Open Geospatial Consortium

Approval Date: 2013-01-18

Pending Date: 2012-12-21

Publication date: 2013-02-06

Reference number of this document: OGC **12-147**

External Identifier for this document: http://www.opengis.net/def/doc-type/per/OWS-9-Aviation-Architecture

Category: Engineering Report

Editor: Claude Speed

OGC® OWS-9 Aviation Architecture Engineering Report

Document type:	OGC® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

Abstract

This OGC® document describes the architecture implemented in the OWS-9 Aviation thread, including:

- A description of the architecture used for the implementation of the OWS-9 Aviation Use Cases.
- An overview of the implemented components and workflows followed by a short description of each component.
- A discussion about discovery and registry methods and practices.
- Documentation of the issues, lessons learned as well as accomplishments and scenarios that were of general interest in the Aviation thread.

More detailed information on specific aspects considered in OWS-9 Aviation may be found in the individual Aviation Engineering Reports.

Keywords

ogcdoc, ows9, ows-9, aviation, architecture

What is OGC Web Services 9 (OWS-9)?

OWS-9 builds on the outcomes of prior OGC interoperability initiatives and is organized around the following threads:

- **Aviation**: Develop and demonstrate the use of the Aeronautical Information Exchange Model (AIXM) and the Weather Exchange Model (WXXM) in an OGC Web Services environment, focusing on support for several Single European Sky ATM Research (SESAR) project requirements as well as FAA (US Federal Aviation Administration) Aeronautical Information Management (AIM) and Aircraft Access to SWIM (System Wide Information Management) (AAtS) requirements.

- **Cross-Community Interoperability (CCI)**: Build on the CCI work accomplished in OWS–8 by increasing interoperability within communities sharing geospatial data, focusing on semantic mediation, query results delivery, data provenance and quality and Single Point of Entry Global Gazetteer.

- **Security and Services Interoperability (SSI)**: Investigate 5 main activities: Security Management, OGC Geography Markup Language (GML) Encoding Standard Application Schema UGAS (UML to GML Application Schema) Updates, Web Services Façade, Reference Architecture Profiling, and Bulk Data Transfer.

- **OWS Innovations**: Explore topics that represent either new areas of work for the Consortium (such as GPS and Mobile Applications), a desire for new approaches to existing technologies to solve new challenges (such as the OGC Web Coverage Service (WCS) work), or some combination of the two.

- **Compliance & Interoperability Testing & Evaluation (CITE)**: Develop a suite of compliance test scripts for testing and validation of products with interfaces implementing the following OGC standards: Web Map Service (WMS) 1.3 Interface Standard, Web Feature Service (WFS) 2.0 Interface Standard, Geography Markup Language (GML) 3.2.1 Encoding Standard, OWS Context 1.0 (candidate encoding standard), Sensor Web Enablement (SWE) standards, Web Coverage Service for Earth Observation (WCS-EO) 1.0 Interface Standard, and TEAM (Test, Evaluation, And Measurement) Engine Capabilities.

The OWS-9 sponsors are: AGC (Army Geospatial Center, US Army Corps of Engineers), CREAF-GeoViQua-EC, EUROCONTROL, FAA (US Federal Aviation Administration), GeoConnections - Natural Resources Canada, Lockheed Martin Corporation, NASA (US National Aeronautics and Space Administration), NGA (US National Geospatial-Intelligence Agency), USGS (US Geological Survey), UK DSTL (UK MoD Defence Science and Technology Laboratory).

License Agreement

Contents

Page

Executive Summary 1

1 Introduction 1

2 Summary of Accomplishments 1
- 2.1 Web Feature Service 2
- 2.2 Registry Service 2
- 2.3 Event Service 2
- 2.4 Web Processing Service 2
- 2.5 Data Management Service 2
- 2.6 Aviation Client 3

3 Observations 3

4 Lessons learned 4

5 Recommendations 5

1 Introduction 7
- 1.1 Scope 7
- 1.2 Document contributor contact points 7
- 1.3 Revision history 8
- 1.4 Future work 8
  - 1.4.1 Spatial Filtering of Non-Spatial Features 8
  - 1.4.2 Geometry Processing via Web Processing Service 9
- 1.5 Forward 9

2 References 10

3 Terms and definitions 12

4 Abbreviated terms 12

5 OWS-9 Aviation Architecture - Overview 14

6 Web Services Architecture 15
- 6.1 Workflows 15
  - 6.1.1 Data Provision 15
  - 6.1.2 Portrayal (ePIB) 17
  - 6.1.3 Metadata / Provenance Retrieval 19
  - 6.1.4 Data Management Service 21
- 6.2 Component Descriptions 24
  - 6.2.1 COMSOFT Web Feature Service 24
  - 6.2.2 Snowflake Web Feature Service 26
  - 6.2.3 Registry Service 29
  - 6.2.4 IDS Event Service 30

6.2.5 IfGI Event Service 33
6.2.6 Luciad Web Processing Service 35
6.2.7 52°North Web Processing Service 38
6.2.8 ePIB Client 40
6.2.9 ATM-TGS Data Management Service 43
6.2.10 Harris Data Management Service 46
6.2.11 ATM-TGS Aviation Client 50
6.2.12 ATM-TGS Dispatch client 51
6.2.13 Luciad Aviation Client 52
6.2.14 Deployment characteristics 54

7 Data Provision via Web Feature Service and Event Service 55
7.1 Overview 55
7.2 General Service Architecture 55
7.2.1 Web Feature Service 56
7.2.2 Event Service 56
7.3 Communication Patterns 56
7.3.1 Data Model 56
7.3.2 Service Chaining 57
7.4 Advanced Filtering Functionality 58
7.4.1 Stored Queries / Filters 59
7.4.2 Simple Altitude Queries 61
7.4.3 Spatial Filtering of Non-spatial Features 63
7.4.4 AIXM Features as Geometry Operands 64
7.4.5 Event Service Update Intervals 66
7.4.6 Selective Metadata Retrieval 69
7.5 Service Discovery Based on Spatial Extent 71
7.6 Accomplishments 71
7.7 Lessons Learned 71
7.8 Future Work 72

8 Geometry Processing via Web Processing Service 73
8.1 Overview 73
8.2 Applications Profiles 73
8.2.1 Intersection Calculation 74
8.2.2 Spatial Relation 76
8.2.3 Geometry Retrieval 77
8.2.4 Limitations 81
8.3 Accomplishments 81
8.4 Lessons Learned 82
8.5 Future Work 82

9 FAA SWIM Compliance Analysis 83
9.1 Introduction 83
9.2 Analysis Process 83
9.3 Assessment Results 85

9.4 Findings and Observations 93
9.4.1 SWIM Requirements 94
9.4.2 Observations 94
9.5 Summary and Recommendations 95

10 Scenario 95
10.1 Introduction 95
10.2 Component assignments 96
10.3 Detailed Scenario 97
10.4 Clarifications 105
10.5 Aspects Not Covered by Main Scenario 106

11 Lessons Learned 106
11.1 Inconsistent Data Publication Models 106
11.1.1 Problem Statement and Description 106
11.1.2 Recommendations 107

12 Technical Challenges / Future Work 107

13 Accomplishments 109
13.1 Web Feature Service 109
13.2 Registry Service 110
13.3 Event Service 110
13.4 Web Processing Service 110
13.5 Data Management Service 110
13.6 Aviation Client 111

Annex A - SWIM Compliance Assessment 112

Annex B - Extended Event Service XML Schema 123

Annex C - Airspace Geometry Computation 128

Annex D – Spatial Relation Process Defintions 131

Figures

Figure 1 Common Component Interactions to Retrieve and Disseminate Data 16

Figure 2 ePIB Generation 17

Figure 3 Metadata / Provenance Workflow 21

Figure 4 Data Management Service Workflow – Session Establishment 22

Figure 5 Data Management Service Workflow – Request Response 23

Figure 6 Data Management Service Workflow – Subscription Request / Receive Notifications 24

Figure 7 Overview of the Snowflake Aviation Component Architecture 27

Figure 8 - Update Interval Framework. 33

Figure 9 Update Interval Internals 35
Figure 10 Example of an Intersection Between an AIXM Route and Airspace 36
Figure 11 52°North WPS Three-tier Architecture 39
Figure 12 52°North WPS with ArcGIS Server Backend 40
Figure 13 Example of an ePIB Bitmap Image Generated by the FPS 41
Figure 14 DMS Context Description 44
Figure 15 - Data Provision Service Architecture. 55
Figure 16 - DNOTAM Workflow. 58
Figure 17 – Update Interval Subscription Illustration. 68
Figure 18 – Intersection of Airspace and RouteSegment. 75
Figure 19 – Spatial Relation "Intersects" for Airspace and RouteSegment. 77
Figure 20 SWIM Compliance Assessment Process 84
Figure 21 OWS-9 Operational Scenario 96

Listings

Listing 1 – Taxiway Closure DNOTAM Event. 56
Listing 2 – Stored Query Execution. 59
Listing 3 – Stored Filter Subscription. 60
Listing 4 - Exemplary Altitude Query. 62
Listing 5 – Custom XPath Function "extentOf". 64
Listing 6 - Feature by Identifier Reference. 65
Listing 7 – Exemplary Geometry Retrieval WPS Request. 65
Listing 8 – Update Intervals Capabilities Portion. 69
Listing 9 – Example Update Interval Subscription. 69
Listing 10 – Example Metadata Exclusion Subscription. 70
Listing 11 - Exemplary Event Service Reference for the IfGI ES. 71

OGC® OWS-9 Aviation Architecture Engineering Report

Executive Summary

1 Introduction

The Open Geospatial Consortium (OGC) is engaged in an Interoperability Program which is a global, hands-on and collaborative prototyping program designed for rapid development and delivery of proven candidate specifications into OGC's Specification Program, which can then be formalized for public release. In OGC's Interoperability Initiatives, international technology developers and providers team together to solve specific geo-processing interoperability problems posed by the initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments, and interoperability support services – all designed to encourage rapid development, testing, validation and adoption of open, consensus-based standards specifications.

This OGC® document describes the architecture implemented in the OGC Web Services Test Bed 9 (OWS-9) Aviation thread, including:

- A description of the architecture used for the implementation of the OWS-9 Aviation Use Cases.
- An overview of the implemented components and workflows followed by a short description of each component.
- A discussion about discovery and registry methods and practices.
- Documentation of the issues and lessons learned as well as accomplishments and scenarios that were of general interest in the Aviation thread.

2 Summary of Accomplishments

The OWS-9 Aviation thread has a number of notable accomplishments:

2.1 Web Feature Service

- Data was loaded, validated, transformed and corrected from multiple sources This effort included splitting the data into two Web Feature Service (WFS) instances, one for North American data and one for Middle-Northern Europe data

- Calculation of airspace extents in full 2.5D, including composite airspaces (unions, intersections and subtractions), support for altitude queries including calculation of extents of non-spatial feature types (runways, taxiways and aprons), and support for spatial filtering of non-spatial feature types.

2.2 Registry Service

The OWS-9 Aviation Registry was used to host resources shared between multiple OGC services (e.g. ISO metadata pertaining to datasets common to multiple Web Feature Service implementations, styling information shared between multiple Feature Portrayal Service implementations, and ISO Service metadata describing the Services deployed in OWS-9. The efficient retrieval of metadata task was also implemented using the Aviation Registry by designing queries to retrieve only the relevant excerpts of the metadata desired by clients.

2.3 Event Service

Compared to the Event Services (ES) used in the previous OGC test beds, major improvements were been implemented during OWS-9, particularly features summarized under the term "Advanced filtering functionality." These included "Event Service Update Intervals", "Stored Filters", "AIXM Features as Geometry Operands", "Spatial Filtering of Non-spatial Features", "Simple Altitude Queries" as well as "Selective Metadata Retrieval".

2.4 Web Processing Service

Web Processing Service profiles were implemented to support Electronic Pre-Flight Information Briefing (ePIB) generation and geometry processing, including calculation of topological relations between two AIXM 5.1 features in ellipsoidal space. Supported topological relations included INTERSECTS, DISJOINT, EQUALS, TOUCHES, CROSSES, OVERLAPS, CONTAINS, WITHIN, COVERS, COVERED_BY.

2.5 Data Management Service

OWS-9 introduced a Data Management Service (DMS) which included a set of functionalities to provide reliable and efficient management of communications between aircraft and services located on the ground. The DMS provided:

- DMS service discovery which included the process of finding and setting the processing options used by the DMS to manage communications between the aircraft and dispatch client.

- DMS basic pass-through which handled the forwarding of request/response and notifications between the aircraft client and OGC web services.

- Reliable messaging.

- Data compression and expansion.

- Data filtering.

- Dispatch synchronization.

2.6 Aviation Client

Aviation clients were implemented and tested that provided map-centric displays with intuitive user interface giving access to data from entities such as WFS, ES, and the DMS. The continuing evolution of these clients provided a rich set of capabilities and features that helped to demonstrate the OWS-9 Aviation scenario and to perform testing and integration with a wide variety of service components.

3 Observations

This test bed presented a number of challenges ranging from inconsistencies among data publishers to calculation of intersecting geometries that provided opportunities for advancing the state of interoperability for the international community.

- Data Management Service - This test bed interjected a new concept into the OGC environment – a value-added proxy between a client and the web services that serve it. Inspired by the Federal Aviation Administration Aircraft Access to System-Wide Information Management (SWIM) project, this initiative explored the efficacy of using proxies to manage many of the tasks required to connect an airborne client to multiple Air Navigation Service Provider (ANSP) data publishers.

- Inconsistent Data Publication Models - As OGC standards, such as AIXM 5.1, are being increasing deployed in operational production environments by data publishers, differing interpretations of the standards are becoming an obstacle to interoperability. It appears that the data publishers are interpreting the standards to create their own publication mechanisms and the complexity of the standards and the nuances in their interpretation lead to incompatibility among the data publishers.

- Calculation of the airspace extent in 2.5D - The correct calculation of composite airspaces, which in turn may be composed of other airspaces and which may in turn include references to geographical borders, in conjunction with taking the vertical dimension into account turned out to be a challenging task that will require continuing work to resolve.

- Feature Portrayal Service Performance - One important and reoccurring challenge when working with a Feature Portrayal Service (FPS) in general is to get optimal performance. The high flexibility of being able to submit any kind of feature data and styling information comes with difficulties to achieve this. The AIXM format is verbose, so large amounts of bandwidth can be needed to provide the FPS with the necessary feature data. While not a particular problem in a test bed environment with limited data throughput requirements, as international production implementations grow it will increasingly become an issue that needs to be resolved.

4 Lessons learned

- Common Definition of WPS Process Definitions - When designing a WPS ProcessDefinition, the format for inputs and outputs can be defined by pointing to a commonly available XML Schema. By implication, a WPS implementing such ProcessDefinition must support all available root elements as inputs/outputs. To enable true interoperability among different WPS instances, one needs to be as exact as possible defining the inputs/outputs.

- Interoperability among ANSPs – One of the important lessons learned in the development of the DMS was creating a solution that could fully communicate with both the North American and European SWIM environments. Although the overarching system concepts for SWIM are the same in both North American and Europe, the actual SWIM implementations currently employ different technical standards for their communication interfaces. Resolution or harmonization of these standards will be required for truly interoperable technical solutions for communications with both SWIMs.

Conclusions

Along with the usual technical challenges associated with implementing interoperable solutions distributed among numerous international data publishers and users, a common theme was apparent throughout most of the implemented demonstration solutions – common interpretations of existing standards and interoperable solutions based on those interpretations. In the interests of achieving long-term interoperability, discipline in standardizing interpretations will be required for all uses of the standards.

1. One approach would be to establish governance processes for identifying data that is common and critical to proper functioning of the services and then managing its application. This governance could consist of:
   - Developing a set of rules for defining who was the definitive authority for assigning a feature its UUID. Ideally this should be the entity that is responsible for creating and maintaining the data as the most knowledgeable about the data capture and lifecycle maintenance rules). Once a UUID is assigned, it could be persisted throughout the data exchange system and no third party should change it once supplied.
   - Developing a process for determining when a service was not following the rules.
2. Conformance testing – OWS-9 participants are building and/or testing Compliance and Interoperability Test (CITE) reference implementations for the following services:
   - SPS 2.0.
   - WMS 1.3 client and server.
   - WFS 2.0.
   - WFS EO 1.0.

   As these components are tested, validated, and incorporated into implementation integration testing, it will be easier to check that a service really supports a certain set of functionality. Then the required set of functionality can be stated more precisely in procurement requirements and service providers will be better able to prove up-front which functionality they support. In conjunction with an active governance and certification process, the variations in standards interpretations can be minimized with improved interoperability as the result.

5 Recommendations

The evolution of the business needs that drive technical standards continually requires the technical community to make choices about customization and tailored applications that manipulate data flowing from a standards-based web service. While these efforts satisfy customer requirements, they tend to create intrinsic incompatibilities when the data is stored in a standards-based data model and then redistributed. The challenge for standards bodies such as the OGC is to guide the standards working groups into creatively resolving incompatibilities among implementations without making the standard more complex than absolutely necessary. Following this theme, it is recommended that OGC expand and formalize its initiatives designed to reduce the number of standards interpretation incompatibility issues, such as:

3. Conformance testing - OWS-9 participants are building and/or testing Compliance and Interoperability Test (CITE) reference implementations for the following services:
   - SPS 2.0
   - WMS 1.3 client and server
   - WFS 2.0

- WFS EO 1.0.

  As these components are tested, validated, and incorporated into implementation integration testing, it will be easier to determine whether a service really supports a certain set of functionality. Then the required set of functionality can be stated more precisely in procurement requirements and service providers will be better able to prove up-front which functionality they support.

- Governance and certification – OGC continues to maintain significant standards harmonization initiatives to help ensure the cross-pollination of good ideas across its many standards working groups. Expansion and formalization of these initiatives in conjunction with an active governance and certification process would help reduce the variations in standards interpretations with the result of improved interoperability.

OGC® OWS-9 Aviation Architecture Engineering Report

1 Introduction

1.1 Scope

This OGC® document describes the architecture implemented in the OWS-9 Aviation thread, including:

- A description of the architecture used for the implementation of the OWS-9 Aviation Use Cases.
- An overview of the implemented components and workflows followed by a short description of each component.
- A discussion about discovery and registry methods and practices.
- Documentation of the issues, lessons learned as well as accomplishments and scenarios that were of general interest in the Aviation thread.

More detailed information on specific aspects considered in OWS-9 Aviation may be found in the individual Aviation Engineering Reports.

