OGC® DOCUMENT: 21-028
External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D021

Open Geospatial Consortium

OGC TESTBED-17: OGC API - MOVING FEATURES ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2021-11-19
Approval Date: 2021-12-09
Publication Date: 2022-01-18
Editor: Dean Younge

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Copyright notice

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

I. ABSTRACT vi

II. EXECUTIVE SUMMARY vi

III. KEYWORDS viii

IV. PREFACE ix

V. SECURITY CONSIDERATIONS x

VI. SUBMITTING ORGANIZATIONS xi

VII. SUBMITTERS xi

1. SCOPE 2

2. NORMATIVE REFERENCES 4

3. TERMS, DEFINITIONS AND ABBREVIATED TERMS 6
   3.1. Terms and definitions 6
   3.2. Abbreviated terms 9

4. CONVENTIONS 11
   4.1. Identifiers 11

5. INTRODUCTION 13

6. SCENARIOS AND ARCHITECTURE 16
   6.1. Scenario (Use Cases) 16
   6.2. Architecture 18

7. KEY FINDINGS 22
   7.1. Implications to OGC API Standardization 22

8. FUTURE WORK 24

9. BACKGROUND 27
   9.1. Existing relevant standards 27

10. DRAFT SPECIFICATION OF THE MOVING FEATURES API 39
   10.1. Requirements Class “Moving Features” 39

10.2. General Requirements 46

ANNEX A (INFORMATIVE) REVISION HISTORY 50

BIBLIOGRAPHY 52

LIST OF TABLES

Table 1 — Testbed-17 Moving Features Requirements Class 39
Table 2 — Moving Features Resources 39
Table 3 — Moving Features Operation requirement 40
Table 4 — Moving Features Response requirement 41
Table 5 — Moving Features Payload Recommendation 43
Table 6 — Table of the properties related to the moving feature 43
Table 7 — General HTTP-Response Requirements Class 46
Table 8 — Typical HTTP status codes 47
Table A.1 — Revision History 50

LIST OF FIGURES

Figure 1 — Moving Features task work items and deliverables. (Source: OGC Testbed-17 CFP) vii
Figure 2 — Moving Features task work items and deliverables 13
Figure 3 — Draft Moving Features Engineering Viewpoint 18
Figure 4 — Draft Moving Features Sequence diagram 19
Figure 5 — GM_Object from ISO 19107:2003 28
Figure 6 — General Feature Model from ISO 19109:2009 30
Figure 7 — Source JSON data example 31
Figure 8 — Logical Representation 31
Figure 9 — Spatial operators from ISO 19143 33
Figure 10 — Trajectory type from ISO 19141 35
Figure 11 — Temporal geometry from ISO 19141 36
Figure 12 — Example of filter extension for moving features 36
Figure 13 — Dynamic attribute from OGC 18-075 37
Figure 14 — Features Response Schema 41
Figure 15 — Features Example 41
Figure 16 — MovingFeature Response Schema 44
Figure 17 — MovingFeature Example 45

Figure 18 — HTTP Response Schema.....46

I ABSTRACT

The OGC Testbed-17 Moving Features thread conducted an interoperability feasibility study that examined specific scenarios that could be supported by a Moving Features Application Programming Interface (API). The use cases considered tracking objects based on motion imagery, analytical processing and visualization. This Engineering Report presents a specification of a prototype Moving Features API, that could serve as the foundation for a future draft OGC API — Moving Features standard.

II EXECUTIVE SUMMARY

Digital representation of moving objects is gaining increasing interest across industry, from both the Observations and Modeling perspectives. An increasing number of scenarios requires expanding three-dimensional (3D) geospatial environments with time and linking data representing same or related objects. Due to its character, it may require a specific approach to the information representation. There are abstract models of the International Organization for Standardization (ISO) available and data encodings within OGC Standards that address these scenarios. The understanding of the moving features idea changes as we find more sophisticated scenarios with new challenges and reveal the need to aggregate, link and represent specific information.

OGC is continuing its efforts to simplify the use of geospatial standards, proposing a suite of modern OGC API Standards. The idea is to continue the proven approach that enabled hundreds of the applications already exploiting legacy OGC Standards to easily use geospatial data. New Web APIs are modern in terms of best practices, protocols, encodings and the definition of tailored solutions.

Previously, Testbed-16 explored technologies to transform detections of moving objects reported using motion imagery standards (e.g. MISB Std. 0903) into the model and encoding defined in the OGC Moving Features Standard (OGC 18-075). That work suggested the following notional workflow:

a) Extract moving object detections from the motion imagery stream

b) Encode the detections as moving features

c) Correlate the detection of moving features into tracks of moving features

d) Perform analytics to enrich and exploit the tracks of moving features

The Testbed-16 work is documented in the Testbed-16 Full Motion Video to Moving Features Engineering Report (OGC 20-036). That work motivated the OGC Moving Features Standard Working Group (SWG) to update their Charter to include work on an OGC API Standard

dedicated to objects on the move. The Testbed-17 participants worked closely with the MF SWG to ensure that all efforts were coordinated accordingly.

To support these efforts, the OGC Testbed-17 Moving Features thread conducted an interoperability feasibility study that examined specific scenarios that could be supported by a hypothetical Moving Features API. The use cases considered tracking objects based on motion imagery (video recordings in practice), analytical processing and visualization. Real-life visual and hybrid visual-lidar detection of objects presents challenges related to object identification, tracking and inferencing.

Purpose of this document

This document is one of the two Testbed-17 Engineering Reports (D021) that focus on the specification of a hypothetical OGC Moving Features API that could be used to implement an agreed scenario. The document presents the considerations and background to the development of the API, as well as the proposed normative clauses containing requirements that could be included in the future **OGC API — Moving Features** standard.

For the detailed description of the experiment itself and technical challenges related to objects tracking and use cases one should see the OGC Testbed-17 D020 Moving Features Engineering Report (OGC 21-036).

Architecture

The experiment that grounded this ER and the Testbed-17 Moving Features thread consisted of the components depicted on the figure below.

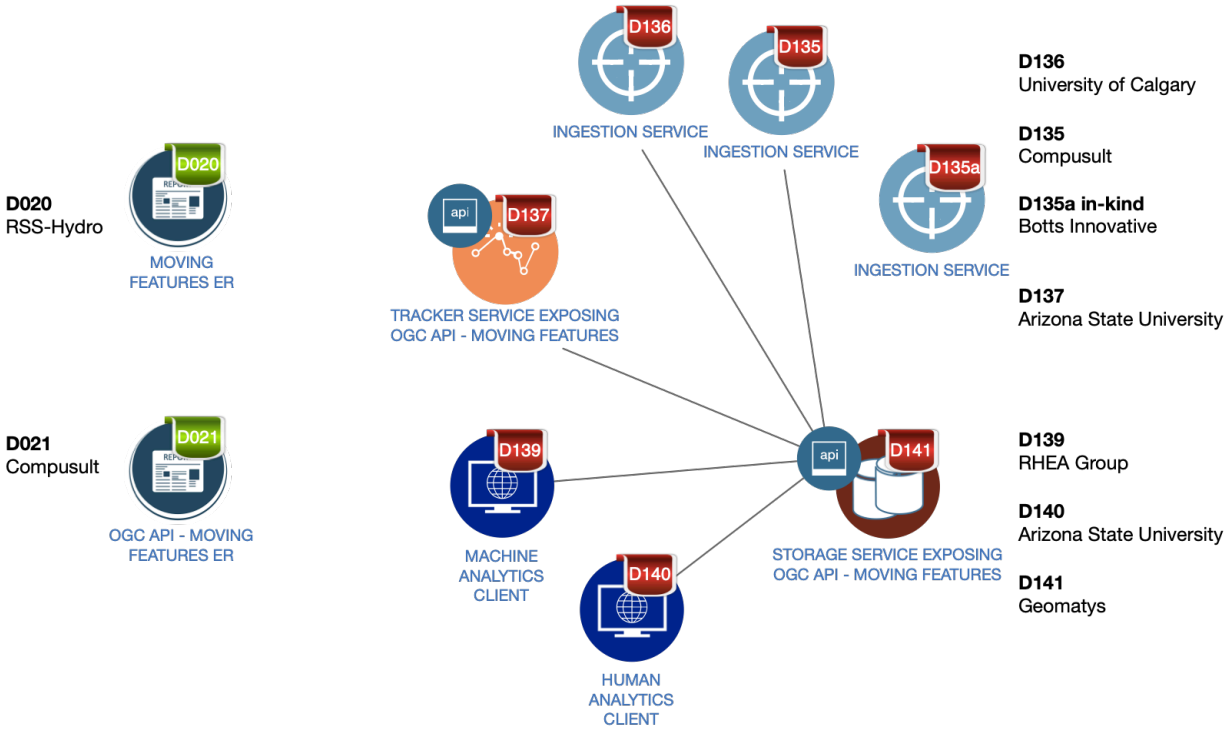

Figure 1 — Moving Features task work items and deliverables. (Source: OGC Testbed-17 CFP)

- D136 Ingestion Service — Software component that processes raw data from an observing system, extracts detections and feed the storage with Sensor Things API (STA) data. OGC TB-17: Moving Features ER: Section 7 (OGC 21-036)
- D135 Ingestion Service — component similar to D136, but not linked to the storage during the experiment. OGC TB-17: Moving Features ER: Section 8 (OGC 21-036)
- D137 Tracking Service — software component that correlates detections and tracklets into longer tracks. Those tracks are then exported as OGC Moving Features to the Storage Service via an interface conforming to the new OGC API — Moving Features. OGC TB-17: Moving Features ER: Section 9 (OGC 21-036)
- D139 Machine Analytics Client — Client component that provides OGC Moving Feature analytics and annotation. The component generates second level information based on the aggregate tracks that characterize the clusters, dwell times etc. OGC TB-17: Moving Features ER: Section 11 (OGC 21-036)
- D140 — Human Analytics Client — Client tool with user interface that allows a user to analyse the unit and aggregated tracking data like particular objects, space distributed histograms. OGC TB-17: Moving Features ER: Section 11 (OGC 21-036)
- D141 Storage Service — central service storing and exposing data from any other components. It provides transformations of the SensorThings API (STA) data into content for the OGC Moving Features API. Storage service hosts both request-response query endpoint as well as the messaging brokering mechanism. OGC TB-17: Moving Features ER: Section 12 (OGC 21-036)
- D020 — Engineering Report describing the experiment in details and general conclusions. OGC TB-17: Moving Features ER (OGC 21-036)
- D021 — herein Engineering Report focused on the API's formal requirements and background.

III KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, Moving Features, API

IV PREFACE

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

V SECURITY CONSIDERATIONS

No security considerations have been made for this document.

VI SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Compusult Limited

VII SUBMITTERS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Dean Younge	Compusult Limited	Editor
Martin Desruisseaux	Geomatys	Contributor
Piotr Zaborowski	Open Geospatial Consortium	Contributor
Guilhem Legal	Geomatys	Contributor
Sepehr Honarparvar	University of Calgary	Contributor
Steve Liang	University of Calgary	Contributor
Brad Miller	Compusult Limited	Contributor
Jason MacDonald	Compusult Limited	Contributor
Sizhe Wang	Arizona State University (ASU)	Contributor
Zhining Gu	Arizona State University (ASU)	Contributor
Rob Smith	Away Team Software	Contributor
Angie Carrillo	RHEA Group	Contributor
Guy Schumann	RSS-Hydro	Contributor

NAME	ORGANIZATION	ROLE
Chuck Heazel	Heazel Technologies	Contributor

1

SCOPE

1 SCOPE

This OGC Engineering Report (ER) is deliverable D021 of the OGC Testbed-17 (TB-17) initiative performed as an activity of the OGC Innovation Program.

The goal of this document is to combine the work of the Moving Features (MF) Standards Working Group (SWG) and the results obtained from TB-17 demonstration to define a draft OGC API specification. This work is the initial step of defining an API for discovery, access, and exchange of moving features and their corresponding tracks such that implementations of the API can be applied in a near real-time scenario.

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

OGC 19-045r3 — OGC Moving Features Encoding Extension — JSON 1.0

Hideki Hayashi, Akinori Asahara, Kyoung-Sook Kim, Ryosuke Shibasaki, Nobuhiro Ishimaru: OGC 16-120r3, *OGC Moving Features Access*. Open Geospatial Consortium (2017). http://docs.opengeospatial.org/is/16-120r3/16-120r3.html

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r3, *OGC API — Features — Part 1: Core*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

Clements Portele, Panagiotis (Peter) A. Vretanos: OGC 18-058, *OGC API — Features — Part 2: Coordinate Reference Systems by Reference*. Open Geospatial Consortium (2020). https://docs.ogc.org/is/18-058/18-058.html

ISO: ISO 19141:2008, *Geographic information — Schema for moving features*. International Organization for Standardization, Geneva (2008). https://www.iso.org/standard/41445.html

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

3.1. Terms and definitions

3.1.1. **Application Programming Interface (API)**

a standard set of documented and supported functions and procedures that expose the capabilities or data of an operating system, application or service to other applications (adapted from ISO/IEC TR 13066-2:2016)

3.1.2. **duration**

the difference between the ending time and the starting time of the trajectory (including the dwell time)

3.1.3. **dwell time**

corresponds to the time that a moving object is stationary

3.1.4. dynamic attribute

characteristic of a feature in which its value varies with time [OGC 16-140]

3.1.5. feature

abstraction of real world phenomena [ISO 19109]
Note: A feature can occur as a type or an instance. Feature type or feature instance should be used when only one is meant.

3.1.6. feature attribute

characteristic of a feature [ISO 19109]

3.1.7. feature table

table where the columns represent feature attributes, and the rows represent features [OGC 06-104]

3.1.8. geographic feature

representation of a real world phenomenon associated with a location relative to the Earth [ISO 19101-2]

3.1.9. geometric object

spatial object representing a geometric set [ISO 19107:2003]

3.1.10. **moving feature**

A representation, using a local origin and local ordinate vectors, of a geometric object at a given reference time (ISO 19141:2008). In the context of this ER, the geometric object is a feature, which is an abstraction of real world phenomena (ISO 19109:2015).

3.1.11. **track**

the entire trajectory of a specific moving object.

3.1.12. **tracklet**

a fragment of the track followed by a moving object.

3.1.13. **property**

facet or attribute of an object referenced by a name [ISO 19143]

3.1.14. **trajectory**

path of a moving point described by a one parameter set of points (ISO 19141:2008).

3.1.15. **trajectory time**

total time a moving object takes to complete a specific trajectory

3.2. Abbreviated terms

API	Application Programming Interface
CSV	Comma Separated Values
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
UML	Unified Modeling Language
XML	Extensible Markup Language
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional

4

CONVENTIONS

4 CONVENTIONS

This section provides details and examples for any conventions used in the document.

4.1. Identifiers

The normative provisions in this specification are denoted by the URI

http://www.opengis.net/spec/ogcapi-mf-1/1.0

All requirements, permissions, recommendations and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

5

INTRODUCTION

5 INTRODUCTION

The Testbed 17 Call for Participation (CFP) identified the following high-level requirements for defining the Moving Features API:

a) **Detection ingest:** This component ingests data from a moving object detection system, extracts detections and partial tracks (tracklets), and exports the detections and tracklets as OGC Moving Features. These detections and tracklets may also be exported as SensorThings "Things".

b) **Tracker:** This component ingests detections and tracklets as OGC Moving Features, then correlates them into longer tracks. Those tracks are then exported as OGC Moving Features.

c) **Data Store:** Provides persistent storage of the Moving Feature tracks.

d) **Machine Analytics:** Software which enriches the existing tracks and/or generates derived information from the tracks.

e) **Human Analytics:** Software and tools to help users exploit the Motion Imagery tracks and corresponding detections or correlated tracks. For example, a common operational picture showing both static and dynamic features.

NOTE: Several of these interfaces were defined as SensorThings APIs as identified in Figure 3 of the Architecture.

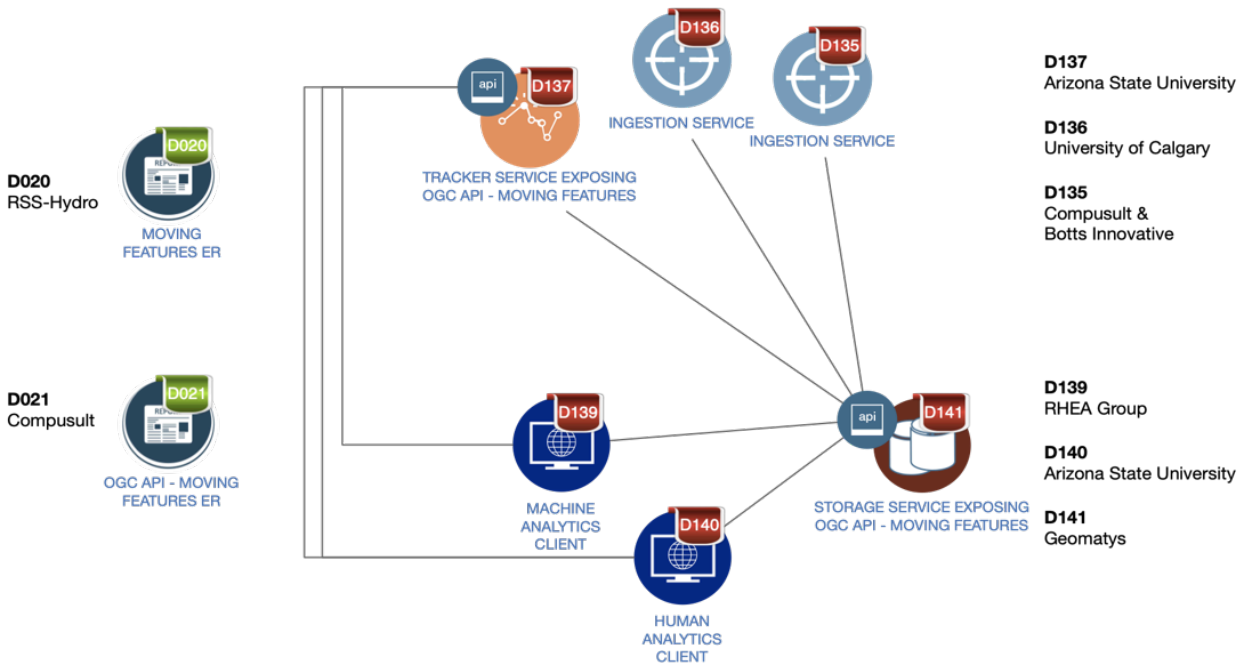

Figure 2 — Moving Features task work items and deliverables

Section 7 provides the background of the testbed and cites existing relevant standards and previous work.

Section 8 describes a draft specification for the Moving Features API.

6

SCENARIOS AND ARCHITECTURE

6 SCENARIOS AND ARCHITECTURE

6.1. Scenario (Use Cases)

For Testbed-17, there were three scenarios weighed for consideration:

- Bus Tracking
- Autonomous vehicle tracking
- Hazardous Materials (HAZMAT) vehicle tracking

This ER describes the detection and tracking of moving buses in front of a school. The availability of data to support this scenario was the main factor in the selection of Bus Tracking as a primary scenario. The video was acquired by a light-weight Unmanned Aerial Vehicle (UAV) or drone, courtesy of the University of Calgary.

A separate autonomous vehicle use case was analyzed to detect and track people and vehicles moving nearby with WebVMT, a lightweight open format designed to synchronize video with an animated, annotated map on the web. Video and lidar data were captured from a moving StreetDrone vehicle and provided courtesy of Ordnance Survey.

6.1.1. Tracking of Buses

In the framework of the Testbed-17 MF thread, a use case was selected where trajectories are represented by school buses at a school parking lot while students are getting on or off the buses.

The project's raw data was composed of two videos of the same scenario, one recorded with a drone and the other one recorded with a GoPro action camera. The input data for the Machine Analytics Client was the output data from the Tracking Service which was stored and provided by the storage service.

The steps in this scenario included:

a) Detect objects in each frame and track them in consecutive frames.

b) Transform objects' locations from the video space to geographic space.

c) Link object detections in consecutive frames as a single moving object. Smaller incomplete series of detections are classified as tracklets, while a complete series of detections are classified as the track.

d) Send collected moving object's data to a storage service.

e) Analyze data retrieved from the storage service to generate information such as moving feature distribution, average trajectory time, dwell time, detection of moving feature clusters and detection of trajectory clusters.

