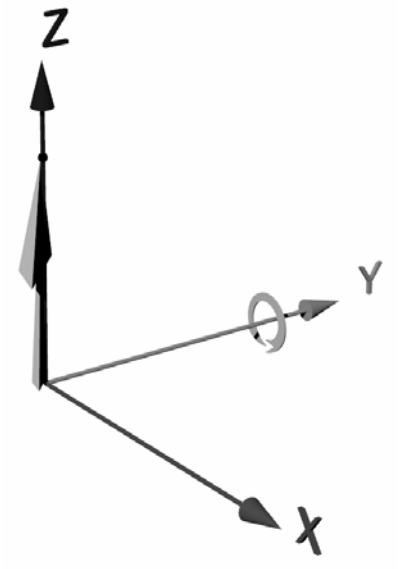

Figure C.3: Y-axis rotation

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!

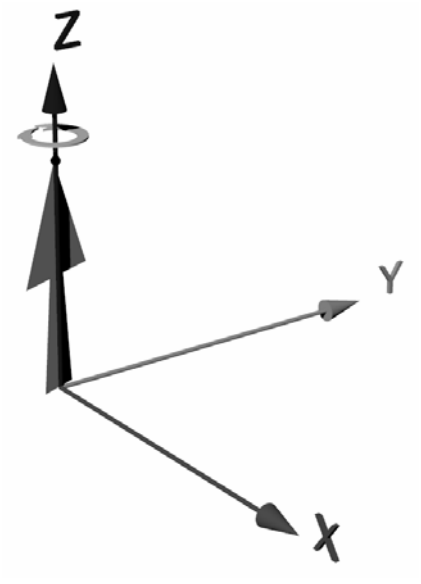

Figure C.4: Final Z-axis rotation

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 which 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 favor 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 [Shoemake85].) 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 [Bourg02]:

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

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

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

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

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." [Shoemake85] 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.

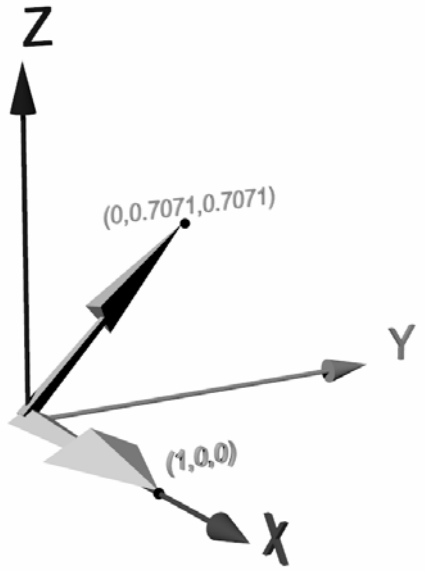

Figure C.5: Initial & final rotations

This orientation is represented by the following matrix (in 4x4 form):

0	0.707107	0.707107	0
-0.707107	0.5	-0.5	0
-0.707107	-0.5	0.5	0
0	0	0	1

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

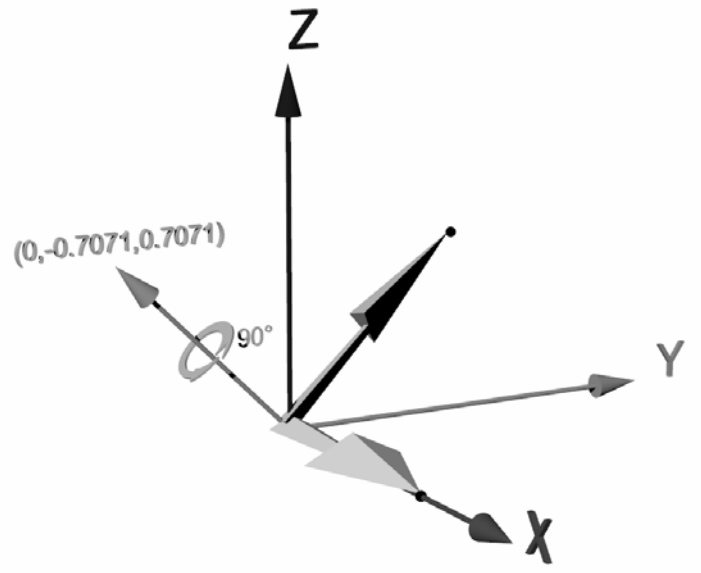

Figure C.6: Axis-angle representation

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:

$$Q_0 = \langle 0 \quad 0 \quad 0 \quad 1 \rangle \tag{25}$$

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

$$\mathrm{T} = 0 + 0.5 + 0.5 + 1 = 2 \tag{26}$$

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

$$X = \frac{mat[6] - mat[9]}{2\sqrt{T}} = \frac{-0.5 - (-0.5)}{2\sqrt{2}} = 0 \tag{27}$$

$$Y = \frac{mat[8] - mat[2]}{2\sqrt{T}} = \frac{-0.707107 - 0.707107}{2\sqrt{2}} = -0.5, \tag{28}$$

$$Z = \frac{mat[1] - mat[4]}{2\sqrt{T}} = \frac{0.707107 - (-0.707107)}{2\sqrt{2}} = 0.5, \tag{29}$$

$$W = \frac{\sqrt{T}}{2} = \frac{\sqrt{2}}{2} = 0.707107. \tag{30}$$

Calculating the values needed for the SLERP equation yields:

$$Q_0 \cdot Q_f = 0.707107 \tag{31}$$

$$\theta = \arcsin(0.707107) = \pi/4 = 45^{\circ} \tag{32}$$

$$Q_{ix} = \frac{Q_{0x} \sin((1\text{-}0.5)*45^{\mathrm{O}}) + Q_{fx} \sin(0.5*45^{\mathrm{O}})}{\sin(45^{\mathrm{O}})} = 0. \tag{33}$$

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

$$Q_i = \langle 0 \quad \text{-}0.2706 \quad 0.2706 \quad 0.9239 \rangle. \tag{34}$$

Converted to matrix form, the interpolated orientation matrix is:

$$\begin{bmatrix} 0.707107 & 0.5 & 0.5 & 0 \\ -0.5 & 0.853553 & 0.14645 & 0 \\ -0.5 & -0.14645 & 0.853553 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35}$$

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

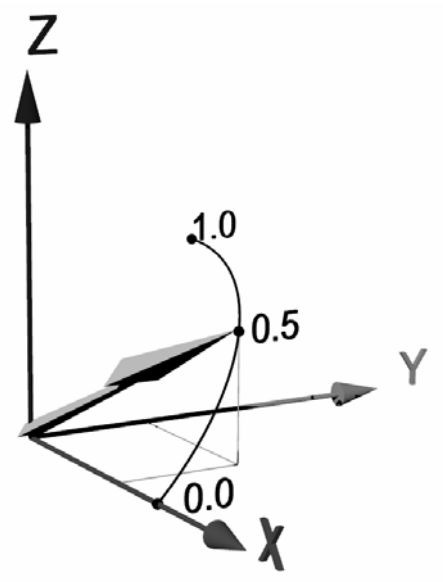

Figure C.7: Interpolated rotation

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 19123, *Geographic information — Coverage geometers and functions.*

[4] ISO 19133, *Geographic information — Location base services tracking and navigation.*

[5] ISO/IEC 19501-1:—[2)], *Information technology — Unified Modeling Language (UML) — Part 1: Specification.*

[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