1.2 Document contributor contact points

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

Name	Organization
Daniel Balog	Luciad
Roger Brackin	Envitia
David Burggraf	Galdos
Thibault Dacla	ATM-TGS
Hoang Dam	IDS
Johannes Echterhoff	IGSI
Daniel Hardwick	Snowflake Software

Robin Houtmeyers	Luciad
Matthes Reike	52^0North / IfGI
Claude Speed	SpeedSquared
Timo Thomas	COMSOFT
Debbie Wilson	Snowflake Software
Stuart Wilson	Harris

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
12/2/2012	0.1	Speed	All	Incorporation of component descriptions and incorporated ERs
12/4/2012	0.2	Speed	Executive Summary	Incorporation of an Executive Summary and minor edits
12/21/2012	1.0	Speed	All	Final edit

1.4 Future work

Future work to advance the state of OGC standards, their implementation, and interoperability is addressed in this section.

1.4.1 Spatial Filtering of Non-Spatial Features

To enable spatial filtering of AIXM features that do not contain a spatial extent, rules for computing the geometry of such features based upon their constituent parts or that of other AIXM types that have a relationship to this feature need to be developed. Within former test beds the so-called “Reverse Associations” had been introduced which provide reverse links to associated features (e.g. reverse associations on a Runway referring to its RunwayElements). But evaluation of that approach revealed that the management of such reverse associations in the AIXM Temporality model is very complex, and the increase in complexity outweighs the benefits. An alternative approach needs to be developed. Since most AIXM features define multiple representations of a geometry (e.g. gml:boundedBy and aixm:ElevatedPoint), future work is proposed to integrate an additional parameter into the XPath function to define the use case / interpretation. This will enable a client to retrieve different representations for specific tasks.

When subscribing for an update interval, client software should be able to define if it is interested in receiving all messages or only the latest event for the specified time window. The capability of a service to detect different features (e.g. two different Airports) within a time window should be specified in the service capabilities and is proposed for future work.

1.4.2 Geometry Processing via Web Processing Service

A WPS geometry retrieval profile can be used to resolve the geometry from an AIXM feature. One specialty of AIXM is that airspaces contain surfaces (AirspaceVolumes) that can be formed using geometric operations (difference, union, intersection). The WPS server is able to return a set of disjoint GML surfaces that represent the original airspace. Additionally, airspaces contain altitude data. This is currently not modeled by GML. Proposed future work includes:

- Altitude modeling – Proposed future work is to create a custom GML schema that explicitly models the altitude to enable WPS profiles to integrate altitude calculations into geometry intersection calculations.

- 2.5D and 3D calculations – Current implementations of the WPS only support the horizontal extent of AIXM features. One future work item should focus on the integration of the vertical extent to determine whether a general approach for transforming AIXM 2.5D geometries into GML 3D geometries should be developed.

- Additional WPS Profiles – Proposed future work includes development of additional profiles calculate actual altitudes (e.g. based on barometric pressure, DEMs), calculate the containing circle for an Airspace border (used for a Digital NOTAM's Q line), and metadata/provenance resolution.

1.5 Forward

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.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OWS-9 Engineering Reports:

- [OGC 12-094] OWS-9 AIRM to WXXM Derivation Engineering Report.
- [OGC 12-144] OWS-9 Aviation Registry Engineering Report.
- [OGC 12-145] OWS-9 Aviation Metadata and Provenance Engineering Report.
- [OGC 12-146] OWS-9 Web Feature Service Temporality Extension Engineering Report.
- [OGC 12-151] OWS-9 Aviation Portrayal Engineering Report.
- [OGC 12-158] OWS-9 Aviation Performance Study Engineering Report.
- [OGC 12-163] OWS-9 Data Transmission (to Aircraft) Management Engineering Report.

Other OGC Documents:

- OGC 05-007r7, OpenGIS Web Processing Service 1.0.0.
- OGC 06-121r3, OGC® Web Services Common Standard.
- OGC 07-110r3, OGC® CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW.
- OGC 07-144r3 OGC® CSW-ebRIM Registry Service - Part 2: Basic extension package.
- OGC 08-133, OpenGIS Sensor Event Service Interface Specification Discussion Paper.
- OGC 09-025r1, OpenGIS Web Feature Service 2.0 Interface Standard.
- OGC 09-026r1 , OpenGIS Filter Encoding 2.0 Encoding Standard.

Aviation Documents:

- Aeronautical Information Exchange Model (AIXM) - AIXM Application Schema

Generation, online at http://www.aixm.aero/public/standard_page/download.html.

- AIXM - AIXM Application Schema Generation, online at http://www.aixm.aero/public/standard_page/download.html .
- AIXM - Temporality Model v1.0, online at http://www.aixm.aero/public/standard_page/download.html .
- AIXM - Temporality Model v1.0, online at http://www.aixm.aero/public/standard_page/download.html .
- AIXM - UML to XML Schema Mapping v1.1, online at http://www.aixm .
- AIXM - UML to XML Schema Mapping v1.1, online at http://www.aixm.aero/public/standard_page/download.html.
- Digital NOTAM Event Specification, ed. 1.0 (Proposed Release), online at http://www.aixm.aero/public/standard_page/digital_notam_specifications.html .

Other Documents:

- AIRM-ISRM Requirements (08.03.02.D04).
- OASIS Web Services Reliable Messaging (WS---Reliable Messaging) Version1.2, http://docs.oasis---open.org/ws---rx/wsrm/200702.
- OASIS WS-BaseNotification Standard , Web Services Base Notification 1.3.
- Operational Service and Environment Definition (OSED Step 1) (13.02.02.D01).
- SESAR Demonstrator Report (08.03.02.D08).
- SWIM Registry Concept of Operations V1 (08.03.02.D03).
- SWIM Service Compliance Requirements - http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/atc_comms_services/swim/documentation/media/compliancy/SWIM%20Service%20Compliance%20Requirements.pdf.
- Systems Rules Model (SRM) – see [OGC 12-163] OWS-9 Data Transmission (to Aircraft) Management Engineering Report.
- The Unified Code for Units of Measure (UCUM), online at http://aurora.regenstrief.org/~ucum/ucum.html.
- X.891: Information technology - Generic applications of ASN.1: Fast infoset – 2007-01-30.

- W3C XML Information Set (second edition) http://www.w3.org/TR/xml---infoset/.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r3] apply.

4 Abbreviated terms

2D	Two dimensional
2.5D	Two and a half dimensional (3/4 perspective)
3D	Three dimensional
AIP	Aeronautical Information Publication
AIM	Aeronautical Information Manual
AIXM	Aeronautical Information Exchange Model
AMDB	Airport Mapping Database
AOC	Airline Operational Communication
ATM	Air Traffic Management
CCI	Cross Community Interoperability
CDMS	Client DMS module
COTS	Commercial Off The Shelf
CSW	OGC Catalogue Service for the Web
DEM	Digital Elevation Model
DMS	Data Management Service
DNOTAM	Digital NOTAM
DS-client	Dispatch synchronization client
E-DMS	Europe Data Management Service
E-ES	Europe Event Service
E-WFS	Europe WFS
EAD	European Organisation for the Safety of Air Navigation
EANS	Estonian Air Navigation Services
ebRIM	OASIS ebXML Registry Information Model
EFB	Electronic Flight Bag

ePIB	Electronic Pre-flight Information Briefing
ES	Event Service
ESA	European Space Agency
FAA	Federal Aviation Administration
FES	Filter Encoding Specification
FI/FIS	FastInfoSet
FPS	Feature Portrayal Service
GML	Geography Markup Language
GP-WFS	Geometry Processing WFS
GZIP	GNU ZIP
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISO	International Standards Organization
METAR	A format for reporting weather information
N-client	NOTAM generation client
NA-DMS	North American Data Management Service
NA-ES	North American Event Service
NA-WFS	North American WFS
NASR	National Airspace System Resources
NOTAM	Notice To Airmen
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OWS	OGC Web Service
OWS-9	OGC Web Services (OWS), phase 9
PDP	Policy Decision Point (typically in a XACML framework)
PEP	Policy Enforcement Point (typically in a XACML framework)
QoS	Quality of Service
QName	Qualified Name
SAA	Special Activity Airspace
SE	Symbology Encoding
SESAR	Single European Sky ATM Research Programme
SLD	Styled Layer Descriptor
SOAP	Simple Object Access Protocol
SPS	SWIM Product Standardization

SWIM	System-Wide Information Management
TAF	Terminal Area Forecast
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
URN	Uniform Resource Name
WCS	Web Coverage Service
WFS	Web Feature Service
WFS-T	WFS Transactional
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Web Services Description Language
WS-N	OASIS Web Services Notification
WXXM	Weather Information Exchange Model
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

5 OWS-9 Aviation Architecture - Overview

The OWS-9 Aviation thread architecture can be separated into three tiers (see Figure 1):

- The *Client Tier* contains the client applications.
- The *Business Process Tier* contains components that offer services on top of the Access Tier: Web Processing Service, Registry, Feature Processing Service, Data Management Service, and Event Service.
- The *Access Tier* contains Web Feature Services serving AIXM 5.1 data as well as Weather Information Exchange Model (WXXM) 1.1.3 and other data.
- *Additional or Orthogonal Components* contain a performance assessment tool and AIRM to WXXM Mapping / Encoding tool.

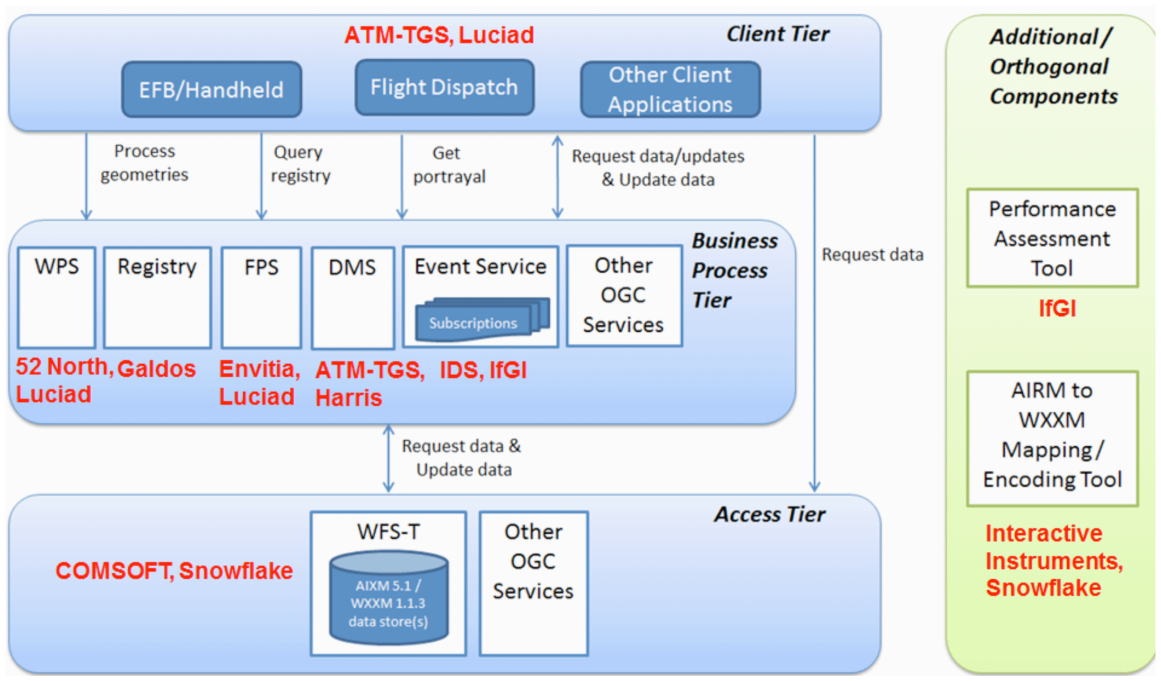


Figure 1 — OWS-9 Aviation Architecture – High-Level Overview

Figure 1 shows the links between the tiers and the general functionality that is invoked. It also shows which participants provided which components. A summary description of the components is provided in chapter 7.

6 Web Services Architecture

6.1 Workflows

This section provides a high-level overview of the common interactions between client and service components in the OWS-9 Aviation service infrastructure.

6.1.1 Data Provision

The following sequence diagram shows common interactions for retrieving and disseminating data that is of interest to a client.

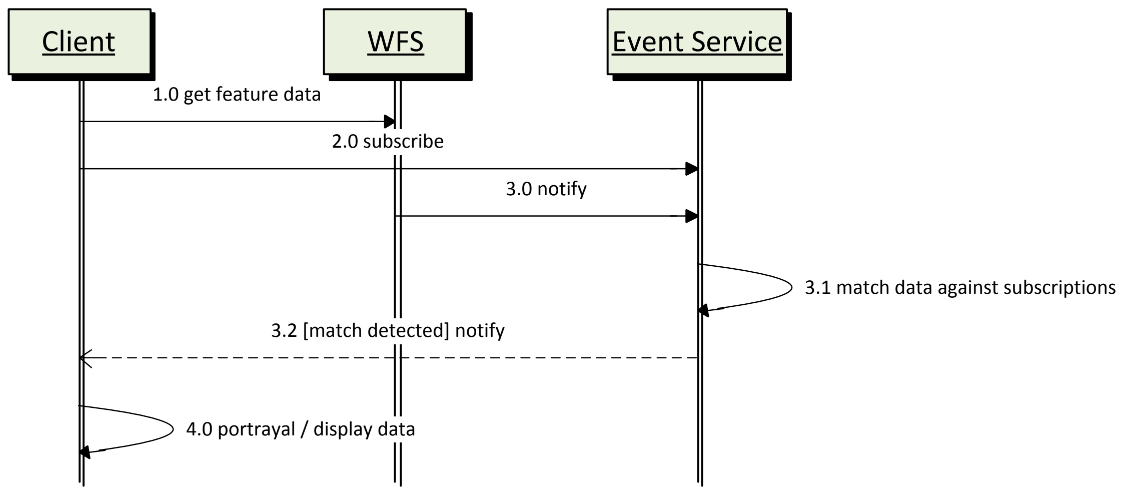

Client
WFS
Event Service
1.0 get feature data
2.0 subscribe
3.0 notify
3.1 match data against subscriptions
3.2 [match detected] notify
4.0 portrayal / display data

Figure 1 Common Component Interactions to Retrieve and Disseminate Data

The following paragraphs describe the interactions shown Figure 1.

1.0: The client retrieves feature data from the Web Feature Service (WFS) (e.g. via the GetFeature operation). The client can request general information, for example, on airspaces and airports. Using specific filter criteria, it can also query the WFS to identify suitable alternate/diversion airports (e.g. by searching for airports that have a passenger terminal, re-fueling facilities, a hard-surface runway of certain required minimum length, etc.). Weather data formatted as METARs and Terminal Area Forecasts (TAFs) can also be served by and requested from a WFS.

2.0: The client creates a subscription at an Event Service (ES) to automatically be notified whenever the Event Service receives new data that matches the subscriptions filter criteria. For example, the subscription can be used to receive Digital Notices to Airmen (DNOTAMs) for airports (e.g. the destination and alternates, airspaces, and airspace activations). In this step, only the subscription is created and the interaction ends. The actual publishing of new data is described in Step 3.0.

3.0: Whenever a WFS detects a relevant change in its feature data (for example that an airspace got a new activation or that a runway was closed), it generates a notification that represents the change (for example encoded as a DNOTAM) and sends it to the Event Service.

3.1: The Event Service processes the content of the notification and matches it against all subscriptions to detect matches.

3.2: If the data matches a given subscription, then the Event Service notifies the recipient defined for the subscription (here it is the client).

4.0: The client processes the data that it either retrieved from a WFS or received from an Event Service and usually portrays it according to some style.

6.1.2 Portrayal (ePIB)

The following sequence diagram shows common interactions for retrieving data and merging it with airport data to enable generation of the Electronic Pilot Information Briefing (ePIB) document.

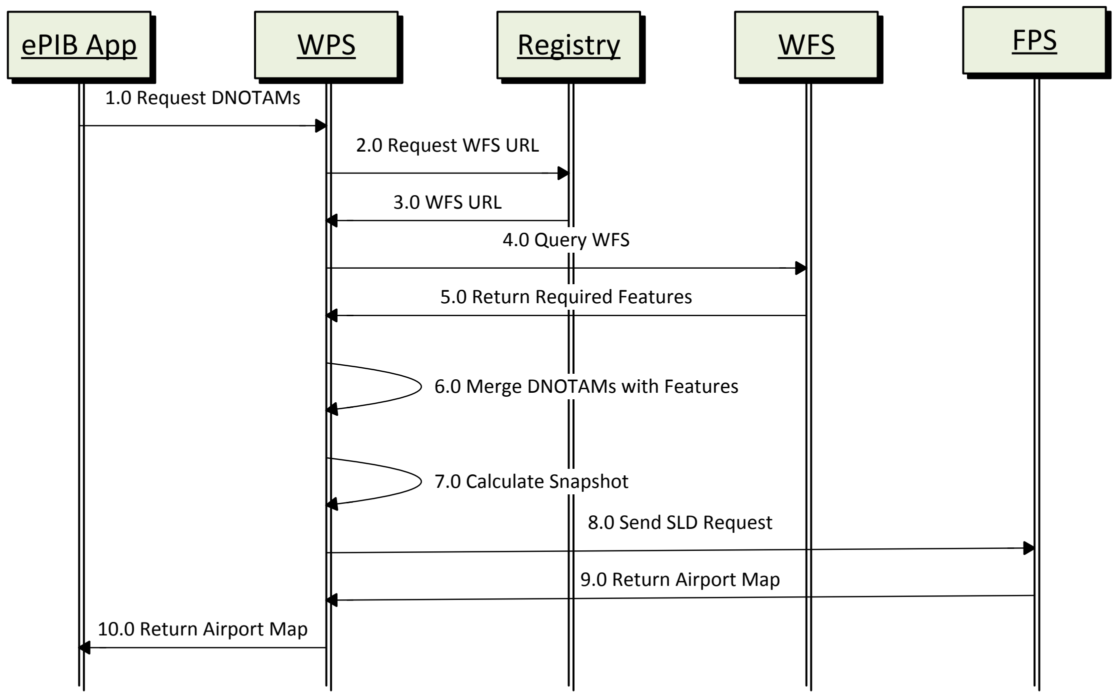


Figure 2 ePIB Generation

The following paragraphs describe the interactions shown in Figure 2.

1. The ePIB component sends the airport map request to the Web Processing Service (WPS), consisting of an airport (GML) identifier and a list of Digital NOTAMs. The Digital NOTAMs are encoded according to the Digital NOTAM Event Specification.