6.1.2. Autonomous vehicle use case

Following on from work in the Testbed-16 initiative, Web Video Map Track (WebVMT) has been used to previzualize and analyze multi-sensor data from an autonomous vehicle in the Testbed-17 initiative.

Ordnance Survey provided video and lidar data from a StreetDrone vehicle navigating the roads around their headquarters in Southampton, UK. The footage was shot from a front-facing camera on the vehicle and shows a number of moving objects including a cyclist, pedestrians and other road vehicles as well as static objects such as hedges, road signs and lamp posts.

a) **Tracking Vehicles:** If we could detect and track the oncoming vehicles in the StreetDrone videos, this would produce a set of short tracklets which correspond to different moving (or stationary) features. Being able to track the stationary cars in the car park would also be useful to classify them correctly as parked vehicles, which is easy for humans but trickier for a computer as it's the camera that's moving rather than the vehicle itself. Tracking moving features from a moving camera is a particularly useful capability for autonomous vehicles and dashcams.

b) **Tracking Dangers:** If we could detect and track the cyclist in the second StreetDrone video, this would produce longer tracklets as the bike stays in shot for a while, though they pass behind a couple of lamp posts around 0:50. There's also a pedestrian at about 1:25. Being able to track cyclists and pedestrians would also be a useful capability for autonomous vehicles and dashcams.

c) **Associating Views:** We could extend these ideas to multiple cameras by using the BikeBro video which includes two views and avoids synchronization issues by baking both into a single video frame. If we could detect and track moving vehicles from the front and rear views, the resulting tracklets could be (trivially) paired together to form longer tracks for the passing vehicles. This capability enables autonomous vehicles to track other vehicles moving around them on roads with multiple lanes as the same vehicle could appear in different camera views at different times and differ in shape due to the different view angles.

All of these capabilities are also useful for the traffic camera use case, which would enable roadside camera footage to be aggregated with dashcam video to track vehicle movements.

6.1.3. Windsor Scenario

6.1.3.1. Objective

Illustrate the ability to track and map the location of the HAZMAT vehicle as reported real-time.

6.1.3.2. Overview

The Critical Infrastructure Protection Initiative (CIPI-1) demonstration scenario involved a leaking hazmat truck which was heading from Windsor over to Detroit. The challenge was to track the truck, project where it was going, plan interception, and assess the impact of released HAZMAT at all points of the route.

6.2. Architecture

The following figure provides the engineering viewpoint utilized during the initial T17 Moving Features discussions.

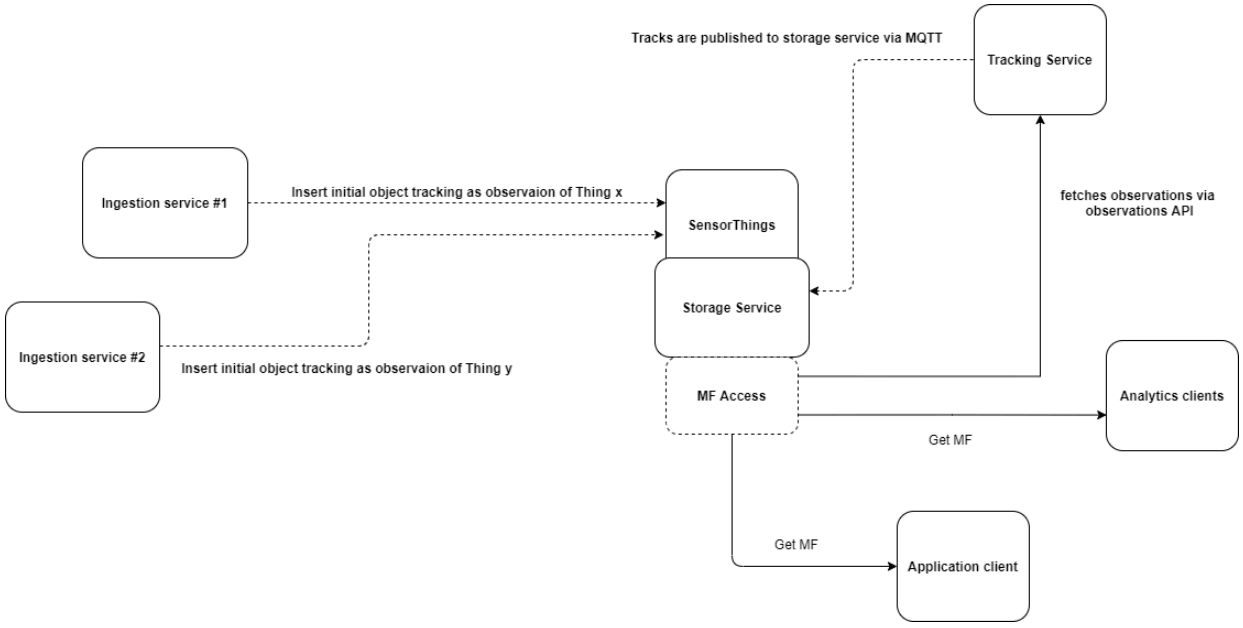

Figure 3 — Draft Moving Features Engineering Viewpoint

The sequence diagram in Figure 4 provides the component interactions from the initial data ingestion to retrieval by a moving feature client.

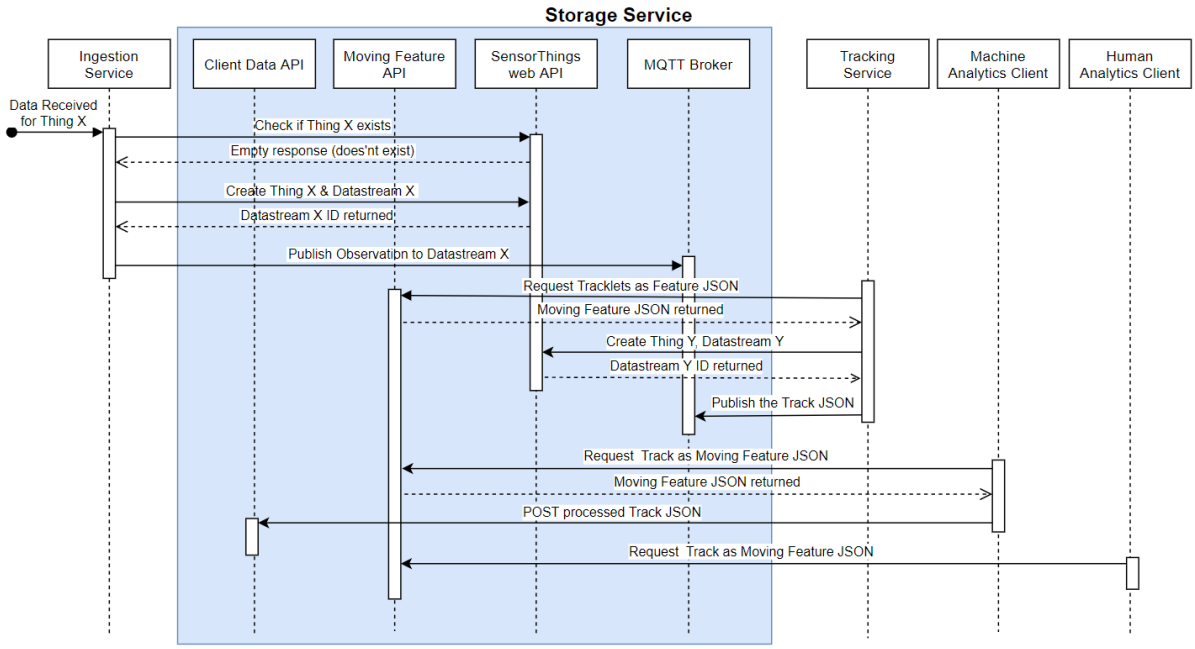

Figure 4 — **Draft** Moving Features Sequence diagram

6.2.1. Storage service

The storage service receives and returns moving features as JSON objects. The current format does not conform to OGC Moving Features JSON encoding (OGC 16-140r1), but this shortcoming is due to a lack of time in implementation development rather than a problem with the JSON encoding.

JSON objects are not stored "as-is". Instead, the objects are deconstructed. Each feature type is represented by a database table and each property such as `start_time`, `duration`, *etc.* is extracted and stored in a column of the feature table. This data organization is described in OGC 06-104 (Simple Features SQL). An inconvenience is that all features and properties must be known in advance since they are hard-coded both in the Java code reading JSON objects and in the database schema (note: a dynamic approach is possible but has not been implemented in this testbed). But advantages of this approach are that:

- A database index can be applied on those property values,
- The full power of SQL expressions is available for data manipulation,
- Data can more easily be exported to other formats such as netCDF (OGC 16-114r3).

For this testbed, a PostgreSQL database was used. The bus coordinates were stored in a geometry column using the PostGIS extension. It would have been possible to go one step further by using the MobilityDB extension, which would allow storing the "moving geometries" instead of the static geometries in a PostgreSQL database, however, that moved to future development.

Queries can be done using either an implementation of OGC API — Features (OGC 17-069r3) or by using a Common Query Language (CQL) query. CQL supports the use of more advanced filtering conditions such as “features at a distance closer than X from geometry Y”. The storage service can parse CQL and translate it to a mix of SQL and Java code. The SQL statement uses instructions such as `ST_Within` or `ST_Intersects`. A spatial database will use its index for efficient execution of those statements. If some parts of the query cannot be translated to SQL, then the storage service will execute them in Java code on the remaining features after the initial filtering done by the database.

The set of operations defined by the OGC/ISO Filter Encoding standard (ISO 19143) and SQLMM standard assumes static geometries. For example, the above-cited example “features at a distance closer than X from geometry Y” does not take time into account. If geometries are trajectories, that operation will check the shortest distance between trajectories regardless of whether the moving features were at those locations at the same time or not. The ISO 19141 (Moving Features) standard defines new operators such as `nearestApproach` which could be used in CQL expressions. There is no OGC standard defining a filter encoding for moving features operations, but this testbed explored what they may look like (without reaching the point of experimenting with it in the storage implementation).

7

KEY FINDINGS

7 KEY FINDINGS

7.1. Implications to OGC API Standardization

OGC API Standards enable access to resources using the HTTP protocol and its associated operations (GET, PUT, POST, etc.) OGC API — Common defines a set of capabilities which are applicable to all OGC APIs. Other OGC API Standards extend OGC API — Common with functionality specific to a resource type.