2. The WPS needs features to enrich the Digital NOTAM dataset, and relies on the Registry Service to discover a WFS (or multiple ones) that can deliver the necessary features. Discovery can be mainly based on the GML identifier of a feature and

feature types. For instance, retrieve the WFS that can deliver the Navaid feature with this Geography Markup Language (GML) identifier, or retrieve the WFS that can deliver Runway features for the airport with a given GML identifier. The following features are considered necessary:

- Features relevant for the airport layout. All features related to the airport layout (i.e., runways, taxiways, aprons ...) are needed for the resulting airport map. For this, a WFS request is sent to query airport layout features (e.g. a hardcoded set of feature types) for the specified airport.

- Features impacted by the Digital NOTAMs. Each Digital NOTAM is analysed: if it concerns an airport layout feature (e.g. a TaxiwayElement, RunwayElement, VerticalStructure) that was already retrieved in 2.1, the corresponding feature representation on the map is updated / overlaid with the information coming from the Digital NOTAM.; if it concerns a feature that is not yet part of the airport layout (e.g. the Digital NOTAM is about the unavailability of an ILS Navaid) which is not part of the standard airport layout, then this feature needs to be retrieved from a WFS and its graphical "temporary status" situation is overlaid on the map.

3. The Registry Service returns the URL(s) of the WFS components that can deliver the required features.

4. The WFS is contacted to retrieve the requested features.

5. The WFS returns the required features.

6. The WPS now has a set of AIXM 5.1 features retrieved from the WFS (BASELINE timeslices but potentially also TEMPDELTAs/PERMDELTAs) and a set of Digital NOTAM features (essentially TEMPDELTA timeslices). This data is merged to obtain one coherent AIXM 5.1 feature collection that serves as a basis for the airport map.

7. The AIXM 5.1 feature collection still contains multiple timeslices for each feature, valid for various time intervals. To avoid timeslice calculations in the FPS, the WPS calculates a snapshot for the AIXM 5.1 feature collection. The snapshot date could be determined by inspecting the Digital NOTAMs.

8. The WPS constructs a Styled Layer Descriptor (SLD) request with a user layer embedded with the snapshot features and sends it to the FPS.

9. The FPS renders the snapshot features and returns the airport map as a bitmap.

10. The airport map (bitmap) is sent to the ePIB component.

In this design, the WPS is the central component to manage the airport map generation and contains all the business logic. An advantage is that the WFS and FPS components remain general and do not require any modification that is the specific for the ePIB setup.

6.1.3 Metadata / Provenance Retrieval

Dataset Metadata can be defined as data about a dataset. One class of metadata of particular interest in OWS-9 Aviation is data provenance (or lineage), which records the salient characteristics of the evolution of shared datasets, typically due to quality additions or enhancements by different users. The creation of such metadata is meant to tell a potential data user about its quality, whether it is fit for purpose, and provide guidance on its use and re-use. The value of metadata increases exponentially with the number of different responsible parties involved in sharing and updating information resources in distributed data environments. In the metadata creation, discovery, access, and management sub-tasks of the OWS-9 Aviation thread, we implemented an aviation/weather metadata profile of ISO 19139 and tested its handling and processing by a community of users of OWS-9 software services. The common metadata (e.g. license terms) about aviation datasets (AIXM/WXXM) and services published by authorized distributors was used to discover and access shared resources (e.g. data, services, styling information) and answer the following questions:

- Who was the creator/publisher?
- Who has permission to use the resources and what are the terms?
- When was the data last modified? How was the data transformed?
- Where can it be accessed? How can it be processed and by which services?
- Which applications, software configuration or tool’s settings were used to process the data?

Six distinguished categories of metadata are summarized as follows:

1. Lineage/provenance - identifies the original sources from which the data was derived and details the processing steps through which the data has gone to reach its current form. The traceability of the data can help determine its accuracy and relevance tasks by different users.
2. Quality - determine fitness for use, conformance to design specifications and fulfillment of requirements of a coordinated effort.
3. Accuracy - degree of correspondence between data and the real world depending on the quality and precision of the instruments used to capture the data.
4. Currency - measures the degree of temporal relevance of the data to the present real world.

5. Precision - represents the exactness of measurement (or description) and is determined by the input. Data can always be output at lower, but not higher, resolution.

6. Scale - is the ratio of distance on a map to the equivalent distance on the Earth's surface

Based on the assumption that the proposed Data_Quality extension to the CIM model is adopted, there are several steps involved in publishing provenance data. More detailed information on the proposed Data_Quality extension can be found in 07-038 OGC Cataloging of ISO Metadata CIM Using the ebRim Profile of CS-W discussion paper. Different components of the model must be published by different actors, as shown in the sequence diagram in Figure 3. In OWS-9, the Data Management Service (DMS) was a primary user of the metadata / provenance workflow. The sets of activities involved in this workflow are summarized below the Figure.

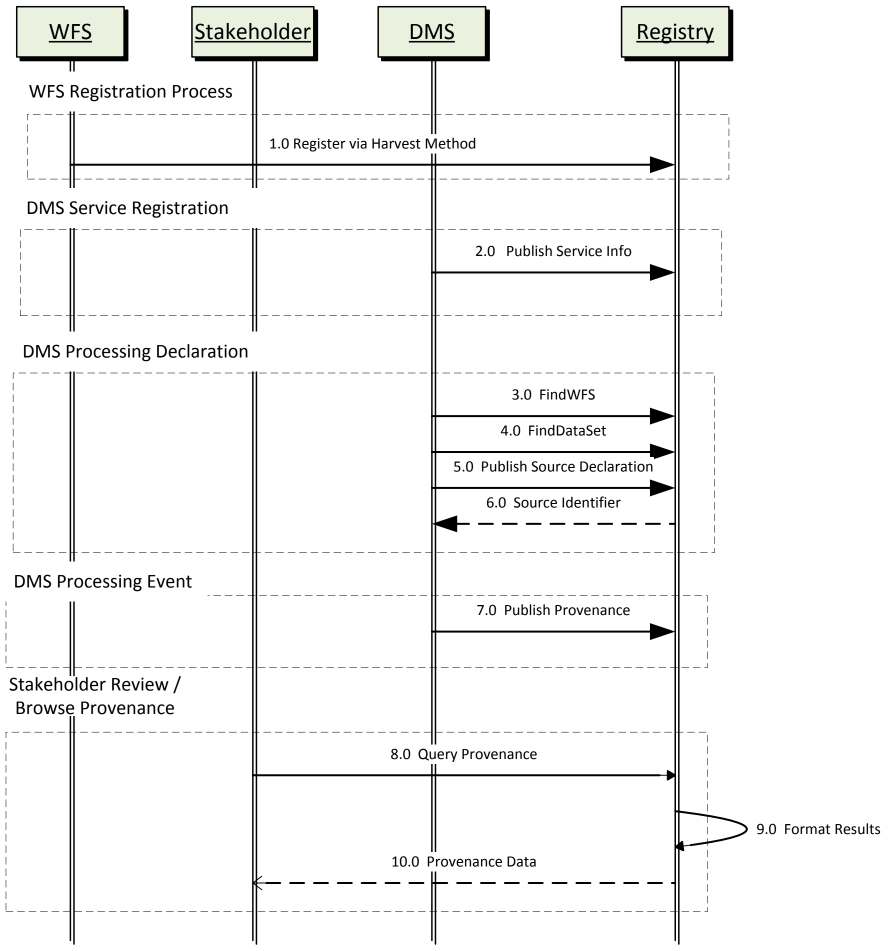


Figure 3 Metadata / Provenance Workflow

1. WFS Registration Process - DataMetadata was published along with DataQuality and Lineage objects using CSW-ebRIM with CIM model.

2. DMS Service Registration - This is a one-time step to publish service identification information (e.g. Organization object).

3. DMS Processing Declaration - For each DMS processing option, the DMS declared its intent to capture provenance for a given WFS/Dataset pair by publishing a Source object describing DMS option (e.g. filtering approach) and associating it to the Lineage object for the selected WFS.

4. DMS Processing Event - For each DMS request processed, provenance details were published to a ProcessSetp instance related to the appropriate DMS Source entry specified by identifier.

5. Stakeholder Review/Browse Provenance - Stakeholders had the capability of querying provenance data in a variety of ways using OGC Filter queries. The Registry formatted the results if response content needed to be something other than XML.

6.1.4 Data Management Service

The Data Management Service (DMS) was introduced in OWS-9 as a new entity in the OGC architecture to provide a value-added proxy service between the client and OGC web services. The purpose of this new entity was to enhance the quality of the exchanges between the information producers or brokers and the information consumers. The workflows illustrated in Figure 4, Figure 5, and Figure 6 summarize the flow of processing and data among the actors associated with the DMS.

6.1.4.1 Session Establishment

The workflow illustrated in Figure 4 describes how the Client performed a handshaking protocol with DMS to select reliable messaging option, compression algorithm to use, and configuration options for data validation and filtering. The DMS created a unique consumer reference for the Client to ensure that the right data was delivered to the right client and to manage subscriptions. The Client contacted the DMS to request a listing of available DMS modules. The DMS responded and the Client selected the modules it wanted to use (e.g. reliable messaging, compression algorithm, validation, and filtering configuration options).

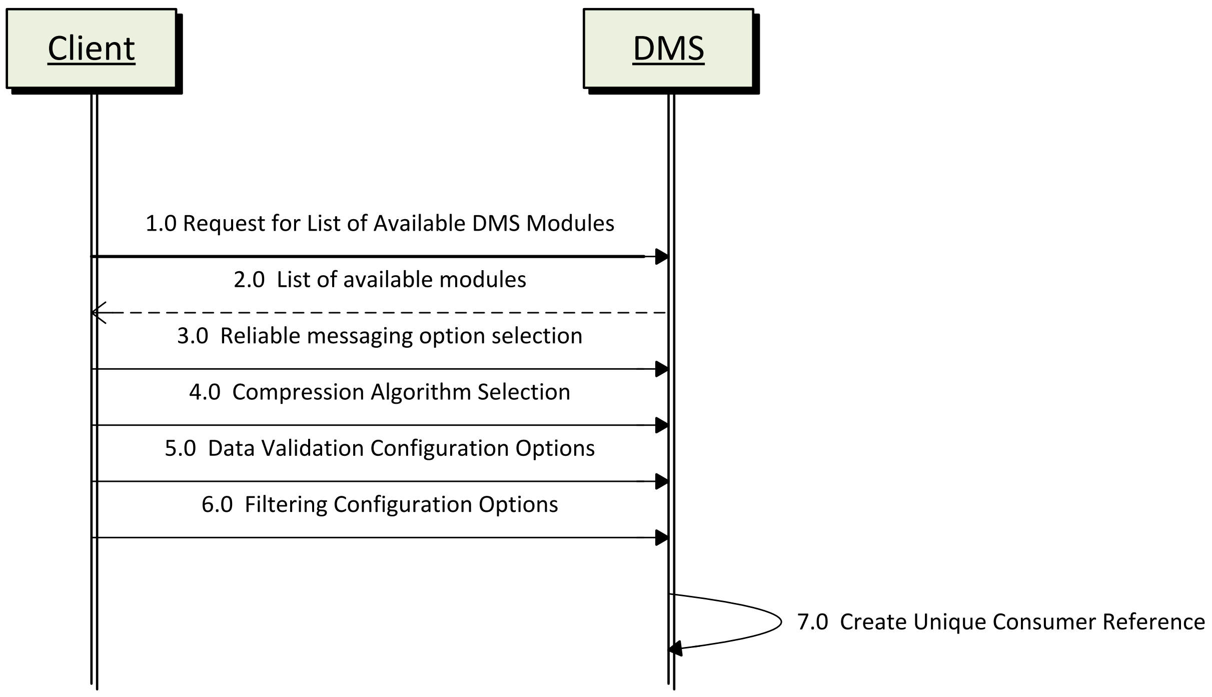

Client
DMS
1.0 Request for List of Available DMS Modules
2.0 List of available modules
3.0 Reliable messaging option selection
4.0 Compression Algorithm Selection
5.0 Data Validation Configuration Options
6.0 Filtering Configuration Options
7.0 Create Unique Consumer Reference

Figure 4 Data Management Service Workflow – Session Establishment

6.1.4.2 Request-Response

The workflow illustrated in Figure 5 describes the process used by the Client to request data from an OGC Web Service (OWS). The Client sent the request to the DMS including an OWS endpoint and the DMS forwarded the request to the appropriate OWS. Upon receipt of the data from the OWS, the DMS performed data validation, filtering, and prioritization in accordance with its re-configured set of technical rules. This processed data was sent to the Client and a copy or summary was sent to the Dispatcher.

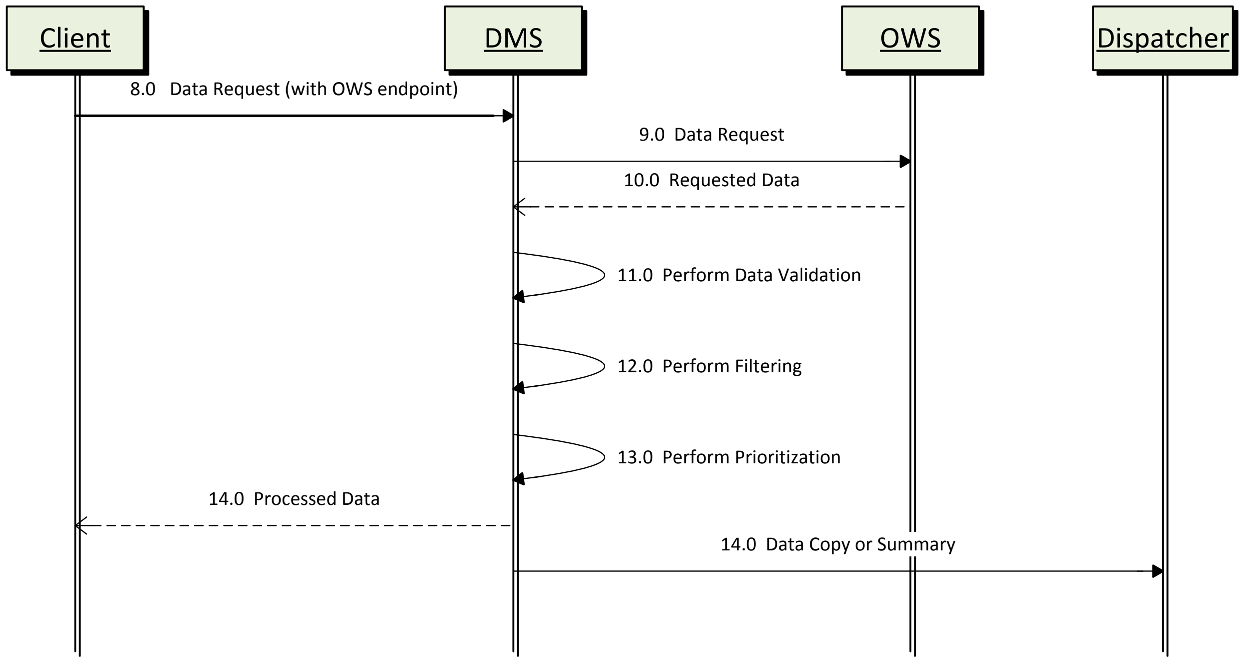

Client
DMS
OWS
Dispatcher
8.0 Data Request (with OWS endpoint)
9.0 Data Request
10.0 Requested Data
11.0 Perform Data Validation
12.0 Perform Filtering
13.0 Perform Prioritization
14.0 Processed Data
14.0 Data Copy or Summary

Figure 5 Data Management Service Workflow – Request Response

The workflow illustrated in Figure 6 describes the process used by the Client to request a subscription from an Event Service (ES). The Client sends the subscription request to the DMS including an ES endpoint and the DMS forwards the request to the appropriate ES. Upon receipt of the data updates or notifications from the ES, the DMS maps the update interval (if required) to the consumer reference to verify that update interval specifications are being satisfied. Additionally, the DMS performed data validation, filtering, and prioritization in accordance with its re-configured set of technical rules. This processed data was sent to the Client and a copy or summary was sent to the Dispatcher.

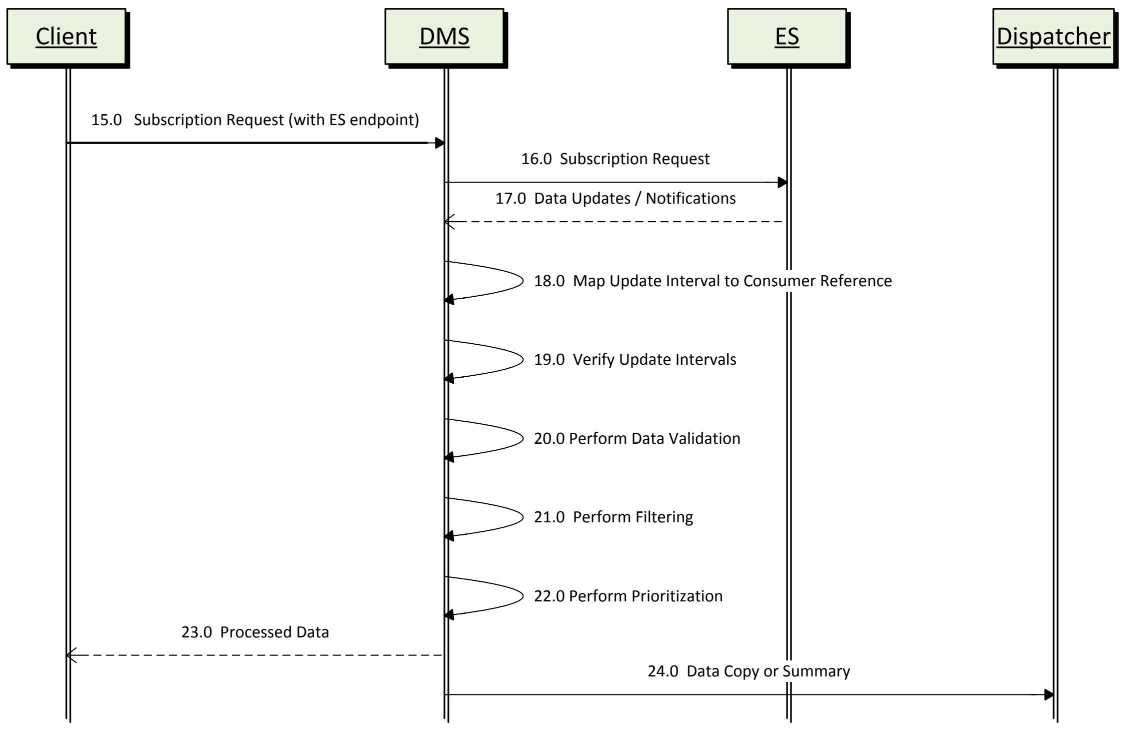

Client
DMS
ES
Dispatcher
15.0 Subscription Request (with ES endpoint)
16.0 Subscription Request
17.0 Data Updates / Notifications
18.0 Map Update Interval to Consumer Reference
19.0 Verify Update Intervals
20.0 Perform Data Validation
21.0 Perform Filtering
22.0 Perform Prioritization
23.0 Processed Data
24.0 Data Copy or Summary

Figure 6 Data Management Service Workflow – Subscription Request / Receive Notifications

6.2 Component Descriptions

6.2.1 COMSOFT Web Feature Service

6.2.1.1 Overview

COMSOFT's Aeronautical Information Management Database (CADAS-AIM$_{DB}$) is a fully featured AIXM 5 database. It has been especially developed to natively support all concepts of AIXM 5, including static and dynamic data, digital NOTAMs and custom application schemas. It is designed to serve as a base for integrating AIM products and components such as electronic Aeronautical Information Publication (AIP), Charting, NOTAM Office, or Briefing with a central Aeronautical Database.

A design principle is the interoperability with other systems. As the database is the core of any integrated Aeronautical Information Manual (AIM) solution, an open interface that can be used independently from any platform and programming language is one of the key features. For an optimal support of AIXM 5 CADAS-AIM$_{DB}$ provides the CAW-interface. In OWS-9, an extension to the WFS 2.0 specification was developed and implemented for better support for dynamic feature data such as AIXM 5.

6.2.1.2 Purpose in OWS-9

In OWS-9, the CADAS-AIM$_{DB}$ was used as a WFS data store and DNOTAM event source. As a WFS data store, it hosts static (BASELINE and PERMDELTA time slices) and dynamic data (TEMPDELTA time slices). Time slices can be retrieved from and stored to the WFS. The retrieval operations support complex filters built of logical, spatial, temporal and comparison operators.

When TEMPDELTA time slices are inserted, corresponding DNOTAM events are created and sent to registered Event Services by a Web Service message.

6.2.1.3 Accomplishments

Several accomplishments in different work areas were achieved.

- Data loading and service deployment
    - Data loaded, validated, transformed and corrected from various sources:
        - Productive European Organisation for the Safety of Air Navigation (EAD) AIXM 4.5 data, mapped by the COMSOFT AIXM 4.5 to 5.1 Mapper to AIXM 5.1 data (North America and Middle-Northern Europe).
        - OWS-9 simulated airport, DONLON, data set, corrected according to AIXM schema and temporality model and enriched with metadata
        - US ePIB data set, General Mitchell International Airport and Bradley International Airport, corrected according to temporality model and enriched with metadata
    - Deployment of two WFS instances, one for North American data and one for Middle-Northern Europe data.
    - Registering Event Services as recipients of DNOTAM messages, which are generated on data insertion at the Web Feature Service

- Implementation
    - Calculation of airspace extents in full 2.5D and time, including composite airspaces (unions, intersections and subtractions) and airspace parts based on borders (references to GeoBorder features).
    - Support for altitude queries on airspaces
    - Calculation of extents of non-spatial feature types (runways, taxiways and aprons) and support for spatial filtering of non-spatial feature types.
    - Support for AIXM features as geometry operands by feature reference.
    - Support for the WFS Temporality Extension.
    - Support for selective metadata retrieval.
- Conceptual Work
    - Improving and finalizing the WFS Temporality Extension OGC Discussion Paper, incorporating feedback from other participants.
    - Specification of altitude queries, of using AIXM features as geometry operands and of selective metadata retrieval.

6.2.1.4 Challenges

A key challenge was the airspace extent calculation in 2.5D. The correct calculation of composite airspaces, which in turn may be composed of other airspaces and which may in turn include references to geographical borders, turned out to be a challenging task, especially when taking the vertical dimension into account. As the specification is a mixture of AIXM and GML, a dedicated algorithm had to be developed, combining 2D calculations on the earth's ellipsoid with AIXM specific properties for the vertical extent and composition operators.

6.2.2 Snowflake Web Feature Service

6.2.2.1 Overview

For the Aviation Thread of OWS-9 Snowflake Software used its commercial off-the shelf (COTS) products: GO Publisher and GO Loader which are comprised of a series of flexible, scalable components supporting the transformation and data exchange requirements of aeronautical information systems as illustrated in Figure 7.

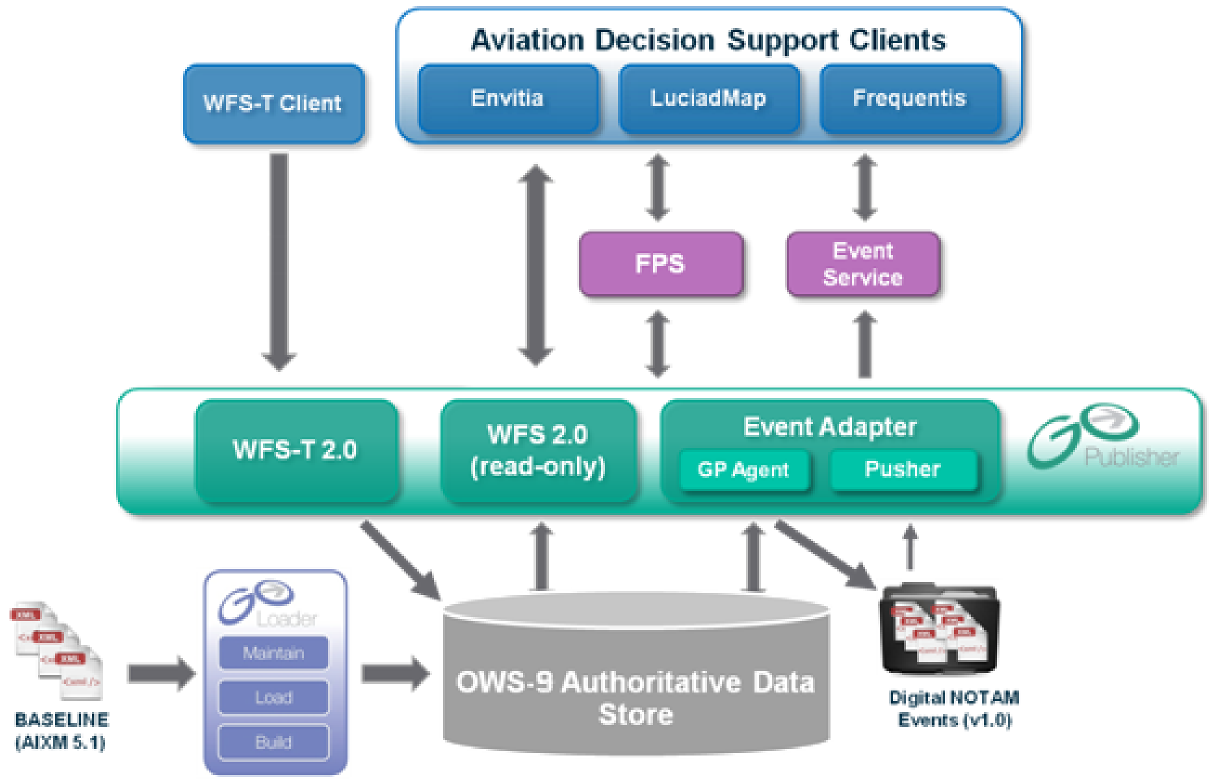


Figure 7 Overview of the Snowflake Aviation Component Architecture

Aeronautical data was received from a range of sources and integrated into two consolidated AIXM 5.1 databases representing North American and European data sources. The Snowflake Software component architecture consisted of three components.

6.2.2.1.1 WFS 2.0 (read-only)

Two different instances of the read-only WFS 2.0 were established to represent United States and European data sources. These WFS instances were configured so that each feature contained only one timeslice. Thus multiple features were published that represented the real world objects.

- European WFS.
- United States WFS.

6.2.2.1.2 WFS-T 2.0 (transactional)

A separate transactional Web Feature Service – Transactional (WFS-T) 2.0 instance was established to support the insertion of events. The WFS-T was integrated into the data maintenance architecture for the consolidated database. On insertion of a Feature, a series of processes were then triggered to auto-generate additional information to publish the data via the European and United States WFS and Event Publisher in real-time.

- WFS-T.
- European WFS (T).
- United States WFS (T).

6.2.2.1.3 Event Publisher

The Event Publisher is composed of two components:

- **GO Publisher Agent:** a server-side bulk data publishing system that generates event messages.
- **Event Pusher:** registers with one or more event services as an event source and pushes the messages generated to the event service brokers.

Publication of Events is triggered by the insertion of a feature into the database via a WFS-T insert transaction. The GO Publisher Agent generates a Digital NOTAM Event and transfers it to an Event pusher which forwards the message to the Event Services.

6.2.2.2 Component functionality

No new component functionality was developed for either GO Publisher WFS or the Event Publisher during OWS-9. GO Publisher WFS currently implements a large proportion of the OGC WFS 2.0 specification and the Event Publisher continues to have sufficient functionality to connect to the IfGI and IDS Event Services.

Within OWS-9, the aim was to further evaluate the effectiveness of the basic mandatory WFS 2.0 operations and full filter encoding (FE 2.0) to retrieve AIXM 5.1 data for the various flight planning and dispatch scenarios identified.

6.2.2.3 Data available via the Components

The OWS-9 consolidated database consisted of AIXM data from the following sources:

- Eurocontrol EAD data (North America and Europe).
- Comsoft EAD data (North America, Canada and Middle Europe).
- Comsoft Estonian Air Navigation Services (EANS) - Estonia data (Europe).
- FAA Airport Mapping Database (AMDB) data from OWS-8.
- FAA Chicago National Airspace System Resources (NASR) data.
- Detailed Airport Mapping data for Bradley International Airport (KBDL) and General Mitchell International Airport (KMKE) airport (North America).

The data was split spatially between a North America and Europe feed from the WFS.

WFS also contained test events created to support the various scenarios that were inserted into the consolidated database via the WFS-T 2.0

6.2.2.4 Accomplishments

Several key accomplishments were developed for OWS-9 within the data maintenance and publication architecture developed by Snowflake:

- Data was consolidated from multiple sources and published using different AIXM 5.1 extension schemas into a single authoritative database. Access to these data was provided via the WFS 2.0 and Event Publisher.
- The data was split spatially into North America and Europe sources for the WFS.
- The WFS-T was configured to allow events to be inserted into the database. On insertion additional processes for publication were activated via Event Publisher and WFS.

6.2.2.5 Challenges

Key challenges were identified when loading the AIXM source data.

1. **Features without spatial property**

   There were a number of features types which do not contain a geometry property. It was thus difficult to separate these feature types into a North American and European service.

2. **Deeply nested Airspace by reference geometry elements**

   The source data for the Airspace feature type contain geometries with reference to geometry elements in other features which in turn could contain references to yet more features. This nesting of geometry references could be several levels deep and proved too time consuming to sort out within the scope of the OWS9 project.

3. **Issues with the source data**

   There were a number of issues with the source data geometry. For example Circle-by-Centre-point geometries with a zero radius value and Arc-by-Centre-point geometries with a missing radius. Primarily, but not exclusively, this occurred within the Airspace feature type.

6.2.3 Registry Service

The Galdos INdicio™ Web Registry Service (WRS) was deployed in OWS-9 as an Aviation Registry to discover, manage, and retrieve aviation resources, such as metadata and styling information. The INdicio™ Web Registry Service implements the Open Geospatial Consortium (OGC) ebRIM profile of the Catalogue Service for the Web standard (CSW-ebRIM 1.0.1, OGC doc 07-110r4). Galdos refers to this service as WRS for shorthand notation. The OGC Catalogue Service (OGC document 07-006r1) is an abstract catalogue standard that defines the basic notion of a Record (i.e. registered item) and the ebRIM profile applies a flexible registry information model to supplement the common interface (Get, Insert, Update, Delete Records, etc).

The OWS-9 Aviation Registry was used to host resources shared between multiple OGC services (e.g. ISO metadata pertaining to datasets common to multiple WFS implementations (Snowflake, Comsoft), Styling information shared between multiple Feature Portrayal Services (FPS) implementations (Envitia, Luciad), and ISO Service metadata describing the Services deployed in OWS-9). The efficient retrieval of metadata task was also implemented using the Aviation Registry by designing queries to retrieve only the relevant excerpts of the metadata desired by clients. The participants used Insert/Update/Get transactions to create User accounts, publish common resources, and efficiently retrieve resources using complex filtered queries. No functional or performance issues were encountered during the OWS-9 initiative.

Unlike conventional geographic catalogues, INdicio™ is highly configurable and can be readily deployed to manage a wide variety of objects.

INdicio™ ships with a CSW-ebRIM Basic Extension Package which provides a variety of useful objects for geospatial applications including:

- Service taxonomy and Dataset metadata model (source: ISO 19115/19119/19139) via the preloaded OGC Cataloging ISO Metadata (CIM) Registry Information Model.
- Feature data dictionaries and catalogues (source: ISO 19109/19110/19126).
- Coordinate Reference Systems (source: ISO 19111).
- Country codes (source: ISO 3166-1 "Codes for the representation of names of countries and their subdivisions – Part 1: Country codes").
- Geographical regions (source: UN Statistics Division).
- Property categories based on Dublin Core (source: DCMI metadata terms).

6.2.4 IDS Event Service

IDS provided one of the two ES implementations of the broker component which is based on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Brokered Notification 1.3 specification. Its main functionalities included finding a match between a notification sent by a producer and subscriptions subscribed by a consumer, then disseminating those matching records to that consumer. The IDS ES has been used since OWS-8.

6.2.4.1 Interfaces

The IDS ES consists of five endpoints, and their descriptions can be found at: http://vpn.ubitech.com:9998/IdsBrokerInterface. Only the endpoints that would be used in this test bed are listed below.

Endpoint	*Available Methods*	*URL*
NotificationBroker	Notify, Subscribe, CreateStoreFilter, ListStoredFilters, DescribedStoreFilters and GetCurrentMessage	http://vpn.ubitech.com:9998/IdsBrokerInterface/IdsBroker
PauseableSubscription Manager	Unsubscribe, PauseSubscription, ResumeSubscription and Renew	http://vpn.ubitech.com:9998/IdsBrokerInterface/IdsPausableSubscriptionManager

Users can access the service WSDL by concatenating the "?wsdl" at the end of the service endpoint (e.g. http://vpn.ubitech.com:9998/IdsBrokerInterface/IdsBroker).

All available methods are defined by OASIS Web Services Notification (WS-N), with the exception that all stored filter methods are extended and described in an experimental schema (http://test.schemas.opengis.net/ows-9/aviation/es/storedFilters.xsd). The methods used in this test bed are described in the following table.

Method	*Description*
Notify	Used by a producer to push a NotificationMessage to the ES. The contents are Digital NOTAM Events or position update messages of an aircraft.
Subscribe	Used by a consumer to define a subset of all incoming notifications. A subset can be defined using topic filters, XPath expressions and FES 1.1/2.0 filters.. A Subscription can have an InitialTerminationTime (encoded as a period or a date time) to define its lifetime.
Unsubscribe	Used by a client to cancel a subscription.
CreateStoredFilter	Used by a client to create a stored filter.
ListStoredFilters	Used by a client to discover the availability of all stored filters have been created.
DescribedStoredFilters	Used by a client to retrieve stored filter descriptions.

6.2.4.2 Changes to the Event Service Implementation

In the previous test bed, the ES only supported spatial capabilities of OGC filter. For example, filtering based on geometry intersections of a bounding box wrapping the entire route. Simulation was used to simulate the consumer and producer components. In this test bed, additional features were added:

- FES 2.0 filters.
- Interactions with producer (e.g. WFS) and consumer.

- Advanced Filtering functionality: Event Service Update Interval, Stored Filters, AIXM Features as Geometry Operands, Spatial Filtering of Non-spatial Features, Simple Altitude Queries and Selective Metadata Retrieval.

The following sections provide implementation descriptions of selected features.

6.2.4.3 Stored Filters

The IDS ES implements "CreateStoredFilters", "ListStoredFilters" and "DescribedStoredFilters" methods that are exposed via the NotificationBroker service end point. Clients can use Simple Object Access Protocol (SOAP) messaging to create a stored filter, get a list of all available stored filters, or get a detailed description of a stored filter such as title, filter and required parameters.

A client can subscribe to a stored filter via the normal ES' Subscribe operation, using the new filter dialect, i.e. StoredFilterSubscription.

6.2.4.4 Update Intervals

This feature was implemented as an add-on component instead of modifying the filtering component, since WS-N's "Subscription Policy" has been utilized to define this feature capability. Hence, the filtering component was not impacted by this feature.

The client needs to specify the interval (how often the notifications will be sent out), dissemination method (all notifications since the last time, or only the latest), and treatment (matched subscription only, or any notifications). An experimental schema has been created for this purpose.

The following figure shows how the new update interval component fit into the existing event service.

.

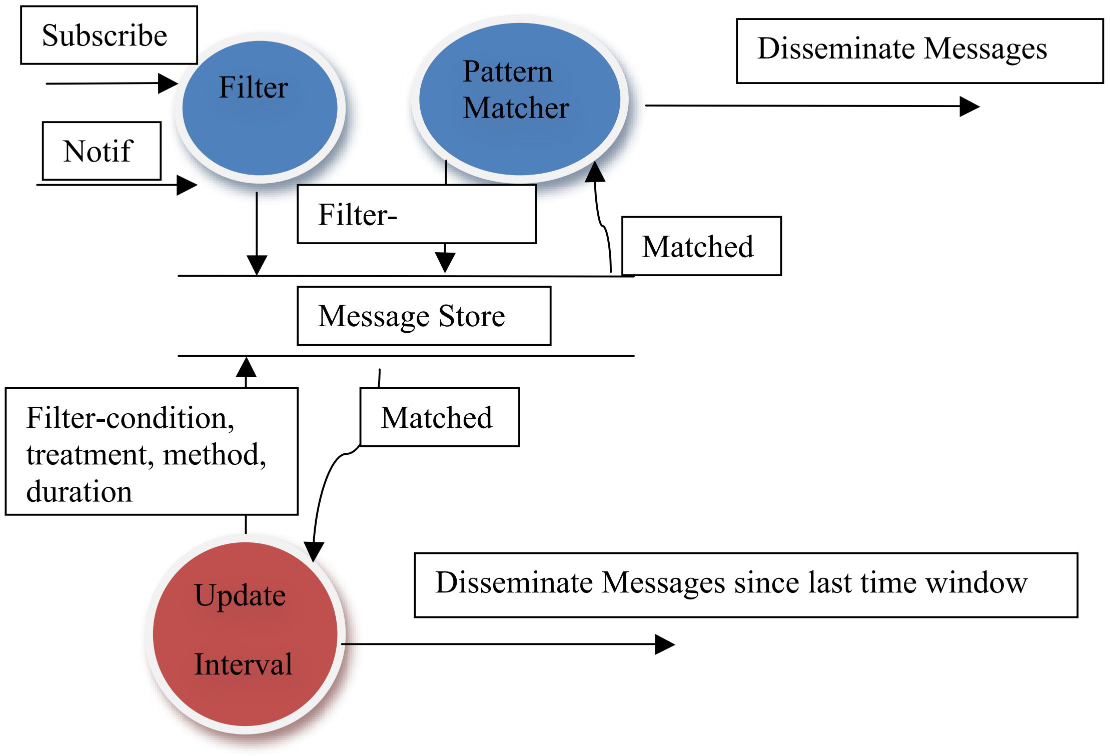


Figure 8 - Update Interval Framework.