8

FUTURE WORK

8 FUTURE WORK

In order to perform more sophisticated queries beyond those by ID or bounding box, the testbed participants recommend implementing OGC API — Features — Part 3: Filtering and the Common Query Language (CQL).

It is also recommended to investigate using the MobilityDB extension of PostgreSQL, which will allow storing "moving geometries" instead of static geometries in a PostgreSQL database. An interesting aspect of MobilityDB is their definition of new operators derived from ISO 19141. MobilityDB is ahead of current standardization state since the use of Moving Feature operators in SQL expressions is not yet backed by an OGC/ISO standard. This is in contrast with operators such as "distance" which are first defined in an abstract specification (ISO 19107: spatial schema), then "implemented" by other OGC/ISO standards in filter expressions (ISO 19143), SQL statements (SQLMM), CQL expressions, etc.

In the Moving Feature case, the operators are only defined in the abstract specification. They have not yet been propagated to the Filter Encoding, CQL nor SQLMM standards. This situation is described in more detail in the "background" section below. This testbed has made good progress toward experimenting with Moving Feature operators in filter expressions, but without completion. Future work could continue with this experiment and, if revisions of relevant OGC Standards are available, test those revisions using MobilityDB, Java or other implementations.

Additionally, detailed future work and recommendations are provided in OGC Testbed-17 D020 Moving Features Engineering Report (OGC 21-036). A summary of these include:

a) Ingestion service

1. Update the framework to support stateless SensorThings.
2. Update Ingestion service to temporarily store observations locally.
3. Test and implement an ingestion service to handle both fixed and moving camera observations.
4. Improving the transformation accuracy.
5. Automation of registering Things based on the stateless STA data model would help to get data from new sources.

b) Machine analytics client

1. Segmentation of detected Moving Features
2. Conduct Seasonality Analysis

c) Autonomous vehicle use case

1. combining multi-sensor data to improve detection accuracy and cognitive guidance
2. data aggregation to associate segmented trajectories from spatially-distributed sensors
3. autonomous location reporting of obstructions to moving objects;
4. combination of multi-angle sensor data to track nearby vehicles and obstacles;
5. use of spatially-distributed data for pre-emptive responses.

9

BACKGROUND

9 BACKGROUND

Testbed-16 explored technologies to transform detections of moving objects reported using motion imagery standards (MISB Std. 0903) into OGC Moving Features (OGC 18-075).

In that testbed activity, the ability to extract Motion Imagery derived Video Moving Target Indicators (VMTI) from an MPEG-2 motion imagery stream and represented as OGC Moving Features was demonstrated.

That work suggested a notional workflow:

a) Extract moving object detections from the motion imagery stream.

b) Encode the detections as moving features.

c) Correlate the detection of moving features into track moving features.

d) Perform analytics to enrich and exploit the tracks of moving features.

This work is documented in the Testbed-16 Full Motion Video to Moving Features Engineering Report (OGC 20-036).

9.1. Existing relevant standards

This Testbed-17 thread is grounded in the following OGC/ISO Standards. The geometry concept is presented first, followed by the feature concept. Note that in this document, a feature is *not* considered as a geometry. Instead, a feature *contains* a geometry as one of its attributes. A feature also contains non-geometric attributes such as the bus color.

9.1.1. Geometry (ISO 19107)

The ISO 19107, *Geographic information – Spatial schema* standard defines a `GM_Object` base type which is the root of all geometric objects. A `GM_Object` instance can be regarded as an infinite set of points in a particular coordinate reference system. The `GM_Object` base type has various subtypes such as `GM_Point`, `GM_Curve`, `GM_Surface` and `GM_Solid`. The UML below shows the `GM_Object` base type with its operations, for example `distance(…)` for computing the distance between two geometries. All those operations assume static objects, without time-varying coordinates or attributes. For example the `distance(…)` operator does not take in account whether the two objects were *at the same time* at the location where shortest distance is found.

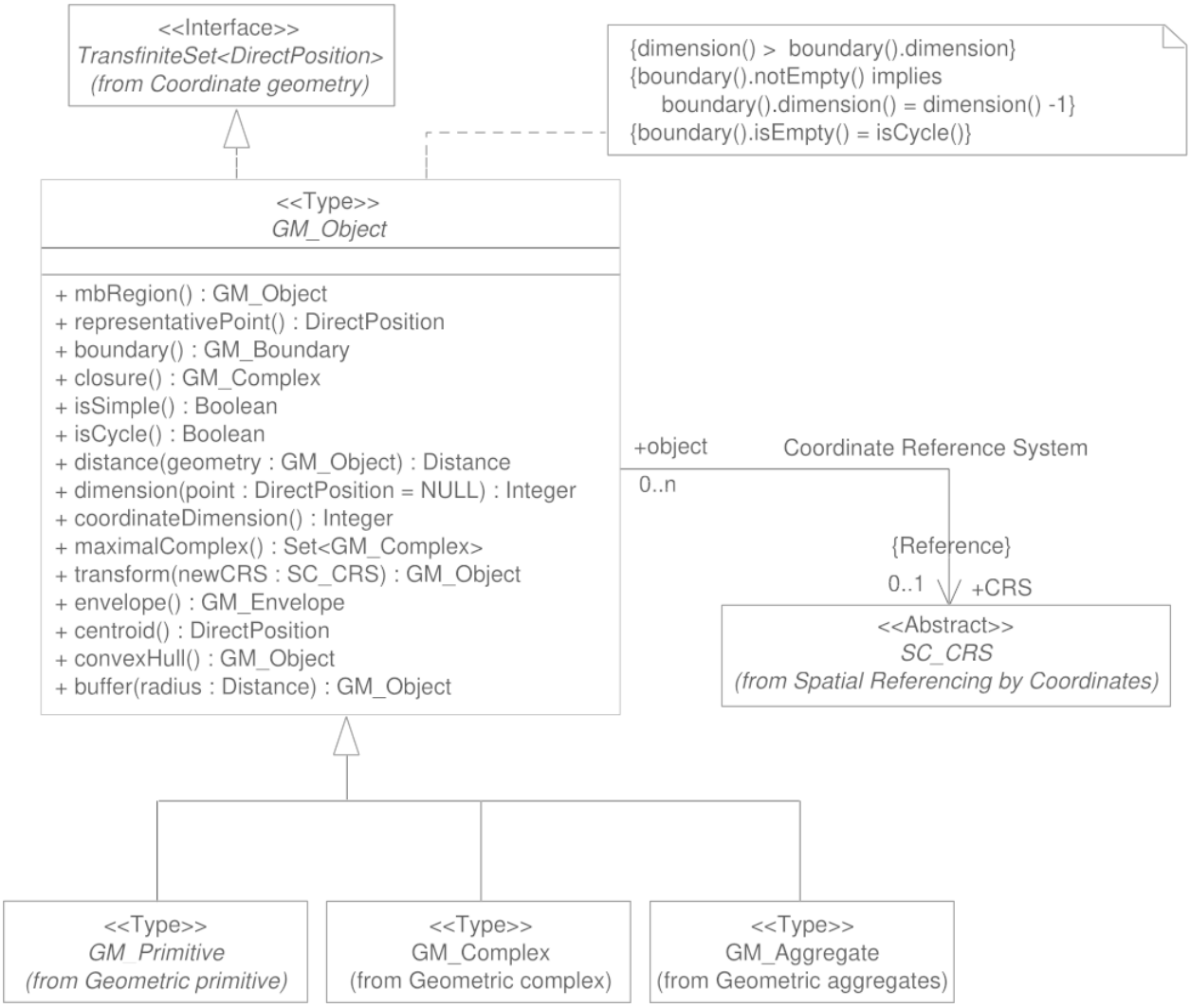

Figure 5 — GM_Object from ISO 19107:2003

Geometry, topology and temporal-objects (`GM_Object`, `TP_Object`, `TM_Object`) are not features. These types can provide types for feature *properties*, but cannot be specialized to feature types. Feature types and properties are described in the next section.

9.1.2. Features (ISO 19109)

The ISO 19109, *Geographic information — Rules for application schema* standard defines types for the definition of features. A feature is an abstraction of a real-world phenomenon. The terms “feature type” and “feature instance” are used to separate the following concepts of “feature”:

Feature type	The whole collection of real-world phenomena classified in a concept. For example, the “bridge” feature type is the abstraction of the collection of all real-world phenomena that is classified into the concept behind the term “bridge”.

Feature instance	A certain occurrence of a feature type. For example, “Tower Bridge” feature instance is the abstraction of a certain real-world bridge in London.
NOTE 1:	In object-oriented modeling, feature types are equivalent to classes and feature instances are equivalent to objects. The feature properties (presented below) are equivalent to fields.
Feature type instance	The UML shown below contains a subtlety explained here but ignored for simplicity in the rest of this document: `FeatureType` is defined as a metaclass, i. e. a class for describing other classes. A *feature type instance* (not to be confused with a *feature instance*) is a class that represents an individual feature type. For example, “bridge” and “park” are two instances of `FeatureType`. Then “Tower Bridge” and “Golden Gate Bridge” are two `Feature` instances of the “bridge” `FeatureType` instance, and “Jardin des Tuileries” is a `Feature` instance of the “park” `FeatureType` instance.
NOTE 2:	The assertion that `FeatureType` is a metaclass means that in statically typed languages such as Java or C++, a `FeatureType` instance can generally not be represented directly as a Java or C++ class (unless the class properties are known at compile-time). Developers in those languages have to use more indirect mechanisms such as reflections or dictionaries (hash maps). Dynamic languages such as Python can use feature type instances more directly, but at the cost of type safety in the general case.

The UML class diagram in Figure 6 shows the ISO 19109 General Feature Model. A `FeatureType` contains the list of properties (attributes, associations and operations) that `Feature` instances of that type can contain. In other words, a feature can be seen as a collection of property values. Geometries are feature properties (more precisely attributes) like any other, without any special treatment. All properties shown below are assumed static, without time-varying values.