6.2.4.5 Simple Altitude Queries

Upon receiving a producer's notification, the Event Service performed additional checks to determine if a particular subscription included altitude information as part of its filtering message. For OWS-9, this was embedded in the route segment. A separate thread was created to perform a WFS GetFeature request. That query performed a logical "AND" operation on the notification's feature identifier and the subscription's route segment filtering. If the WFS query returned a non-empty result that meant the notification matches the given subscription. In that case, it was disseminated to the subscriber.

6.2.5 IfGI Event Service

One of the two ES implementations for OWS-9 was provided by the Institute for Geoinformatics. It was based on the OGC Sensor Event Service discussion paper (OGC 08-133). The source code is maintained and provided in collaboration with the 52°North Initiative for Geospatial Open Source Software GmbH. Following the Publish/Subscribe paradigm of OASIS' Web Service Notification (WS-N) family of standards, it acted as a notification broker. The IfGI ES has been used in aviation threads of former OGC test beds starting with OWS-6, including the FAA Special Activity Airspace (SAA) Dissemination Pilot.

6.2.5.1 Interfaces

The ES consisted of three endpoints. The PublisherRegistrationManager can be used to register data providers at the ES instance but was not used in this test bed and hence not described.

Endpoint	*Available Methods*	*URL*
Broker	GetCapabilities, Notify, Subscribe	http://v-tml.uni-muenster.de:8080/EventService/services/Broker
SubscriptionManager	Unsubscribe, PauseSubscription, RenewSubscription	http://v-tml.uni-muenster.de:8080/EventService/services/SubscriptionManager

To enable SOAP bootstrapping both endpoints provide a WSDL using HTTP Get method (<endpoint-url>?wsdl).

Besides the GetCapabilities method, all available methods are defined by OASIS WS-N. The methods used in this test bed are described in the following table.

Method	*Description*	*involved OWS-9 Component*
Notify	Used to push a NotificationMessage to the ES. The contents are Digital NOTAM Events or position update messages of an aircraft (sent by clients for demo purposes) and separated into different topics (NOTAM-Topic, Weather-Topic). This method does not use request-response communication, as notifications are pushed to the ES.	WFS providers, Clients (for demo purposes), Performance Tool
Subscribe	Used to define a subset of all incoming notifications. A subset can be defined using Topic filters, XPath Expressions and FES 2.0 filters. A SubscriptionReference is returned to the client, enabling the management of it. A Subscription can have an InitialTerminationTime (encoded as a period or a date time) to define its lifetime.	Clients, Performance Tool
Unsubscribe	Used to cancel a Subscription (e.g. when the flight has arrived).	Clients, Performance Tool

6.2.5.2 Changes to the Event Service Implementation

Compared to the ES used in the previous OGC test beds some major improvements have been made during this test bed. The IfGI Event Service implemented all features summarized under the term “Advanced filtering functionality”, namely “Event Service Update Intervals”, “Stored Filters”, “AIXM Features as Geometry Operands”, “Spatial Filtering of Non-spatial Features”, “Simple Altitude Queries” as well as “Selective Metadata Retrieval”. See Section 7 Data Provision via Web Feature Service and Event Service for details on the different features. The following sections provide some implementation insights on selected features.

6.2.5.3 Update Intervals

The implementation of the "Update Intervals" can be understood as an add-on to the existing event dissemination. As this feature has been designed using Subscription Policies, it does not affect any internal filtering mechanisms. Thus, an additional component was designed to apply the Update Interval policy of a given subscription (see Figure 8).

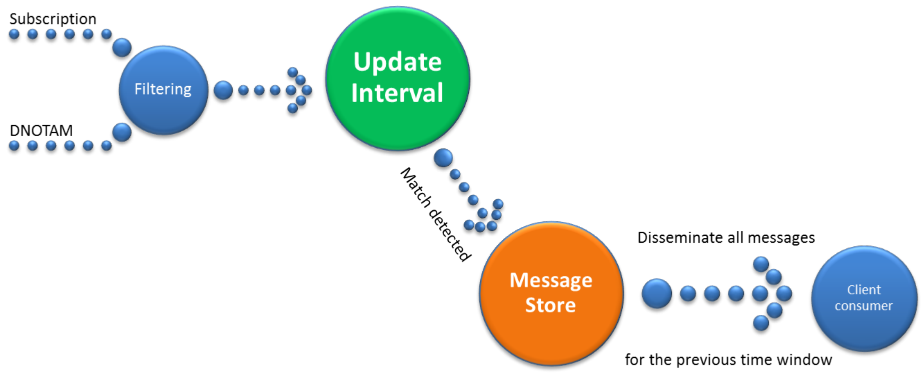


Figure 9 Update Interval Internals

6.2.5.4 Simple Altitide Queries

The Altitude Queries implementation leveraged the WFS for applying the developed approach. A WFS providing Simple Altitude Queries capabilities provided all needed methods to enable outsourcing of the actual altitude computation at the Event Service. This feature was integrated at the time of DNOTAM reception: A previously defined subscription with altitude information (for OWS-9, aixm:RouteSegment was identified to provide all necessary parameter) was evaluated with a GetFeature request at the WFS. For instance, when receiving an update on an aixm:Airspace, the query defined the following parameters:

- Property "gml:identifier" was equal to the one received in the DNOTAM.
- Spatial Operator evaluated true for the aixm:RouteSegment of the Subscription.

If and only if the query returned a non-empty result, the DNOTAM matched the given subscription and was disseminated to the subscriber.

6.2.6 Luciad Web Processing Service

6.2.6.1 Introduction

An OGC Web Processing Service (WPS) is a service that enables the user to perform a process (such as a computation) on a service. One use case is to offload heavy computation to a server, so that lightweight clients can rely on it without having to implement or perform the computation themselves. Each process is described in a WPS profile, which lists the available input parameters and the structure of the response.

Within the OWS-9 Aviation thread, the WPS was used for a number of processing tasks. Luciad provided a WPS service component based on its COTS software product LuciadLightspeed to support the following OWS-9 use cases:

- **ePIB map generation**
    - Use case description: enable rendering of a list of Digital NOTAMs submitted by a user on top of an airport map. The response should be a bitmap image, ready for embedding in an ePIB.
    - Solution: the logic to retrieve the airport map data and styling information was implemented by a WPS profile. The result was an SLD that embedded the combined feature data (Digital NOTAMs and airport layout features) and SE styling information, which was sent to an FPS for final rendering.
- **Geometry processing**
    - Use case description: support calculating the topology relation and intersection between two AIXM features, such as between a route and an airspace as illustrated in Figure 10.
    - Solution: two WPS profiles are defined and implemented, one to check the topological relationship between two AIXM features (e.g. do the features intersect, is one feature contained in the other, are they touching, etc.) and one to calculate the actual intersection points between two AIXM features.

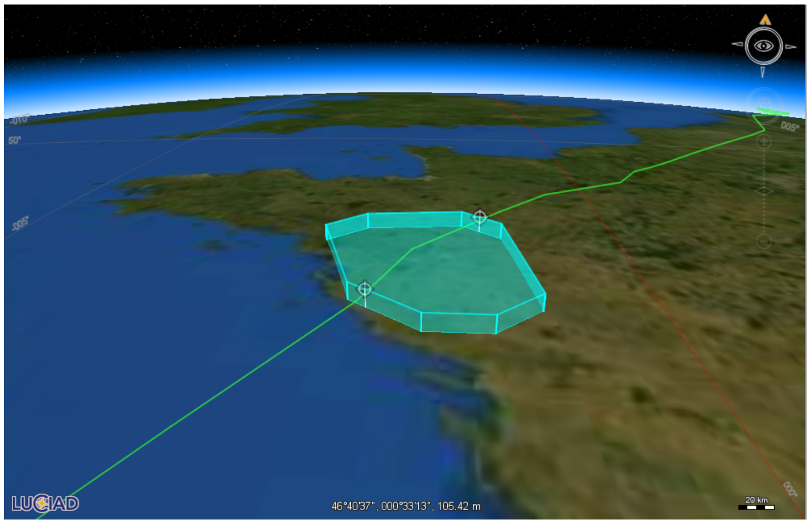


Figure 10 Example of an Intersection Between an AIXM Route and Airspace

- **Geometry retrieval**
    - Use case description: support the retrieval of the geometry for non-spatial AIXM features. For instance, an AIXM Runway feature represents a runway entity but does not contain the actual geometry; this is stored in RunwayElement features. The use case is to be able to determine and retrieve the geometry, given the Runway feature.
    - Solution: a WPS profile has been defined and implemented that accepts an AIXM feature and that determines and retrieves its geometry from a WFS.

6.2.6.2 Functional overview

The Luciad WPS service component had the following functionality, developed during OWS-9:

- WPS 1.0 service interface with support for the requests GetCapabilities, DescribeProcess and Execute.
- Implementation of a WPS profile to support ePIB map generation:
    - Accepted Digital NOTAM events and map configuration settings.
    - Interacted with OGC Catalogue Service for the Web (CSW) and WFS services to discover and query the necessary feature data and styling information.
    - Interacted with an FPS service to render the ePIB map.
- Implementation of a WPS profile to support geometry processing:
    - Calculation of topological relations between two AIXM 5.1 features in ellipsoidal space. Supported topological relations: INTERSECTS, DISJOINT, EQUALS, TOUCHES, CROSSES, OVERLAPS, CONTAINS, WITHIN, COVERS, COVERED_BY. The calculation framework is based on the Dimensionally Extended Nine Intersection Matrix (DE-9IM), which is defined in the OGC Implementation Specification for Geographic information – Simple feature access (http://www.opengeospatial.org/standards/sfa).
    - Calculation of the intersection points between two AIXM 5.1 features in ellipsoidal space.
- Implementation of a WPS profile to support geometry retrieval:
    - Determined the geometry for non-geometry AIXM 5.1 features.

6.2.6.3 Deployment characteristics

The WPS service component is based on Java Servlet technology. To run, it requires a servlet container or application server compatible with Java Servlet 2.4 or higher. Apache

Tomcat 6 has been used by Luciad during OWS-9. Other than being capable of running a Java Virtual Machine 1.6 (or higher) and an appropriate servlet container / application servlet, no requirements are posed on the underlying hardware or operating system.

6.2.6.4 Challenges

One particular challenge identified when working on the ePIB WPS process was to get optimal performance. Due to several additional service interactions needed in this WPS process (discovery of styles and WFS URLs via a CSW, retrieval of feature data from a WFS, generation of the map via an FPS), it was a continuing challenge to achieve good performance in calculating and rendering the ePIB map. Caching of fairly static data (e.g. baseline AIXM data or styling) helped, but other possibilities exist to improve the usability for the user (e.g. asynchronous generation of the map) (WPS allows this) or a closer integration of services to reduce data bandwidth consumption (e.g. integration of a WPS and FPS).

6.2.6.5 Accomplishments

The key accomplishments for Luciad's WPS service component in OWS-9 included:

- Demonstration of multiple use cases to support ATC/ATM operations, such as the SESAR ePIB map generation use case and route/airspace intersections.
- Collaboration with OWS-9 WPS and portrayal stakeholders and Eurocontrol to develop the custom OWS-9 Aviation profiles needed for the WPS.
- Fast deployment of a WPS service with full support for the geometry processing use case within 2 months after the start of the project. This was possible by the use of the LuciadLightspeed COTS product, which provides extensive support for the AIXM format and a wide range of geometry calculations.

6.2.7 52°North Web Processing Service

One of the two WPS implementations for OWS-9 was provided by the 52°North Initiative for Geospatial Open Source Software GmbH. It was based on the OpenGIS Web Processsing Service 1.0.0 specification (OGC 05-007r7).

The endpoint URL of the service is http://geoprocessing.demo.52north.org:8080/aviation-wps/.

To enable SOAP bootstrapping, the service provided a WSDL using HTTP Get method at http://geoprocessing.demo.52north.org:8080/aviation-wps/services/WPS?wsdl.

6.2.7.1 Interfaces

The WPS specification defines a set of interface methods. The following table provides a short summary of the methods used within this test bed.

Method	*Description*	*involved OWS-9 Component*
GetCapabilites	Used for discovery of available Processes.	Clients, Registry
DescribeProcess	Used to get details on the inputs and outputs of a specific process and their formats	Clients, Event Services
Execute	Used to start the execution of the specified process. Here, the actual inputs and the desired output formats are provided.	Clients, Event Services

6.2.7.2 Service Architecture

The 52°North WPS was based on a three-tier architecture (see Figure 11). To fulfill the requirements of this thread certain components fitting in this architecture had to be developed:

- Data Parser instances to transform the input AIXM features into the internal feature representation; for OWS-9 aixm:Airspace and aixm:RouteSegment are supported.
- Algorithm implementations based on the defined ProcessDescriptions .
- Data Generator instances to support the required result output formats. This included a Generator to provide the result as a gml:MultiGeometry.

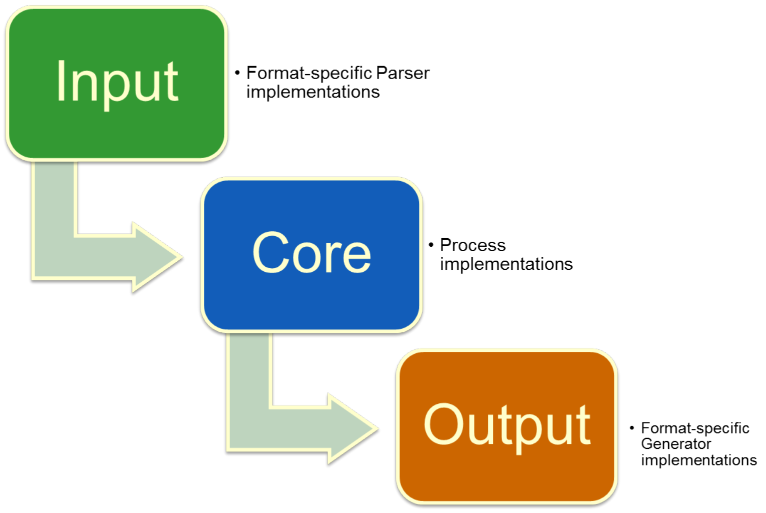

Input
• Format-specific Parser implementations
Core
• Process implementations
Output
• Format-specific Generator implementations

Figure 11 52°North WPS Three-tier Architecture

6.2.7.3 Implementation Specifics

To provide robust spatial computation capabilities, a well-established computation backend was implemented. For OWS-9, ESRI's ArcGIS Server 10.0 (see Figure 12) was considered a good solution since it provided strong support for reference systems and thus was capable of applying geodetic calculations.

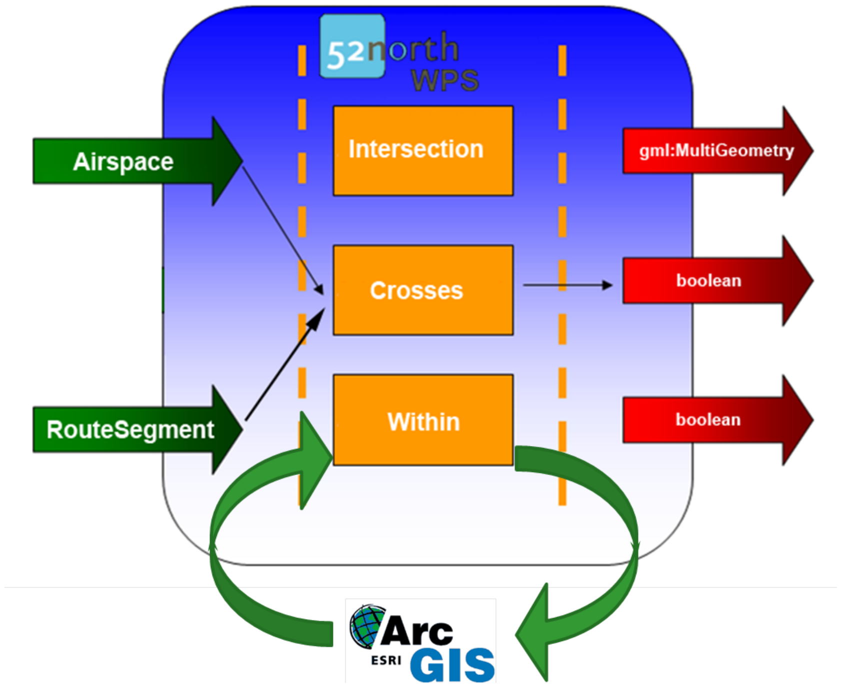


Figure 12 52°North WPS with ArcGIS Server Backend

6.2.8 ePIB Client

6.2.8.1 Introduction

A Feature Portrayal Service (FPS) is an OGC service that enables the user to render maps based on feature data and styling information. It is an extension of OGC's Web Map Service (WMS) that makes it possible for users to supply their own feature data and styling information. This user input is defined in an OGC Styled Layer Descriptor (SLD) document, which gives access to the feature data, either embedded or as a link to an OGC Web Feature Service, and the styling information, encoded with OGC Symbology Encoding (SE).

Within the OWS-9 Aviation thread, an FPS was used to support the Single European Sky ATM Research Programme (SESAR) ePIB map generation use case. The main requirement for this use case was to have access to a service that could render a list of Digital NOTAMs submitted by the user on top of an airport map. The response was a bitmap image, ready for embedding in an ePIB.

The logic to retrieve the airport map data and styling information was offloaded to an OGC Web Processing Service (WPS) component (see component description 6.2.6 and the OWS-9 Aviation Portrayal ER [OGC 12-151]. The result of this was an SLD that was sent to the FPS which embedded the combined feature data (Digital NOTAMs and airport layout features) and SE styling information. The FPS interpreted the SLD and rendered it into a bitmap image. An example for Chicago O'Hare airport is shown in Figure 13. The image shows AIXM 5.1 airport layout features for this airport and one Digital NOTAM event indicating a temporary obstacle (crane).

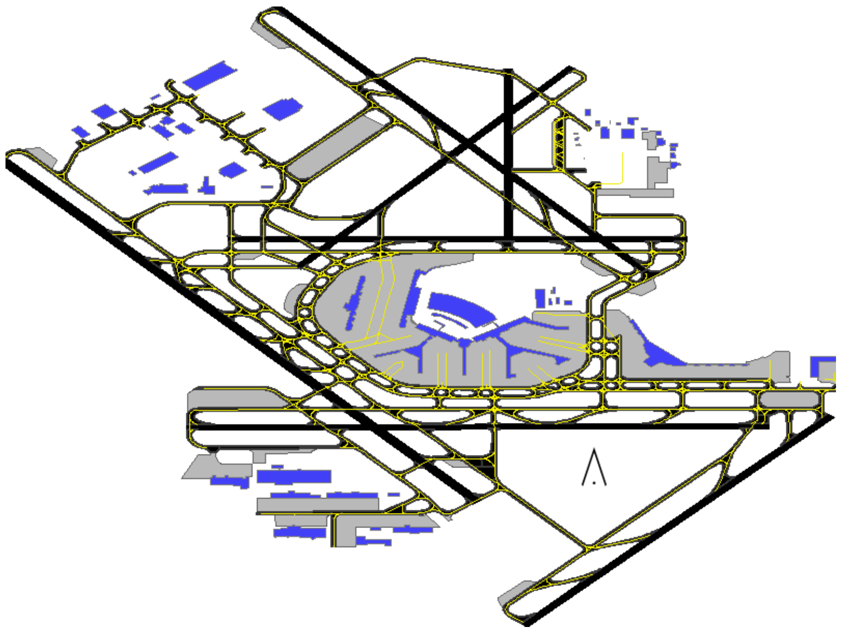


Figure 13 Example of an ePIB Bitmap Image Generated by the FPS

For the OWS-9 Aviation thread, Luciad provided an FPS service component using its COTS software product LuciadLightspeed. LuciadLightspeed offers a rich set of standards-based software components, including an OGC Web Services Suite equipped with an OGC-compliant Web Map Service (WMS) 1.1.1 & 1.3.0 service component with support for the SLD / SE extension.

6.2.8.2 Functional overview

The FPS did not need any new functionality other than the capabilities already defined by OGC's WMS, SLD and SE standards. The custom business logic, i.e. the querying of the necessary feature data and style information and the construction of an SLD was offloaded to a WPS.

The FPS provided by Luciad had the following functionality:

- OGC-compliant WMS 1.1.1 & 1.3.0 service interface with support for the following requests: GetCapabilities, GetMap and GetFeatureInfo. Supported request encodings are GET and POST.
- Support for the SLD 1.0 / 1.1 profile (FPS), including support for user-defined styles and user-defined layers. User-defined layers can either embed the feature data or link to an OGC WFS.

- Support for SE 1.1 to define styling rules.
- Support to render any type of GML-based feature data (AIXM, WXXM …).

6.2.8.3 Deployment characteristics

The Luciad FPS is based on Java Servlet technology. To run, it requires a Java servlet container or application server compatible with Java Servlet 2.4 or higher. Apache Tomcat 6 has been used by Luciad during OWS-9. Other than being capable of running a Java Virtual Machine 1.6 (or higher) and an appropriate servlet container / application server, no requirements are posed on the underlying hardware or operating system.

6.2.8.4 Challenges

One important and reoccurring challenge when working with an FPS in general is to get optimal performance. The high flexibility of being able to submit any kind of feature data and styling information comes with difficulties to achieve this. The AIXM format is verbose, so large amounts of bandwidth can be needed to provide the FPS with the necessary feature data. Reducing the amount of data (e.g. filtering properties to the bare minimum that is actually needed by the FPS) and/or applying compression are possible ways to resolve this. Additionally, a SE style can also have an impact on performance: complex styles since many rules / filter combinations can be defined that can slow down their interpretation and therefore impact rendering performance.

These challenges were not addressed in detail during OWS-9, as the main focus was to functionally address the ePIB map generation use case.

6.2.8.5 Accomplishments

The key accomplishments for Luciad's FPS service component in OWS-9 included:

- Fast setup and deployment of Luciad's COTS-based WPS service, ready-to-use by other participants within 2 months after the start of the project.
- Delivery of a FPS service component with a rich feature set to enable testing and verifying multiple ePIB map generation approaches. Key features are:
    - Support for SLD's with either embedded (inline) feature data or links to a WFS. Both approaches were tested and compared in the project.
    - Support for complex AIXM geometries, such as composite airspaces, the use of arc segments, geometry referencing via XLink. As these regularly occur in real-world AIXM datasets, this helped to have an operational FPS service in time.

- Provision of data to test and demonstrate the ePIB scenario and the FPS service in particular:
    - Creation of a modified version of Eurocontrol's DONLON sample data, shifted from the Atlantic Ocean to Europe for demonstration purposes.
    - Provision of airport layout features for Chicago O'Hare airport, by converting public sample data for this airport available in the AMXS format to AIXM 5.1.
    - Creation of example Digital NOTAM events (temporary crane, unserviceable navaid) for Chicago O'Hare and DONLON airports.

6.2.9 ATM-TGS Data Management Service

6.2.9.1 General Context

The Data Management Service (DMS) was introduced in OWS-9 as a new entity in the OGC architecture. The purpose of this new entity was to enhance the quality of the exchanges between the information producers or brokers and the information consumers. On the one side, there are the information producers and brokers, among which are the Web Feature Services (WFS) and the Event Services (ES). On the other side are the clients that consume the information provided by those entities. In OWS-9, the scope of the DMS action has been limited to the WFS and ES. However, the philosophy applicable to those entities could, in the future, be extended to other information providers and brokers, such as the Web Processing Service, Web Mapping Service, and Feature Portrayal Service.

Considering information exchange on the ground, the communication technologies available today provide sufficient resources for information transfer without having to consider much optimization. The bandwidth is generally very high and the communication services are available and reliable. However, especially in the context of aviation where communications are not limited to the ground but are extended to ground-to-air and air-to ground, new challenges appear (e.g. low bandwidth, uncertain availability of data link between the ground and aircraft). Addressing those challenges is the reason for the introduction of a new entity in Service Oriented Architectures; the Data Management Service.

The DMS role was to cope with the limitations of the communication link between the ground and the air, looking into more efficient communications means, increasing the reliability of the data link, and ensuring that the final client (e.g. the aircraft) was provided with the exact information it needed.

To achieve this, the DMS stood between the client and the other entities, as depicted in Figure 14.

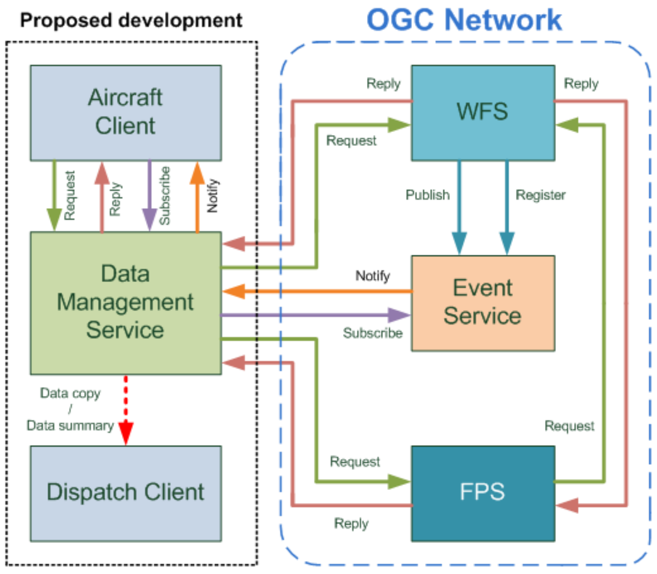


Figure 14 DMS Context Description

6.2.9.2 Functionalities

To manage the information exchange between producers/brokers and consumers of the information, the DMS provided the following functionalities:

- ***Reliable messaging***: The DMS established a reliable communication framework that enhanced data link reliability. This encompassed resending information not received by the client, scheduling the messages and handling network failures.
- ***Filtering***: The client had the capability to define certain filtering rules applied to any information intended to it relayed by the DMS. This allowed removal of certain information in the data set not deemed essential by the client.
- ***Compression***: In order to reduce the volume of data sent to the aircraft (usually on data links with low and expensive bandwidth), the client had the capabililty to define a compression algorithm used by the DMS when forwarding information. In OWS-9, the compression method used was Fast Infoset, based on compression benchmarking undertaken in OWS-8.
- ***Validation***: In order to use the data link with maximum efficiency, invalid or irrelevant information arriving at the DMS should not be forwarded to the client. To achieve this, the DMS included a validation module that performed a check on the information based on criteria defined with the client. If those criteria were not satisfied, the information was not sent to the client.
- ***Prioritization***: In the amount of information that arrives at the DMS, some may be more important to client. The prioritization module function assessed the level of importance of different data sets intended to the client based on criteria defined by

the client and sent those data sets in order of importance in the event of concurrent arrival at the DMS.

- ***Synchronization***: In the aviation context, to increase situation awareness, the flight management entity (e.g. dispatch) may have interest in being aware of the information that is requested and communicated to aircraft it is responsible for. To tackle that challenge, the DMS offered the capability to dispatch units to subscribe to clients to receive copies or summaries of the information received by the client.

6.2.9.3 Deployment

The DMS server and its functions has been implemented and is running on TriaGnoSys premises.

6.2.9.4 Interfaces

There are 3 endpoints for the DMS, one for each type of entity that can communicate with it.

Endpoint	**Available operations**	**URL**
Aircraft Client	getSessionOptions createSession clientRequest clientSubscribeRequest	http://192.37.61.154:8080/Services/DMServices
Event Service	Notify	http://192.37.61.154:8080/Services/GroundDMServices
Dispatch Client	getConsumerReference subscribeToClient unsubscribeFromClient	http://192.37.61.154:8080/Services/DispatchServices

The different operations are described here:

Operation	**Description**
getSessionOptions	This operation was called by the client to obtain the list of options available at the DMS for Data Management
createSession	This operation was called by the client to define the activated module for its session with the DMS and the potential options associated
clientRequest	This operation was called by the client to send a request (feature request, unsubscribe request …) through the DMS.
clientSuscribeRequest	This operation was called by the client to create a subscription to an Event Service that will be managed by the DMS.
Notify	This operation was called by the Event Service to send a notification (that will then be forwarded to the client)

getConsumerReference	This operation was called by the dispatch client to obtain the consumer reference of a given client at the DMS in order to make subscription on behalf of this client.
subscribeToClient	This operation was called by the dispatch client to subscribe to a given client. The DMS was then be notified with any information that arrived at the DMS intended for this particular client.
unsubscribeFromClient	This operation was called by the dispatch client to unsubscribe from a client.

6.2.9.5 Component functionality

Within the OWS-9, the following DMS functionalities have been implemented and tested:

- DMS service discovery; which mainly concerns the process of finding and setting the processing options to be done at the DMS by the aircraft and dispatch client.
- DMS basic pass-through; which handles the forwarding of request/response and notifications between aircraft client and OGC web services.
- Reliable messaging.
- Data compression / expansion.
- Data filtering.
- Dispatch synchronization.

6.2.10 Harris Data Management Service

6.2.10.1 Overview

For the Aviation Thread of OWS-9, Harris Corporation investigated and developed an interoperable data transmission management solution for the efficient exchange of data between clients (e.g. aircraft, dispatch) and information services(e.g. System-Wide Information Management (SWIM)), located on the ground. This solution was implemented as a web service, the Data Management Service (DMS), in accordance with OGC's web service architecture, and provided a set of service interfaces that facilitated client – DMS connection creation, DMS service discovery, message compression, data validation, message prioritization, data provenance tracking, message content filtering, and ground synchronization. The DMS provided aircraft clients an efficient and reliable means of communicating with ground services through an interoperable interface.

6.2.10.2 Data Management Service

The DMS Functionality was separated into two categories: Base Features and Modules. This enabled the DMS to standardize certain functions, such as reliable messaging, while facilitating the customization of more customizable functions such as data compression.

Base Features include:

- Client session creation.
- Module description / configuration.
- Client session information retrieval.
- Client - ground services message relay.
- Reliable messaging.

Module Features include:

- Data Compression.
- Message Validation.
- Provenance Metadata tracking.
- Dispatch Client.
- Message Content Filtering.
- Message Prioritization.

The DMS Base Features created a baseline of DMS capabilities to accomplish the primary goal of the data transmission management: to allow aircraft clients to reliably and efficiently communicate with ground services. The DMS Base Features were designed to be implemented as common functionality across all Data Management Services. In conjunction with the Base Features, the Module Features allowed for an individual DMS provider to provide implementation-specific functionality beyond the Base Features. When aircraft clients initiated a connection into a DMS, they performed a GetSessionOptions request during initial session negotiation. In response, the DMS returned all available module features to the client as a list of supported module specific options. This allowed aircraft clients wishing to connect to a DMS to discover how to interact with unique DMS Module Features. Aircraft clients invoked the DMS functionality by creating a session profile with the DMS which defined how Module Features are applied during data transmission. All messages were forwarded to the client using the WS-Reliable messaging protocol to ensure reliable communications.

6.2.10.3 DMS Service Discovery

When an aircraft client first connected to the DMS, a GetSessionOptions function was called to discover what Module Features were available and how to use them. The client created a DMS session profile which saved the settings to define the configuration for how modules are to be used by the DMS when forwarding or receiving messages from the aircraft client. An aircraft client may communicate with a DMS using multiple communication patterns including Features negotiation, Request-response, Subscription Request, and Event Notifications. Once configured by the aircraft client, the DMS proxied the request-response pattern to a WFS. Subscription requests and notifications are proxied to an Event Service.

6.2.10.4 Component Functionality

6.2.10.4.1 Data Filtering Module

The Data Filtering Module was a client-configurable Module Feature designed to extend the message filtering capabilities of ground services. Message content was filtered by the module using an aircraft client provided XSLT string as a guide. By allowing clients to provide their own filtering instructions to the DMS, the Filtering Module could be customized to fit the client's individual needs.

6.2.10.4.2 Data Validation Module

The Data Validation Module is a client-configurable Module Feature designed to track the validity of messages being transmitted to the aircraft client. The module determined the currency of messages received which is provided by TimeSlice data included in AIXM messages. Depending on a client's session profile, non-current messages were either dropped or flagged and forwarded to the client. The message latency of transmission from DMS to reception by the client was evaluated against a configurable variance amount stored in the aircraft client's session profile. This provided a metric to determine the timeliness of data transmission to clients. The DMS collected and stored data from the Data Validation Module to track the number of non-current messages received or transmitted and the number of messages that exceed the client's latency tolerance.

6.2.10.4.3 Data Prioritization Module

The Prioritization Module was a client-configurable Module Feature designed to ensure that messages of high importance were delivered to the aircraft client when transmitted over bandwidth restricted links. This module prioritized messages based on settings in a client's DMS session profile. Using this configuration, the Prioritization Module assigned priority levels to messages scheduled to be transmitted to a client. The session profile stored a list of message types that were considered higher priority than all other messages. When configured, the DMS created a queue for all messages being transmitted to a client. When a message is removed from the queue and transmitted to the client, the DMS waited for the client's acknowledgement of message reception before transmission of the next message in the queue. When new messages are added to the transmission queue, messages that are defined as high priority were sent to the front of the queue. Messages defined as low priority were added at the end of the transmission queue.

6.2.10.4.4 Data Provenance Module

The Data Provenance Module is a client-configurable Module Feature designed to track changes to message provenance by the Filtering Module prior to client transmission. Whenever the Filtering Module removes content from a message, the aircraft client was be informed by the Provenance Module of the alteration. The Provenance module generated metadata that describes the alterations made by the Filtering Module to the content of a message. The generatation of this metadata by the Provenance Module, followed ISO Standard 19115/19135 and was inserted into SOAP message headers before transmission to the client.

6.2.10.4.5 Dispatch Module

The Dispatch Module is a feature that allows a ground based client (such as an airline dispatcher) to subscribe to and receive copies of the messages sent to an aircraft client. This enabled the dispatcher client to track the information that has been received by the aircraft client. The Dispatch Module provided the dispatch client with the aircraft client's consumer reference endpoint. The dispatch client used this endpoint to subscribe an aircraft client to a Web Feature Service or Event Service data product if it determined the aircraft client would benefit from the additional information.

6.2.10.5 Challenges

The key challenge in the development of the Data Management Service (DMS) was creating a solution that can fully communicate with both the North American and European System Wide Information Management (SWIM) environments. Although the overarching system concepts for SWIM are the same in both North American and Europe, the actual SWIM implementations currently employ different technical standards for their communication interfaces. During the development of DMS for OWS-9, the web service technical standard was used as the communication protocol between aircraft, DMS, and ground services. The use of web service as a communication protocol works well as connection to European SWIM ground services. Future OGC activity should focus on the integration of Message-Oriented Middleware (MOM), which is the communication protocol primarily used by North American SWIM ground services. The extent of the technical gap between what was created in OWS-9 and what needs to be developed for integration of FAA SWIM ground services was documented by OWS-9 in the SWIM Compliance Assessment document. The creation of a solution that bridges the gap between North American and European SWIM messaging protocol is a strong candidate for future OGC work.

6.2.10.6 Accomplishments

Several key accomplishments were achieved within OWS-9 Data Transmission Management during the development of the Data Management Service by Harris Corporation:

- Creation of an operational Data Management Service that was accessible by clients and provided reliable and efficient management of communications between aircraft and services located on the ground.
- Submission of an engineering report that provided technical details describing how the DMS standard was implemented, as well as feedback and comments that described lessons learn and recommendations for future improvements.
- Collaboration with other developers of the Data Management Service and aircraft clients Atmosphere, TriaGnoSys, and Luciad to develop a common interface for the connection of aircraft clients into the DMS for global interoperability.
- Feedback provided to OWS-9 stakeholders to increase understanding of the DMS requirement while providing feedback for the shaping of future DMS requirements.
- Designing the DMS as an interoperable system by providing SWIM Web Services Connectivity in the OGC architecture.

6.2.11 ATM-TGS Aviation Client

In order to demonstrate the functionalities of the DMS, a client was developed to test the different operations at the DMS and to demonstrate its usefulness and efficiency.

6.2.11.1 Functionalities

The client developed in OWS-9 had the capacity to communicate with the following entities:

- Web Feature Service.
- Event Services.
- Data Management Service.

The client was divided in 2 parts; the client display application and the client DMS module (CDMS).