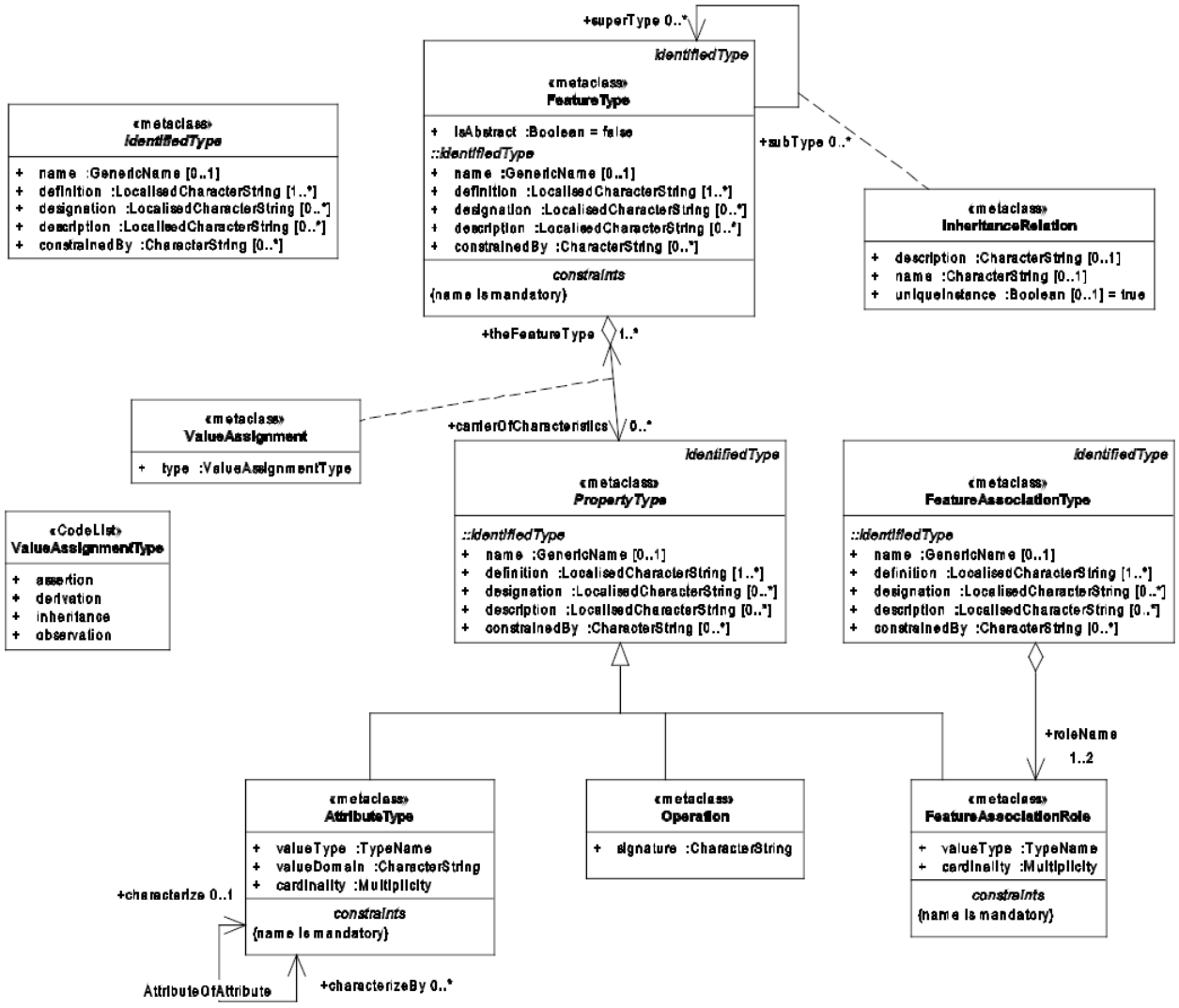

Figure 6 — General Feature Model from ISO 19109:2009

9.1.2.1. Usage in Testbed 17

The UML class diagram above defines the GeoAPI interfaces in the `org.opengis.feature` package (development branch not yet released). The storage service in this testbed uses those interfaces as the "universal" internal representation of features. The service supports the transfer of features between JSON, netCDF and database representations, together with operations on features. Of note, the UML shown in Figure 6 represents meta-classes and not classes, providing information on features in much the same way that metadata describes data.

The following provides a simplified view of a feature in JSON format. The UML model on the right-hand side of Figure 7 describes the JSON on the left-hand side.

Source JSON data (simplified)

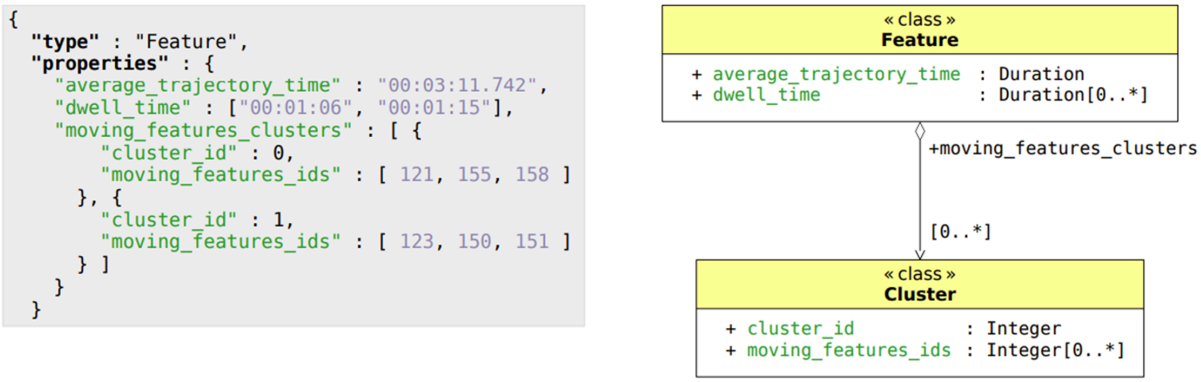

Figure 7 — Source JSON data example

The following is an example logical presentation of the feature and its associated attributes. This approach of classifying features provides a method of making a dynamic feature within the data model.

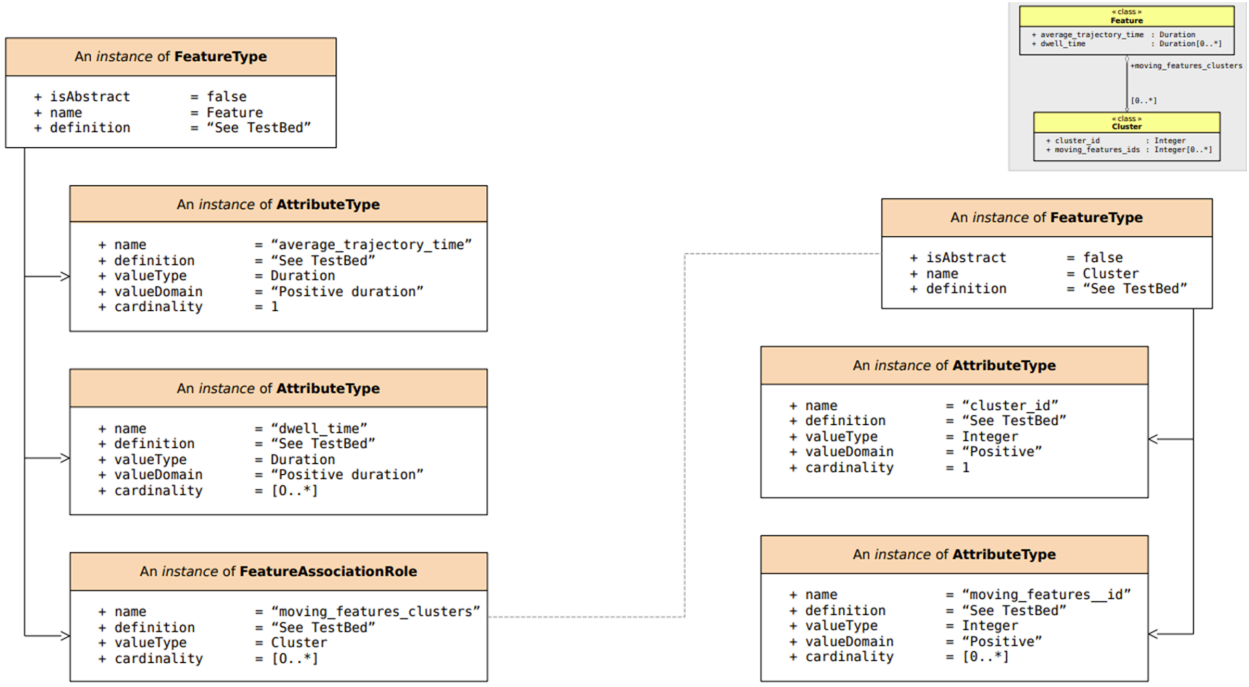

Figure 8 — Logical Representation

9.1.3. Simple Features SQL (OGC 06-104)

The Simple Feature Access — Part 2: SQL Option standard describes a feature access implementation in SQL based on a profile of ISO 19107. This standard defines *feature table* as a table where the columns represent feature attributes, and the rows represent feature

instances. The geometry of a feature is an ordinary feature attribute, i.e. a single column in the table. In spatial databases, the geometry columns have a geometry type such as `ST_Point` or `ST_Polygon`; in databases without spatial support, an alternative approach described by the standard can be used.

9.1.3.1. Usage in Testbed 17

The storage service organizes data according to the requirements stated in the "Simple Feature SQL" standard. Moving features received in JSON format are not stored verbatim, but are based on their properties. Each feature type is instantiated by a database table, then each attribute value is stored in a table column. Associations to other features are stored as foreign keys.

An inconvenience of this approach is that the set of feature types and their properties must be known in advance in order to define the database schema (a more dynamic approach is possible but has not been implemented in this testbed). Two advantages of this approach are that 1) efficient SQL queries can be used as described in the filter encoding section, and 2) the feature instances can be reconstructed and exported in various formats such as JSON, XML, CSV or netCDF.

9.1.4. Filter Encoding (ISO 19143)

The ISO 19143, *Geographic information – Filter encoding* standard (also an OGC Standard) provides types for constructing queries. These objects can be transformed into a SQL "SELECT … FROM … WHERE … ORDER BY …" statement to fetch data stored in a SQL-based relational database. Similarly, the same objects can be transformed into an XQuery expression in order to retrieve data from XML documents. The UML in Figure 9 below shows the objects used for querying a subset based on spatial operations such as "contains" or "intersects".