The application part was in charge of the direct interaction with the human client. It encompassed a moving map and an interface for information queries. All display was in the client application

The CDMS part is responsible for seamless interaction of the client application with the DMS. Its role was to:

- Negotiate a client session with the DMS.
- Create a reliable messaging session with the DMS.
- Compress requests (to the DMS) and decompress responses (from the DMS).
- Receive notifications from the DMS and forward them to the client application.

6.2.11.2 Deployment

The client application was been developed with Qt/Marble running on Linux/Ubuntu.

The CDMS was developed in Java in order to include the libraries used at the DMS (Sandesha and Fast Infoset).

6.2.11.3 Interfaces

In order to receive notifications from the DMS, the CDMS used an endpoint where the Notify operation was available.

6.2.12 ATM-TGS Dispatch client

In order to demonstrate the synchronization functionality of the DMS, a Dispatch client was developed. It communicated with the DMS to retrieve consumer reference of a given client and to subscribe to clients.

6.2.12.1 Functionalities

The dispatch client developed in OWS-9 had the following functionalities:

- Retrieve the ConsumerReference of a client at the DMS in order to make subscription on behalf of that client.
- Subscribe to a client at the DMS.
- Be notified of any information arriving at the DMS intended for a client the dispatch subscribed to.
- Display this information.

The Dispatch Client encompassed a simple 1 endpoint – 1 operation web service in order to be notified by the DMS in the frame of the synchronisation functionality.

6.2.12.2 Deployment

The dispatch client was developed with Qt running on Linux/Ubuntu

The Dispatch web service CDMS was developed in Java.

6.2.13 Luciad Aviation Client

6.2.13.1 Introduction

To support the integration & testing of the various service components developed within OWS-9 Aviation, Luciad contributed an Aviation Client component. This component was based on Luciad's COTS product LuciadLightspeed, which offers numerous capabilities and benefits that are of direct use in the client: full support for AIXM (including its temporality and metadata models), connectors to OGC services such as WMS, WCS and WFS-T, flight simulation, 2D & 3D visualization. On top of this product, a thin layer was developed to support custom functionality needed in OWS-9 and to provide the user with a dedicated user interface focusing on the OWS-9 tasks. Figure 3 shows a screenshot of the Luciad client's user interface.

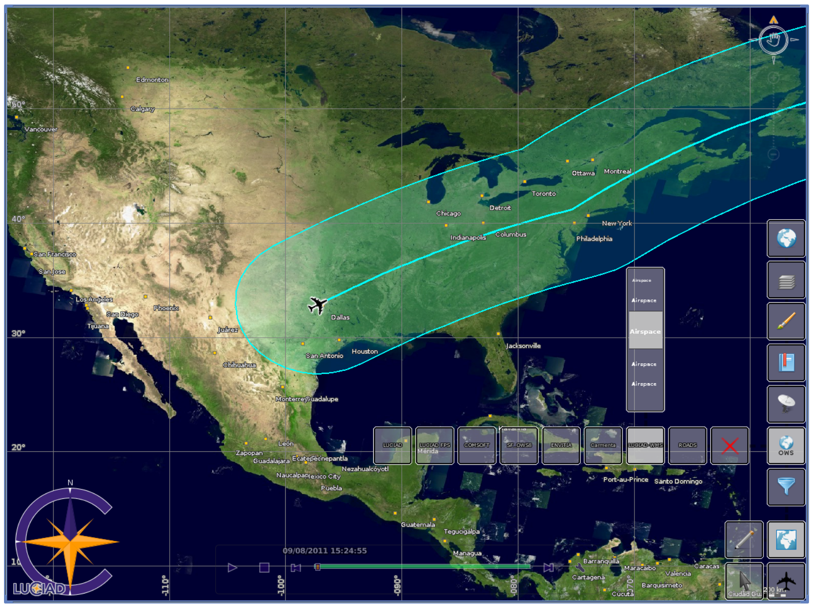


Figure 3 – The Luciad Aviation Client in action.

6.2.13.2 Functional overview

The Luciad Aviation client provided the following functionality to support OWS-9 Aviation:

- Map-centric display with intuitive user interface giving access to various actions, including:
    - Map controllers to manipulate the map (zoom, pan, …).
    - Map layer control with access to predefined background data layers.

  - Gazetteer and bookmarks to quickly search for & navigate to places / airports.
  - OGC web service connectors.
  - Flight preview / simulation.
  - Visualization in 2D & 3D.
  - Visual representation & browsing of feature properties inside a balloon.
- Client interface to interact with the Geometry Processing profile of the WPS service components. Users could select two AIXM features (e.g. an airspace and a route segment) and determine their topological relationship or calculate the intersection points.
- Client interface to interact with the Cross Community Interoperability (CCI) WPS mediation service developed in the OWS-9 CCI thread. Users could enter a pilot term and search for AIXM features that comply with that term. The CCI WPS mediation service sent back a list of AIXM features and WFS services that provided them, which were then resolved and visualized on the map.
- Client interface to a WFS 1.0, 1.1 & 2.0 service. Users could query and retrieve feature data from a WFS. The application also allowed defining and using filters based on OGC Filter Encoding 1.0, 1.1 and 2.0 (such as a DWithin spatial filter around a flight trajectory illustrated in Figure 3.)
- Client interface to a Web Coverage Service (WCS) 1.0, 1.1 & 2.0. Users could query and retrieve coverage data in the GeoTIFF and JPEG 2000 (including GMLJP2) format.
- Client interface to a WMS 1.1.1 & 1.3.0 service. Users could query and retrieve layers from a WMS.
- Client interface to the Event Service. Users could subscribe / unsubscribe to events (also using OGC Filters) and consume incoming events (e.g. visualization of an incoming Digital NOTAM event).
- Rendering engine with support for SLD / SE 1.0/1.1 to render vector feature and raster data. This engine integrated extensions to support ICAO Annex 4 rendering guidelines for aeronautical data.
- Support to represent & browse ISO 19115-compatible metadata and to encode / decode to / from an ISO 19139-compatible data source. The use of metadata extensions / profiles was supported.
- Wide range of data format support, including:
  - AIXM 5.0 & 5.1 for aeronautical data (including extensions).
  - WXXM 1.1 for weather data (including extensions).

  - Additional aeronautical and weather data formats like AIXM 3.3/4.0/4.5, ARINC 424, DAFIF, ASTERIX, ASDI and GRIB.
  - Raster format support (GeoTIFF, TIFF, JPEG, JPEG 2000, GMLJP2, JPIP, PNG, GIF, ECW, MrSID, CADRG/ADRG/USRP, DTED, USGS DEM, Oracle GeoRaster …) for satellite imagery and elevation background data.
  - Vector format support (ESRI Shape, MapInfo MIF/MAP, GML 2/3.1.1/3.2.1, SVG, DGN, DWG, Oracle/Informix/MSSQL databases …) for vector-based background data.
  - Other formats: OBJ, OpenFlight, OGC KML 2.2 and GeoPDF.

6.2.14 Deployment characteristics

All development was done in Java, using the Java Development Kit (JDK) 1.6. The client application was developed on top of Luciad's COTS product LuciadLightspeed. The software runs on any operating system for which a Java Virtual Machine 1.6 or higher exists. For the 3D visualization, a graphics card with support for OpenGL 1.2 or higher is required.

6.2.14.1 Challenges

A reoccurring challenge when working on the client is obtaining proper interoperability with all services and overcoming implementation differences. Adherence to the specification of data formats and OGC services is only one step, as there is typically always room for implementation differences that impede interoperability in practice. We therefore want to emphasize the usefulness of official compliancy tests such as the ones developed by OGC (e.g. for the WMS, WFS and WCS), as they can help to increase overall interoperability. It might be an idea for the future to develop official compliancy tests to verify the guidelines and best practices have been developed the past years with respect to using OGC services in the Aviation domain.

6.2.14.2 Accomplishments

The key accomplishments for Luciad's Aviation client component in OWS-9 included:

- Demonstration of and interaction with most of the service components provided within the Aviation thread and with the Semantic Mediation WPS provided within the CCI thread.
- Contribution of a rich client equipped with lots of capabilities and features that help to demonstrate the OWS-9 Aviation scenario and to perform testing and integration with a wide variety of service components.
- Collaboration with OWS-9 service component providers on the developed functionality and provision of feedback from a client's perspective.

7 Data Provision via Web Feature Service and Event Service

7.1 Overview

This section describes the architecture for data provision used in OWS-9. In particular, the dynamic provision and processing of AIXM 5.1 data (Feature updates via Digital NOTAM events) based on the service architecture of previous test beds using WFS-T and Event Service is described. Besides AIXM the support for providing weather data (encoded as WXXM 1.1.3) is illustrated. This includes the description of the used event encodings as well as the involved components. The advanced filtering functionality is described per topic in detail. Finally, the issues and drawbacks which have been observed during OWS-9 as well as future work items are presented.

7.2 General Service Architecture

The data provision service architecture consists of two layers. Data subsccription matching mechanisms are served by Event Service (ES) implementations. Data storage and provision functionality is supplied via Web Feature Service (WFS, see section 7.2.1).

The data provision layers are accessed from the Data Management Service (DMS) and Aviation Clients via common request/response communication (WFS, WCS) and subscriptions for certain data of interest (ES). Filters (on attributes, spatial extent, etc.) are defined using the Filter Encoding Specification (FES) 2.0 for both WFS and ES.

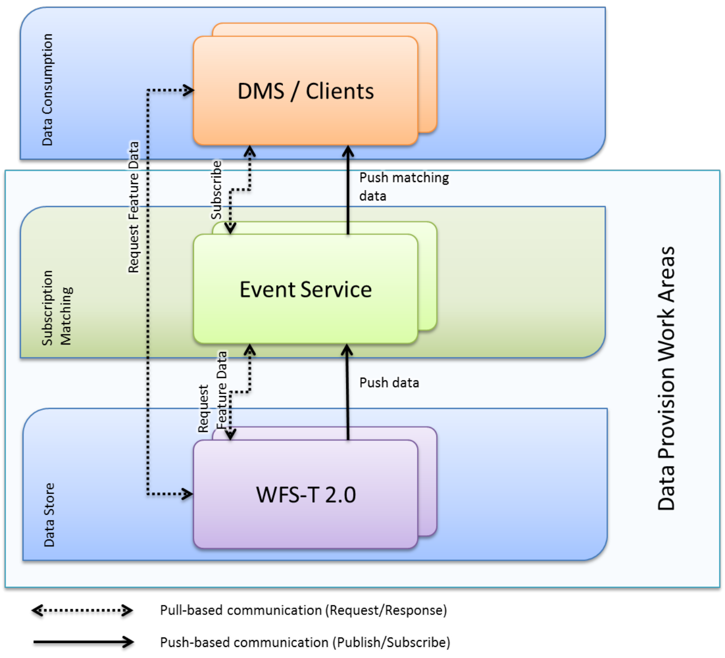


Figure 15 - Data Provision Service Architecture.

7.2.1 Web Feature Service

WFS implementations are based on the OpenGIS Web Feature Service 2.0 Interface Standard (OGC 09-025r1). Both service instances (provided by Snowflake Software and COMSOFT) provide a transactional interface for insertion/update of new/existing AIXM 5.1 features.

7.2.2 Event Service

The Event Service (ES) implementations are based on the OpenGIS Sensor Event Service Interface Specification Discussion Paper (OGC 08-133). Since the publication of this paper additional functionality has been developed within OWS test beds. Hence the term “Event Service” has been established as its capabilities are no longer limited to sensor measurements. ES implementations are based on the OASIS Web Services Notification standards family (WS-N; WS-BaseNotification, WS-BrokeredNotification, WS-Topics 1.3 OASIS Standards) which provide mechanisms to equip a service with publish/subscribe concepts.

Requests to an ES are posted to interfaces with a SOAP 1.2 binding. These interfaces are defined by WSDL 1.1 descriptions.

7.3 Communication Patterns

This section illustrates the underlying data models as well as common service chains used for information dissemination with the data provision architecture.

7.3.1 Data Model

The communication between the involved components makes use of two concepts. Temporal changes to the AIXM features contained in the WFS data stores are encoded and published using the dedicated Event extension of AIXM 5.1 as specified by the Digital NOTAM Event Specification 1.0. For communications between WFS and ES instances the AIXMBasicMessage of this extension is used to carry the DNOTAMs. An exemplary message representing a Taxiway closure is provided in Listing 1.

Listing 1 – Taxiway Closure DNOTAM Event.

```
<message:AIXMBasicMessage  gml:id="FNS-Message"
xmlns:message="http://www.aixm.aero/schema/5.1/message"
xmlns:aixm="http://www.aixm.aero/schema/5.1" xmlns:gco="http://www.isotc211.org/2005/gco"
xmlns:gmd="http://www.isotc211.org/2005/gmd" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:event="http://www.aixm.aero/schema/5.1/event"
xmlns:gml="http://www.opengis.net/gml/3.2" >
    <message:hasMember>
        <event:Event gml:id="uuid.ff3ab666-d8d0-428c-8304-5922ee4636b5">
            <gml:description>Closed taxiway due to maintenance</gml:description>
            <gml:identifier codeSpace="urn:uuid:">aa3ab666-d8d0-428c-8304-
5922ee4636b5</gml:identifier>
            <event:timeSlice>
                <event:EventTimeSlice gml:id="NOTAM_OWS9_17">
                    <gml:validTime>
                        <gml:TimePeriod>
                            <gml:beginPosition>2012-11-22T22:00:00Z</gml:beginPosition>
                            <gml:endPosition>2012-11-23T04:00:00Z</gml:endPosition>
                        </gml:TimePeriod>
                    </gml:validTime>
                    <aixm:interpretation>BASELINE</aixm:interpretation>
                    <aixm:sequenceNumber>1</aixm:sequenceNumber>
                    <aixm:featureLifetime>
                        <gml:TimePeriod>
                            <gml:beginPosition>2012-11-22T22:00:00Z</gml:beginPosition>
                            <gml:endPosition>2012-11-23T04:00:00Z</gml:endPosition>
                        </gml:TimePeriod>
                    </aixm:featureLifetime>
                    <event:name>NOTAM_OWS9_17</event:name>
                    <event:encoding>DIGITAL</event:encoding>
                    <event:scenario>TWY.CLS</event:scenario>
                    <event:textNOTAM>
                        <event:NOTAM>
                            <event:series>A</event:series>
                            <event:number>0005</event:number>
                            <event:year>2012</event:year>
                            <event:type>N</event:type>
                            <event:issued>2012-11-21T11:03:00</event:issued>
                            <event:affectedFIR>MKE</event:affectedFIR>
                            <event:selectionCode>QMRLC</event:selectionCode>
                            <event:traffic>IV</event:traffic>
                            <event:purpose>NBO</event:purpose>
                            <event:scope>A</event:scope>
                            <event:coordinates>42N87W</event:coordinates>
                            <event:radius>999</event:radius>
                            <event:location>MKE</event:location>
                            <event:effectiveStart>1211222200</event:effectiveStart>
                            <event:effectiveEnd>1211230400</event:effectiveEnd>
                            <event:text>TWY CLOSED.</event:text>
                            <event:lowerLimit>000</event:lowerLimit>
                            <event:upperLimit>999</event:upperLimit>
                            <event:publisherNOF xlink:href="#urn.uuid.c225ae5c-540f-4a48-
8867-809b393b2407" xlink:title="CIVIL AVIATION AUTHORITY"/>
                        </event:NOTAM>
                    </event:textNOTAM>
                </event:EventTimeSlice>
            </event:timeSlice>
        </event:Event>
    </message:hasMember>
    <message:hasMember>
        <aixm:TaxiwayElement gml:id="gmlID4640">
            <gml:description>Closed taxiway due to maintenance</gml:description>
            <gml:identifier codeSpace="urn:uuid:">14BDAE02-1D6A-41F5-98AC-
DE94D3253A53</gml:identifier>
            <aixm:timeSlice>
                <aixm:TaxiwayElementTimeSlice>
                    <gml:validTime>
                        <gml:TimePeriod>
                            <gml:beginPosition>2012-11-10T22:00:00Z</gml:beginPosition>
                            <gml:endPosition>2012-11-23T04:00:00Z</gml:endPosition>
                        </gml:TimePeriod>
                    </gml:validTime>
                    <aixm:interpretation>TEMPDELTA</aixm:interpretation>
                    <aixm:sequenceNumber>1</aixm:sequenceNumber>
                    <aixm:availability>
                        <aixm:ManoeuvringAreaAvailability>
                            <aixm:operationalStatus>CLOSED</aixm:operationalStatus>
                        </aixm:ManoeuvringAreaAvailability>
                    </aixm:availability>
                </aixm:TaxiwayElementTimeSlice>
            </aixm:timeSlice>
        </aixm:TaxiwayElement>
    </message:hasMember>
</message:AIXMBasicMessage>
```

7.3.2 Service Chaining

Communication of events is based on the provision of Digital NOTAMs (DNOTAM) through WFS Transactional (WFS-T) instances. If a Feature update/insertion is populated to the WFS-T it automatically generates a DNOTAM and pushes it to the available ES instances. The WFS servers also act as a static repository for retrieval of AIXM features and snapshots of these.

The following sequence diagram (Figure 16) illustrates a common situation where a DNOTAM is automatically created within a WFS-T, pushed to the ES instances and after a positive subscription matching forwarded to the DMS.