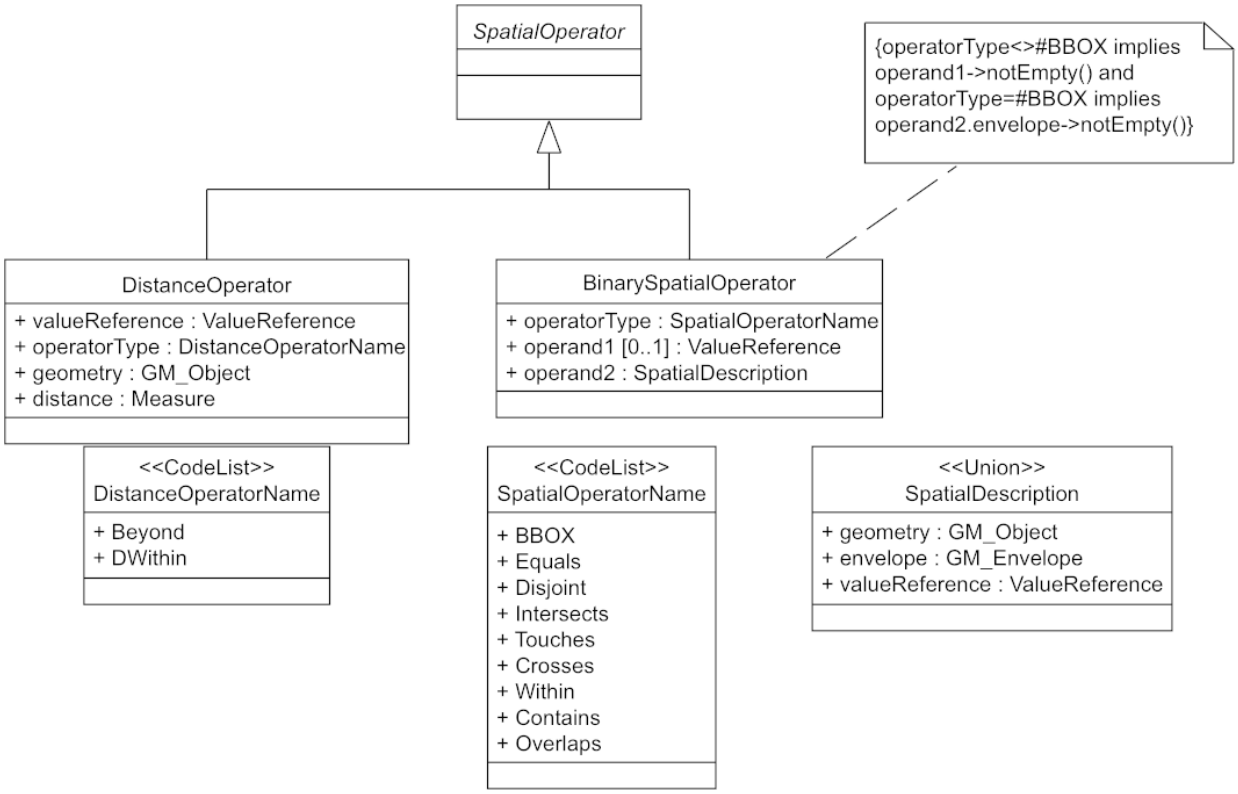

Figure 9 — Spatial operators from ISO 19143

All filters defined in ISO 19143 are static. There is no operator taking movement into account. New operators for moving features are proposed in another section of this ER.

In addition to the abstract model, the ISO 19143 standard also defines how to encode the above objects in XML or CQL formats. The abstract model shown in Figure 9 helps developers to identify which concepts are common to all formats, which makes it easier to implement conversions to SQL or XPath.

9.1.4.1. Usage in Testbed 17

The classes illustrated in Figure 9 are implemented in the GeoAPI interfaces in `org.opengis.filter` package (development branch not yet released). The storage service in this testbed uses those interfaces as the "universal" internal representation of filters. Queries expressed in Common Query Language (CQL) are converted internally into objects defined by above UML. Then those objects are converted either in SQL statements or in Java code, depending on whether the query is supported by the spatial database or not.

9.1.5. OGC API — Features — Part 1 (OGC 17-069r3)

The OGC API — Features — Part 1: Core (OGC 17-069r3) standard specifies the fundamental building blocks for interacting with features using a Web API. This base standard allows a client

application to either get all features from a collection available on a server, to filter by feature properties, or to get feature instances by their identifier.

9.1.5.1. Usage in Testbed 17

The following example gets a feature instance by its identifier (where “215” can be replaced by any other valid identifier):

Example 1: https://tb17.geomatys.com/API/feature/collections/Observation/items/215

The following example gets all feature instances intersecting the given bounding box (BBOX).

Example 2: https://tb17.geomatys.com/API/feature/collections/Observation/items?bbox=-120,-90,180,90

The features are returned as GeoJSON format. The Moving Features JSON encoding was not used in this testbed due to lack of time, but its use is planned for a future evolution of the storage service.

9.1.6. Features Filtering web API

The OGC API — Features — Part 3: Filtering and the Common Query Language (CQL)_ candidate standard extends the OGC API — Features — Part 1: Core standard with capabilities to support more sophisticated queries. The conceptual model is close to ISO 19143.

9.1.6.1. Usage in Testbed 17

The following example uses a simple query which is filtering by identifier. The `%3C=` text in the URL is the encoding for `#`. This syntax offers a slightly different criterion compared to the Features API mentioned above.

Example 1: https://tb17.geomatys.com/API/feature/collections/Tracklet/items?limit=20&filter=id%3C=2

Using a condition on a property other than the feature identifier (`tracklet_id`), combined with a logical operator (OR):

Example 2: https://tb17.geomatys.com/API/feature/collections/Observation/items?limit=20&filter=tracklet_id=1%20OR%20tracklet_id=7

Using a bounding box:

Example 3: https://tb17.geomatys.com/API/feature/collections/Tracklet/items?limit=20&filter=BBOX(geometry,-120,-90,180,90)

Using a geometry with more advanced operation (in this case `INTERSECTS`):

Example 4: https://tb17.geomatys.com/API/feature/collections/Tracklet/items?limit=20&filter=INTERSECTS(geometry,POLYGON(-115%2051,-115%2052,-114%2052,-114%2051,-115%2051))

9.1.7. Moving Features (ISO 19141)

The ISO 19141, *Geographic information — Schema for moving features* standard extends the ISO 19107 spatial schema for addressing features whose locations change over time. Despite the "Moving Features" name, that standard is more about "Moving geometries". The UML below shows how the `MF_Trajectory` type extends the "static" types from ISO 19107.

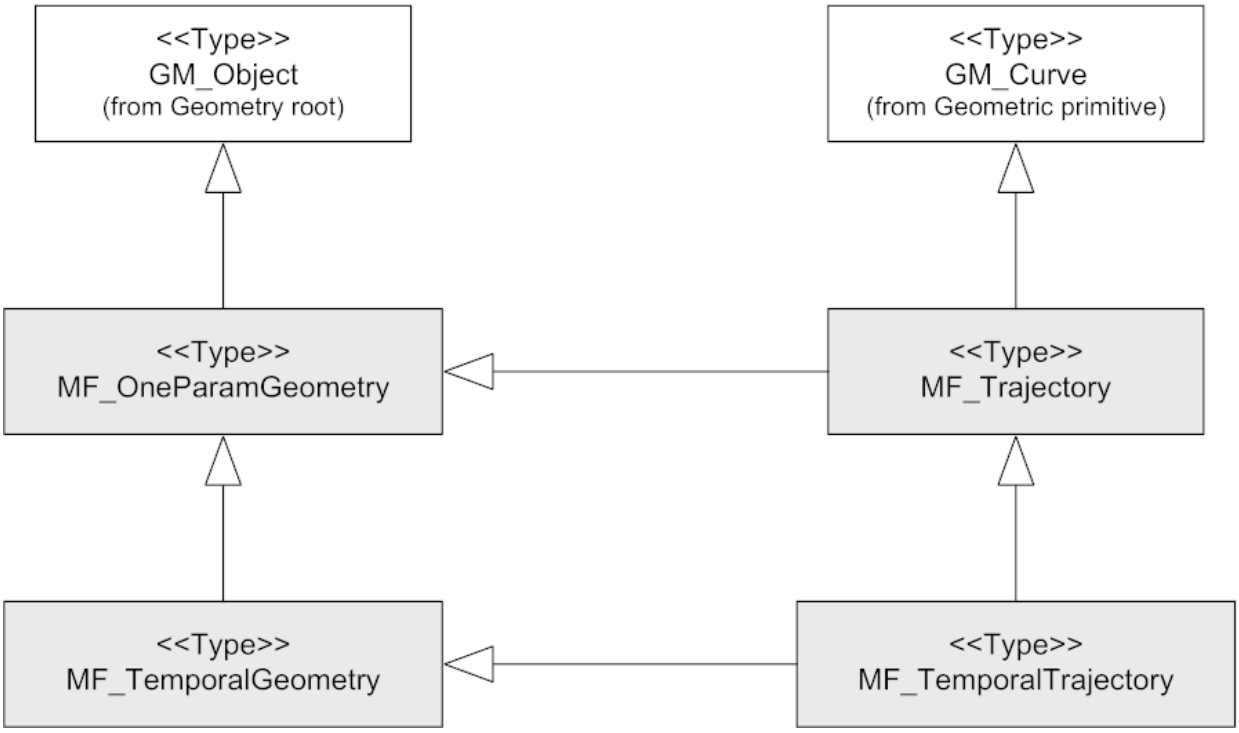

Figure 10 — Trajectory type from ISO 19141

Trajectory inherits some operations shown below. Those operations are in addition to the operations inherited from `GM_Object`. For example the `distance(…)` operation from ISO 19107 is now completed by a time-aware `nearestApproach(…)` operation.

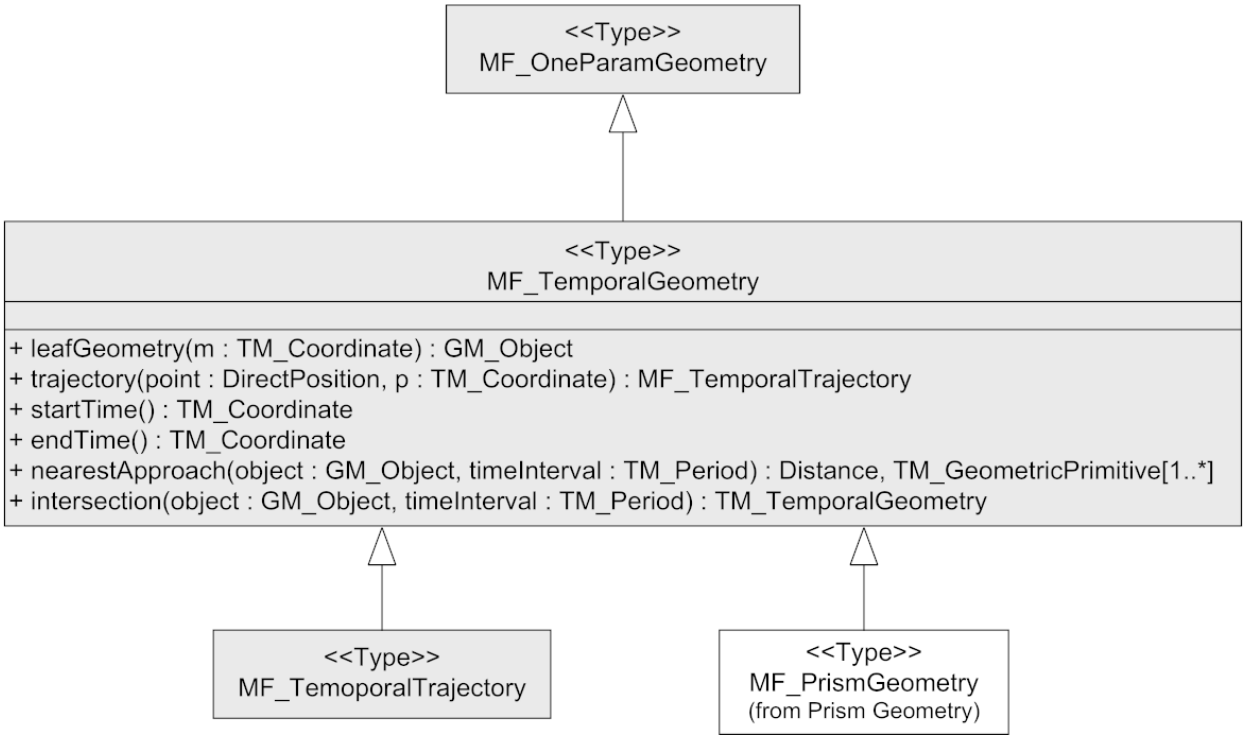

Figure 11 — Temporal geometry from ISO 19141

The ISO 19141 model was not used directly in the Testbed 17 activity. However, the operations defined by the UML class diagram in Figure 11 can be used as a source of definitions for new filter operations that complement the static operations defined by ISO 19141. For example (non-normative):

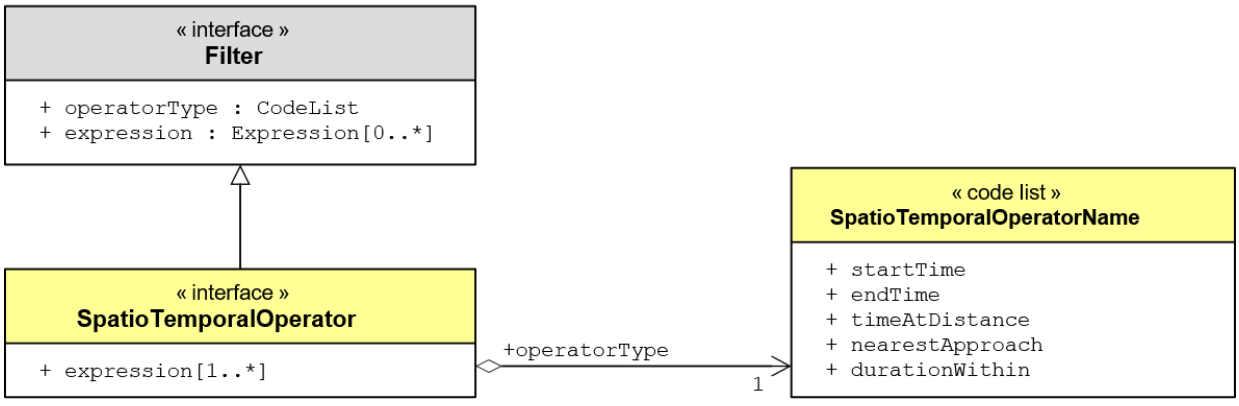

Figure 12 — Example of filter extension for moving features

Some temporal operators defined by ISO 19141 are reused in the OGC Moving Features Access standard. That standard was defined before the OGC adopted the practices seen in the above-cited OGC API — Features standard.

Consequently, the Web API part of the OGC Moving Features Access standard may need to be refactored to align with current OGC practices. That said, the operators developed in that standard may be reused.

9.1.8. Moving Features XML encoding (OGC 18-075)

The OGC 18-075 *Moving Features Encoding Part I: XML Core* standard takes a subset of the ISO 19141 specification and uses it as a basis for encoding an XML document. That standard also completes ISO 19141 by specifying attributes whose values change over time. This extension to the *General Feature Model* is shown below:

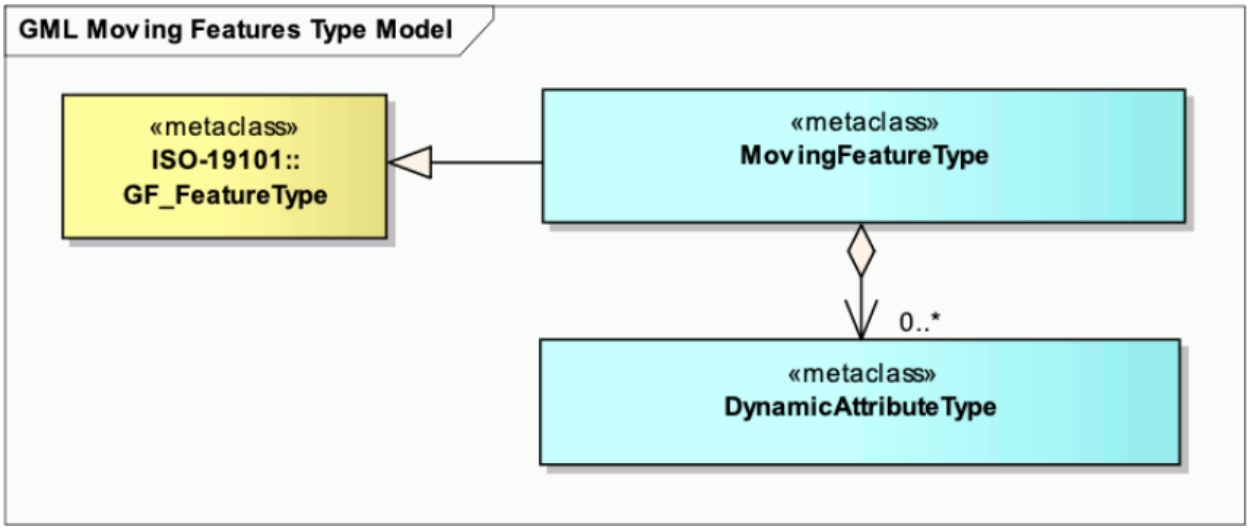

Figure 13 — Dynamic attribute from OGC 18-075

9.1.8.1. Usage in Testbed 17

The dynamic attribute illustrated in Figure 13 and defined by OGC 18-075 is used as the definition of the GeoAPI `org.opengis.feature.DynamicAttribute` interface (development branch not yet released). This is the intended internal representation of time-varying values for transferring data by storage service from (for example) database tables to JSON files. This model may be revised as a result of the Moving Features SWG work.

9.1.9. Moving Features JSON encoding (OGC 19-045)

The OGC 19-045 *Moving Features Encoding Extension — JSON* standard takes a subset of ISO 19141 standard and encodes it in JSON format. The standard provides various UML diagrams summarizing ISO 19141.

10

DRAFT SPECIFICATION OF THE MOVING FEATURES API

10 DRAFT SPECIFICATION OF THE MOVING FEATURES API

10.1. Requirements Class “Moving Features”

10.1.1. Overview

Table 1 — Testbed-17 Moving Features Requirements Class

Requirements Class	
http://www.opengis.net/spec/ogcapi-mf-1/1.0/req/core	
Target type	Web API
Dependency	OGC Moving Features
Dependency	OGC API — Features
Dependency	OGC SensorThings API

The `Moving Features` requirements class defines the requirements for a moving feature as evaluated in Testbed 17. A moving feature is a representation, using a local origin and local ordinate vectors, of a geometric object at a given reference time [ISO 19141:2008]

10.1.2. Information Resources

The two resources defined in this Testbed 17 Requirements Class are summarized in Table 2.

Table 2 — Moving Features Resources

RESOURCE	URI	HTTP METHOD
Features	`{root}/collections/{collectionId}/items`	GET

RESOURCE	URI	HTTP METHOD
MovingFeature	{root}/collections/{collectionId}/items/{mfeatureId}	GET

10.1.3. Resource Features

The MF-API Items query is an OGC API-Features endpoint that may be used to catalog pre-existing moving features. The Features operation returns data which describes the moving features available from this API. If a mFeatureID is not specified, the query will return a list of the available moving features. The list of moving features returned by the response can be limited using the bbox parameter. This behavior is specified in OGC API-Features. All parameters for use with the Items query are defined by OGC API-Features.

10.1.3.1. Operation

This operation is defined in the Features conformance class of OGC API-Features. No modifications are needed to support MovingFeature resources.

a) Issue a GET request on {root}/collections/{collectionID}/items path

Support for GET on the {root}/collections/{collectionID}/items path is required by OGC API-Features.

Table 3 — Moving Features Operation requirement

REQUIREMENT 1	/REQ/MOVINGFEATURES/FEATURES-OP
A	The API implementation SHALL comply with the OGC API — Features core requirements http://www.opengis.net/spec/ogcapi-features-1/1.0.

10.1.3.2. Response

A successful response to the Features operation is a document that contains the static data of moving features. In a typical API deployment, the Features response will list features of all offered resource types.

Table 4 — Moving Features Response requirement

REQUIREMENT 2	/REQ/MOVINGFEATURES/FEATURES-RESPONSE
A	The API implementation SHALL comply with the OGC API — Features — Part 1:Core `Features` response requirement `http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core/fc-response`.
B	The response SHALL only include moving features selected by the request with parameters.
C	Each moving feature in the response SHALL include the mandatory properties listed in Table 6.

```
type: object
required:
  - type
  - features
properties:
  type:
    type: string
    enum:
      - FeatureCollection
  features:
    type: array
    items:
      $ref: movingFeatureGeoJSON.yaml
  links:
    type: array
    items:
      $ref: link.yaml
  timeStamp:
    type: string
    format: date-time
  numberMatched:
    type: integer
    minimum: 0
  numberReturned:
    type: integer
    minimum: 0
```

Figure 14 — Features Response Schema

The following JSON payload is an example of a response to an OGC API-MovingFeatures `Features` operation.

```
{ "type": "FeatureCollection",
  "numberMatched": 2,
  "numberReturned": 2,
  "links": [
    { "href": "{root}/collections/{collectionId/items",
      "rel": "self",
      "type": "application/geo+json",
      "title": "Collection Moving Features"
    }
  ],
  "features": [
    { "type": "Feature",
      "properties": {
        "Moving_Features_Distribution": [
          { "timestamp": "2012-05-27T04:03:00Z",
```

```
            "count": 12
          },
          { "timestamp": "2012-05-27T04:03:30Z",
            "count": 12
          }
        ],
        "Average_Trajectory_Time": {
          "average_time": "00:03:11.742"
        },
        "Dwell_Time": [
          { "moving_feature_id": 158,
            "start_time": "2012-05-27T04:03:00Z",
            "end_time": "2012-05-27T04:05:27Z",
            "duration": "00:02:27",
            "dwell_time": "00:01:06"
          },
          { "moving_feature_id": null,
            "start_time": "2012-05-27T04:07:00Z",
            "end_time": "2012-05-27T04:10:48Z",
            "duration": "00:03:48",
            "dwell_time": "00:01:15."
          }
        ],
        "Moving_Features_Clusters": [
          { "timestamp": "2012-05-13T00:09:18Z",
            "clusters": [
              { "cluster_id": 0,
                "moving_features_ids": [121,155,158]
              },
              { "cluster_id": 1,
                "moving_features_ids": [123,150,151]
              }
            ]
          },
          { "timestamp": "2012-05-13T00:10:18Z",
            "clusters": [
              { "cluster_id": 0,
                "moving_features_ids": [123,155,158]
              },
              { "cluster_id": 1,
                "moving_features_ids": [121,150,151]
              }
            ]
          }
        ],
        "Trajectory_Clusters": [
          { "trajectories_ids": [161,142]
          },
          { "trajectories_ids": [163,145]
          }
        ]
      }
    }
  ]
}
```


Figure 15 — Features Example

Table 5 — Moving Features Payload Recommendation

RECOMMENDATION 1	PAYLOAD ENCODING
A	A Moving Features API implementation SHALL use a GeoJSON encoding compatible with the Features API.

10.1.3.3. Error situations

The requirements for handling unsuccessful requests are provided in Clause 10.2.1. General guidance on HTTP status codes and how they should be handled is provided in Clause 10.2.2.

10.1.4. Resource MovingFeature

10.1.4.1. Overview

A MovingFeature object consists of the set of static information that describes a single moving feature and the set of temporal object information, such as temporal geometry and temporal property. An abbreviated copy of this information is returned for each `MovingFeature` in the `{root}/collections/{collectionId}/items` response.

The schema for the moving feature object presented in this clause is an extension of the `GeoJSON Feature Object` as defined in the GeoJSON RFC. Table 6 defines the set of properties that may be used to describe a moving feature.

Table 6 — Table of the properties related to the moving feature

PROPERTY	REQ	DESCRIPTION
id	M	A unique record identifier assigned by the server.
type	M	A feature type of GeoJSON (i.e., one of 'Feature' or 'FeatureCollection').
geometry	M	A projective geometry of the moving feature.
properties	O	A set of properties for the moving feature.
bbox	O	The bounding box for the moving feature.
interval	O	The date range for the moving feature.

NOTE: The properties *id*, *type*, *geometry*, *properties*, and *bbox* were inherited from GeoJSON.

10.1.4.2. Operation

a) Issue a `GET` request on the `{root}/collections/{collectionId}/items/{mFeatureId}` path

The {mFeatureId} parameter is the unique identifier for a single moving feature offered by the API. The list of valid values for `{mFeatureId}` is provided in the `{root}/collections/{collectionId}/items` response.

10.1.4.3. Response

A successful response to the `MovingFeature` operation is a set of metadata that describes the moving feature identified by the `{mFeatureId}` parameter. This response does not include a set of temporal object information.

The temporal object information may be accessed in the future by using TemporalGeometries and TemporalProperties operations. This work is currently being defined as part of the draft OGC API- Moving Features Standard by the MF SWG.

```
type: object
required:
  - id
  - type
  - geometry
  - properties
properties:
  id:
    type: string
  type:
    type: string
    enum:
      - Feature
  geometry:
    $ref: geometryGeoJSON.yaml
  properties:
    type: object
    nullable: true
  bbox:
    type: array
    minItems: 1
    items:
      type: array
      oneOf:
        - minItems: 4
          maxItems: 4
        - minItems: 6
          maxItems: 6
      items:
        type: number
  interval:
    type: array
    minItems: 1
    items:
      type: array
```

```
      minItems: 2
      maxItems: 2
      items:
        type: string
        format: date-time
        nullable: true
  links:
    type: array
    items:
      $ref: link.yaml
```

Figure 16 — MovingFeature Response Schema

The `interval` property of the `MovingFeature` response represents a particular period of moving feature existence.

The following JSON payload is an example of a response to a `MovingFeature` operation on an implementation of OGC API-Moving Features.


```
{
  "type": "FeatureCollection",
  "numberMatched": 3,
  "numberReturned": 3,
  "links": [
    {
      "href": "http://tb17.geomatys.com/API/feature/collections/Observation/
items",
      "rel": "self",
      "type": "application/geo+json",
      "title": "this document"
    }
  ],
  "features": [
    {
      "type": "Feature",
      "id": 1,
      "geometry": {
        "type": "Point",
        "coordinates": [
          -116.1608,
          50.0851
        ]
      },
      "properties": {
        "observation_id": 1,
        "tracklet_id": 23,
        "class": "bus",
        "time": "2021-07-29T04:04:03Z",
        "color": [44,33,22],
        "bbox": [944.0,765.0,123.0,59.0]
      }
    },
    {
      "type": "Feature",
      "id": 2,
      "geometry": {
        "type": "Point",
        "coordinates": [
          -117.1608,
          51.0851
        ]
      },
```

```
      "properties": {
        "observation_id": 2,
        "tracklet_id": 23,
        "class": "bus",
        "time": "2021-07-29T04:05:03Z",
        "color": [44,33,22],
        "bbox": [944.0,765.0,123.0,59.0]
      }
    }
  ]
}
```

Figure 17 — MovingFeature Example

10.1.4.4. Error situations

The requirements for handling unsuccessful requests are provided in Clause 10.2.1. General guidance on HTTP status codes and how they should be handled is provided in Clause 10.2.2.

10.2. General Requirements

10.2.1. HTTP Response

Each HTTP request shall result in a response that meets the following requirement.

Table 7 — General HTTP-Response Requirements Class

REQUIREMENT 3	/REQ/GENERAL/HTTP-RESPONSE
A	An HTTP operation SHALL return a response which includes a `status code` and an optional `description` element.
B	If the `status code` is not equal to 200, then the `description` element SHALL be populated.

The YAML schema for these results is provided in Figure 18.

```
title: Exception Schema
description: JSON schema for exceptions based on RFC 7807
type: object
required:
  - type
properties:
  type:
    type: string
  title:
    type: string
  status:
    type: integer
  detail:
```

```
    type: string
  instance:
    type: string
```

Figure 18 — HTTP Response Schema

10.2.2. HTTP Status Codes

Table 8 lists the main HTTP status codes that clients should be prepared to receive. This includes support for specific security schemes or URI redirection. In addition, other error situations may occur in the transport layer outside of the server.

Table 8 — Typical HTTP status codes

STATUS CODE	DESCRIPTION
`200`	A successful request.
`202`	A successful request, but the response is still being generated. The response will include a `Retry-After` header field giving a recommendation in seconds for the client to retry.
`204`	A successful request, but the resource has no data resulting from the request. No additional content or message body is provided.
`304`	An entity tag was provided in the request and the resource has not been changed since the previous request.
`308`	The server cannot process the data through a synchronous request. The response includes a `Location` header field which contains the URI of the location the result will be available at once the query is complete Asynchronous queries.
`400`	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
`401`	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.
`403`	The server understood the request, but is refusing to fulfill it. While status code `401` indicates missing or bad authentication, status code `403` indicates that authentication is not the issue, but the client is not authorised to perform the requested operation on the resource.
`404`	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
`405`	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
`406`	Content negotiation failed. For example, the `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource.
`413`	Request entity too large. For example the query would involve returning more data than the server is capable of processing, the implementation should return a message explaining the query limits imposed by the server implementation.

STATUS CODE	DESCRIPTION
500	An internal error occurred in the server.

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

Table A.1 — Revision History

DATE	EDITOR	RELEASE	PRIMARY CLAUSES MODIFIED	DESCRIPTIONS
June 2, 2021	D. Younge	.1	all	initial version
November 9, 2021	D. Younge	.2	all	draft version

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] OGC: OGC Moving Features Encoding Extension — JSON 1.0, OGC 19-045r3, Open Geospatial Consortium (2020), http://docs.opengeospatial.org/is/19-045r3/19-045r3.html

[2] OGC: OGC Moving Features Access, OGC 16-120r3, Open Geospatial Consortium (2017), http://docs.opengeospatial.org/is/16-120r3/16-120r3.html

[3] OGC: OGC API — Features — Part 1: Core, OGC 17-069r3, Open Geospatial Consortium (2019), http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

[4] OGC: OGC API — Features — Part 2: Coordinate Reference Systems by Reference, OGC 18-058, Open Geospatial Consortium (2020), http://docs.opengeospatial.org/is/18-058/18-058.html

[5] OGC: OGC API — Features website, https://ogcapi.ogc.org/features/.

[6] OGC: OGC API — Common website, https://ogcapi.ogc.org/common/

[7] GitHub repository of the OGC API — Moving Features draft, https://github.com/opengeospatial/ogcapi-movingfeatures