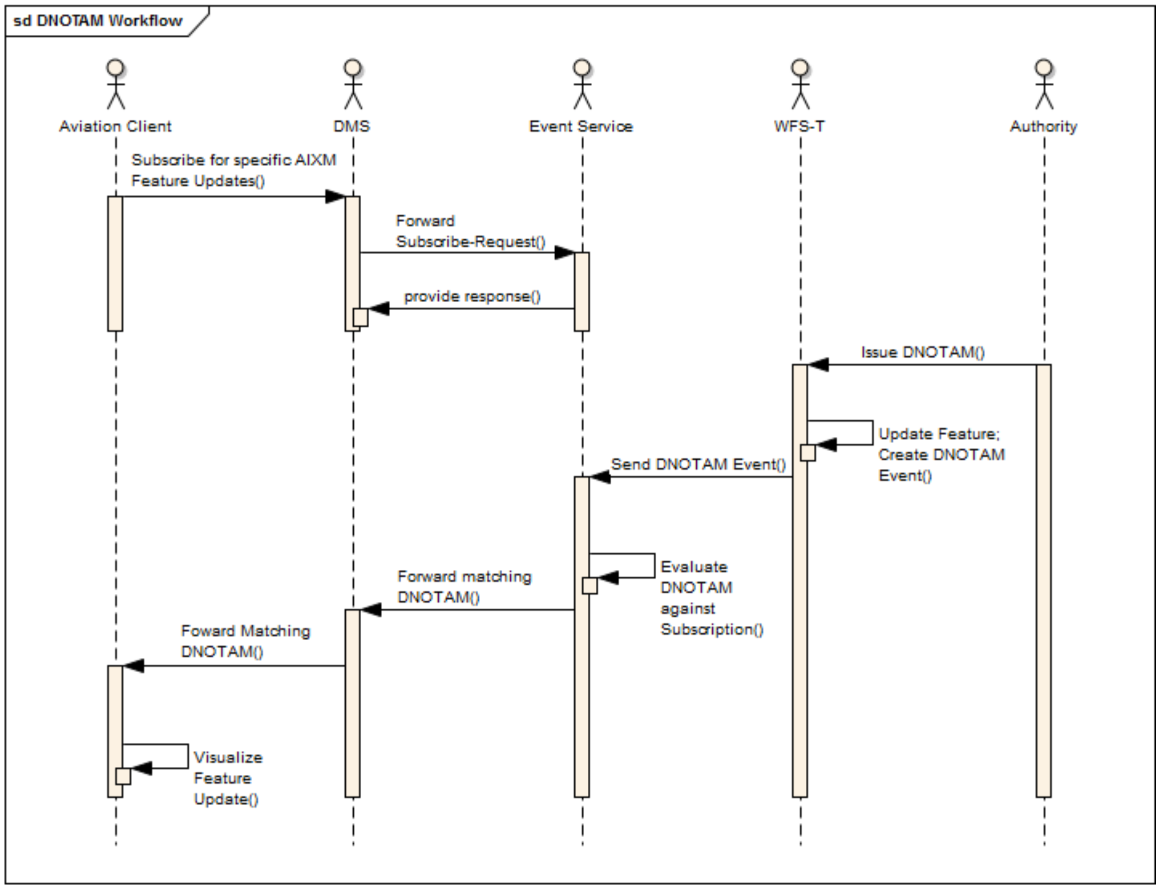

sd DNOTAM Workflow
Aviation Client
DMS
Event Service
WFS-T
Authority
Subscribe for specific AIXM Feature Updates()
Forward Subscribe-Request()
provide response()
Issue DNOTAM()
Update Feature; Create DNOTAM Event()
Send DNOTAM Event()
Evaluate DNOTAM against Subscription()
Forward matching DNOTAM()
Foward Matching DNOTAM()
Visualize Feature Update()

Figure 16 - DNOTAM Workflow.

7.4 Advanced Filtering Functionality

One major work area for the data provision team was the design and implementation of advanced filtering functionality. The goal was to enhance the established service architecture with additional methods which make the life of both the clients and services easier. The following sections provide detailed information on each newly developed filtering feature.

7.4.1 Stored Queries / Filters

Within former test beds the complexity of filters posted to ES and WFS increased constantly. As clients often use the same subscription among different flights a new method was required to enable the straight-forward reusability of filters.

7.4.1.1 Problem Statement

Most ES subscriptions (e.g. a flight route buffer) and WFS queries (e.g. Airspace by designator) are frequently used by client applications. The work on this feature covered the testing and documentation of how WFS stored queries can be used or enhanced to facilitate the execution of common data retrieval tasks at WFS 2.0. Similar to WFS, the additional goal is to develop, test, and document how Event Service stored filters can be used to facilitate the creation of subscriptions for common information needs. Besides the technical solution, another goal was to develop a set of filters that are common in Aviation and make them available via WFS and Event Service.

7.4.1.2 Concept and Methods

The WFS specification defines a set of operations to enable stored queries:

- ListStoredQueries – provide a list of queries with unique identifiers
- DescribeStoredQueries – provide description(s) for all available / a specific stored querie(s)
- CreateStoredQuery – dynamically create a new stored query at the service
- DeleteStoredQuery – remove the stored query from the service.

A client could discover and execute available stored queries and adjust their behavior through a set of parameters (e.g. a designator, a flight route, etc.). A request looks like the following.

Listing 2 – Stored Query Execution.

```
http://demo.snowflakesoftware.com:8081/OWS-
9_AIXM51/OWS9_AIXM51?service=WFS&version=2.0.0&request=GetFeature&
STOREDQUERY_ID=urnx:wfs:StoredQueryId:Snowflake:AirspaceDWithinAltitude&flightpath=32.5%20-
97%2033%20-96%2032.4%20-97.1%2033.1%20-96.1&
distance=100&upperlimit=99999&upperlimit_uom=FT&lowerlimit=00000&lowerlimit_uom=FT
```

To enable stored filters for the ES, these operations were adopted and defined within an ES stored filter extension schema. This schema defines the following interface methods.

- ListStoredFilters – similar to ListStoredQueries
- DescribeStoredFilter – similar to DescribeStoredQueries (with a query identifier)

Consequently, a client was able to discover and retrieve information on stored filters. A stored filter subscription was achieved by providing the unique identifier as well as the parameter values, similar to WFS stored query execution (see **Error! Reference source not found.**).

Listing 3 – Stored Filter Subscription.

```
<wsnt:Subscribe>
    <wsnt:ConsumerReference>
        <wsa:Address>${consumer}</wsa:Address>
    </wsnt:ConsumerReference>
    <wsnt:Filter>
        <wsnt:MessageContent Dialect="http://www.opengis.net/es-sf/0.0">
            <essf:StoredFilterSubscription
                id="urn:ogc:def:filter:OGC-ES::SubscribeForFlightRouteBuffer"
                xmlns:essf="http://www.opengis.net/es-sf/0.0">
              <essf:ParameterValue name="flightroute">
                <gml:posList srsName="urn:ogc:def:crs:EPSG::4326">
                    45.256 -110.45 46.46 -109.48 43.84 -109.86
                </gml:posList>
              </essf:ParameterValue>
              <essf:ParameterValue name="distance">
                <fes:Distance uom="[nmi_i]"
                    xmlns:fes="http://www.opengis.net/fes/2.0" >
                    500
                </fes:Distance>
              </essf:ParameterValue>
            </essf:StoredFilterSubscription>
        </wsnt:MessageContent>
    </wsnt:Filter>
</wsnt:Subscribe>
```

As the Event Service uses SOAP as the request wrapper, the following SOAP actions were defined to be used with the corresponding interface methods:

- **ListStoredFilters -** http://www.opengis.net/es-sf/ListStoredFiltersRequest
- **DescribeStoredFilter -** http://www.opengis.net/es-sf/DescribeStoredFilterRequest.

7.4.1.3 Implementation

All WFS and ES implementations provide the functionality to use Stored Queries/Filters. The ES instances for OWS-9 used the developed XML schema available at http://test.schemas.opengis.net/ows-9/aviation/es/storedFilters.xsd. It is documented in Annex B of this document for convenience.

Within this OWS, the following Stored Queries and Filters were developed.

Type	**ID**	**Parameters**
ES	urn:ogc:def:filter:OGC-ES::SubscribeForFeatureTypeWithinFlightRouteBuffer	• flightroute: a flight route encoded as the contents of a gml:posList element with 'urn:ogc:def:crs:OGC:1.3:CRS84' as the spatial reference system

		distance: the buffer distance as nautical miles typename: the AIXM feature type (e.g. aixm:Runway)
ES	urn:ogc:def:filter:OGC-ES::SubscribeForFeatureType	typename: the AIXM feature type (e.g. aixm:Runway)
ES	urn:ogc:def:filter:OGC-ES::SubscribeForFeatureType AndExcludeMetadata	typename: the AIXM feature type (e.g. aixm:Runway) This stored filter removes metadata associated with the feature and any time slice of it
WFS	GetAirportByName	name: a portion of the AirportHeliport's name attribute The query returns all BASELINE TimeSlices for airports which map the provided portion. It was designed to ease the creation of ePIBs.

7.4.2 Simple Altitude Queries

The aim of this work area was to develop, test and document an approach for enabling more simplified filtering of aeronautical feature data based upon filter expressions that take the vertical extent of the feature into account

7.4.2.1 Problem Statement

Dealing with altitude information within AIXM is a rather complex task due to the underlying model (separated altitude information from 2D geometries). Additional intricacy is added when considering that:

- Aeronautical features may have multiple component objects with different altitudes
- Altitudes may be expressed with special values (GND, UNL, FLOOR, CEILING).

Therefore, a robust documentation of filtering AIXM features based on altitude information is provided in the following sections.

7.4.2.2 Concept and Methods

Within OWS-9, an agreement was made to demonstrate simple altitude queries by retrieving Airspace that intersected a given flight route represented as a set of aixm:RouteSegments. An aixm:RouteSegment provided all needed parameters (e.g. horizontal extent, lowerLimit, upperLimit) to make a good candidate for demonstration. **Error! Reference source not found.** Note that the custom XPath-function "wfs-aixm:extentOf" is the same as defined for the "Spatial Filtering of non-spatial features" work area (see Section 7.4.3.2).

Listing 4 - Exemplary Altitude Query.

```
<wfs:GetFeature xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:fes="http://www.opengis.net/fes/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:wfs="http://www.opengis.net/wfs/2.0" service="WFS" version="2.0.0">
    <wfs:Query typeNames="aixm:Airspace">
        <fes:Filter>
            <fes:DWithin>
                <fes:ValueReference>wfs-aixm:extentOf(.)</fes:ValueReference>
                <wfs:SimpleFeatureCollection>
                  <wfs:member>
                    <aixm:RouteSegmentTimeSlice gml:id="...">
                      <gml:validTime/>
                     <aixm:intepretation>BASELINE</aixm:interpretation>
                      <aixm:upperLimit>...</aixm:upperLimit>
                      <aixm:upperLimitReference>...</aixm:upperLimitReference>
                      ... lower limit properties...
                      <aixm:curveExtent>
                        ... GML Curve ...
                      </aixm:curveExtent>
                    </aixm:RouteSegmentTimeSlice>
                  </wfs:member>
                  <wfs:member>
                      ...another RouteSegmentTimeSlice that makes up the flight path...
                  </wfs:member>
                </wfs:SimpleFeatureCollection>
                <fes:Distance uom="M">25000</fes:Distance>
            </fes:DWithin>
        </fes:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

For the use case "Retrieve all Airspaces that intersect a given Route" it was necessary to include a collection of RouteSegments in the request. This is because there is no native GML encoding for 2.5D yet supporting the AIXM 5.1 requirements, i.e. having latitude, longitude, and vertical range of values (lowerLimit, upperLimit, minimumLimit, maximumLimit) with a specific UOM and reference (e.g. mean sea level or WGS84 ellipsoid).

7.4.2.3 Implementation

In a spatial database not natively supporting altitude, two problems had to be solved:

1. Correctly applying the spatial constraint on the altitude part

2. Converting complex features such as composite Airspaces in collections of simple features ("boxes" in that case)

7.4.2.3.1 Altitude constraint evaluation

Altitude information may be expressed in relation to different references, including the surface of the earth (terrain) and air pressure (flight level). In the most complex scenario possible, a precise calculation would require an elevation model of the earth and information about the air pressure at a specific location and specific time (weather data). This leads to a true 3D calculation with probabilistic measurements.

Obviously, for a first implementation in the test bed, a simpler implementation had to be chosen. If differences in references are ignored and standard pressure can be assumed at all times and locations, both altitudes can be converted into a common unit of measurement (e.g. meters). The altitude comparison is then reduced to the comparison of two number ranges.

7.4.2.3.2 Decomposing airspaces

Airspaces in AIXM may be defined in different ways. Simple airspaces consist of a single GML geometry and altitude information. As GML allows geometry composition by XLinks, airspaces can also be defined by including references to other features such as geographical borders (GeoBorder) in their GML geometry part. The next level of complexity is the composition of airspaces by other airspaces. There, the composition operation is another variable - airspaces can be formed by the union, intersection or subtraction of other airspaces, optionally including or excluding altitude. As all references to other features are time-independent, changes of referenced features during the time of evaluation have also to be taken into account. Thus, the whole process of decomposing airspaces to simple “boxes” is a complex spatio-temporal algorithm. A detailed guidance on how to compute the geometry of a complex Airspace is provided in Annex C of this document.

7.4.3 Spatial Filtering of Non-spatial Features

The intention was to simplify spatial filtering of AIXM features that do not contain the properties that define their spatial extent themselves. In the AIXM model, often other features that reference the non-spatial feature provide information about the extent. This problem has been investigated in previous test beds. A solution simple to use for clients and at the same time feasible for the service to implement as well as being compatible to existing standards is needed.

7.4.3.1 Problem Statement

To enable spatial filtering of AIXM feature that do not contain a spatial extent, rules for computing the geometry of such features based upon their constituent parts or those of other AIXM types that have a relationship to this feature need to be developed. Within former test beds the so-called “Reverse Associations” had been introduced which provide reverse links to associated features (e.g. reverse associations on a Runway referring to its RunwayElements). But evaluation of that approach revealed that the management of such reverse associations in the AIXM Temporality model is very complex, and the increase in complexity outweighs the benefits. An alternative approach needed to be developed. One proposed approach is documented in the following sections.

7.4.3.2 Concept and Methods

The proposed approach makes use of a custom XPath function. An example is provided in Listing 5. A WFS implementing this XPath function ensures that it is capable of internally resolving the geometry of a given TimeSlice regardless of whether the TimeSlice defines a geometry itself or not.

Listing 5 – Custom XPath Function “extentOf”.

```
<fes:Intersects>
  <fes:ValueReference>wfs-aixm:extentOf(.)</fes:ValueReference>
  ... GML geometry or feature identifier ...
</fes:Intersects>
```

Most AIXM features define multiple representations of a geometry (e.g. gml:boundedBy and aixm:ElevatedPoint). It is planned for future work to integrate an additional parameter into the XPath function to define the use case / interpretation. This enables a client to retrieve different representations for specific tasks.

7.4.4 AIXM Features as Geometry Operands

In previous test beds, it was the responsibility of clients to provide (GML) geometries as filter parameters. To facilitate the usage of WFS queries and ES subscriptions these services should be capable of processing AIXM features as geometry inputs for their internal filtering.

This feature is closely related to the WPS “Geometry Retrieval” profile which is designed to resolve/compute the GML geometry of a given AIXM feature. See section 8 for details.

7.4.4.1 Problem Statement

There are two types of AIXM features: ones that define an explicit geometry (e.g. GeoBorder) and those with features obtaining their spatial extent from related features (e.g. Runway, Taxiway). As it is the purpose of these features to shift the responsibility to resolve the actual geometry of a feature from the clients to the servers a robust and well-documented approach is documented in the following.

7.4.4.2 Concept and Methods

One prerequisite was the technical feasibility of including AIXM features into FES 2.0 markup. The BinarySpatialOpType allows xs:any with strict validation as the second input, meaning that a schema definition for the used element must be present.

In aviation-specific applications, limiting the use of bandwidth is always an important goal. Therefore, this functionality defined an extension schema (referring to AIXM 5.1) which allows the provision of a feature identifier (see Listing 6). This can be considered as referring to a uniquely identified feature within a WFS server instance[1]. The implementation is linked to Aviation Processing work area (see section 8).

Listing 6 - Feature by Identifier Reference.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature service="WFS" version="2.0.0"
    outputFormat="application/gml+xml; version=3.2" ...>
    <wfs:Query typeNames="avwx:SIGMET" handle="Q01">
        <fes:Filter>
            <fes:Intersects>
                <fes:ValueReference>wfs-aixm:extentOf(.)</fes:ValueReference>
                <aixm-ext:FeatureIdentifier>
                     <aixm-ext:identifier codeSpace="urn:uuid:">
                         4fd9f4be-8c65-43f6-b083-3ced9a4b2a7f
                     </aixm-ext:identifier>
                <aixm-ext:FeatureIdentifier>
            </fes:Intersects>
        </fes:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

7.4.4.3 Implementation

Within OWS-9 it was agreed that the WFS instances would implement the functionality inside their own services since relying on an external service is not preferable for data stores which are intended to be stable and reliable.

The Event Service instances make use of the “Geometry Retrieval” WPS Profile. At the time of subscription at the service, a WPS Execute request with the AIXM Feature placed inline or referenced (see Listing 7**Error! Reference source not found.**). As the result is a GML 3.2 geometry, the ES instances were then able to treat it as a common geometry.

Listing 7 – Exemplary Geometry Retrieval WPS Request.

```
<wps:Execute service="WPS"
             version="1.0.0"
             xmlns:wps="http://www.opengis.net/wps/1.0.0"
             xmlns:ows="http://www.opengis.net/ows/1.1"
             xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:aixm-ext="http://www.opengis.net/ows9/aviation/aixm/extension"
             xsi:schemaLocation="http://www.aixm.aero/schema/5.1
http://test.schemas.opengis.net/ows-9/aviation/aixm/5.1/AIXM_Features.xsd
http://www.opengis.net/ows9/aviation/aixm/extension http://test.schemas.opengis.net/ows-
9/aviation/aixm/aixm-extension.xsd">
  <ows:Identifier>ResolveAIXMFeatureGeometry</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>Feature</ows:Identifier>
      <wps:Data>
        <wps:ComplexData>
          <aixm-ext:FeatureIdentifier>
            <aixm-ext:identifier codeSpace="urn:uuid:">4fd9f4be-8c65-43f6-b083-
3ced9a4b2a7f</aixm-ext:identifier>
```

[1] The built-in join functionality of the WFS specification is not as compact, as it always returns both join partners.

```
          </aixm-ext:FeatureIdentifier>
        </wps:ComplexData>
      </wps:Data>
    </wps:Input>
  </wps:DataInnputs>
</wps:Execute>
```

7.4.4.4 Retrieval of WXXM Data

Within OWS-9, this feature was only implemented to retrieve AIXM-encoded data. However, another interesting application is the filtering of WXXM-formatted data (e.g. provided through a WFS or WCS). This section provides some insights about possible benefits and drawbacks when realizing such functionality.

In situations where the AIXM feature that is used as geometry operand is referenced instead of given inline and if that feature is not known to the service (because it is not stored there), the AIXM feature as geometry operand functionality does not work – unless, of course, the WXXM-WFS can dynamically discover said feature.

Additionally, as this feature is currently implemented using WFS spatial joins, it could add an additional effort to provided support for both encodings in spatial joins. However, spatial joins allow more compact results as no tuples are returned which would ease the processing of a result.

The following table summarizes additional pros and cons of such an approach.

Pros	**Cons**
- Easier execution of WFS GetFeature filters to retrieve WXXM data that satisfy a spatial constraint, especially if that spatial constraint is based upon the spatial extent of an AIXM feature (e.g. a route, an airspace, an ARTCC).	- WXXM - providing WFS today often do not have knowledge/support for AIXM features – adding the AIXM feature as geometry operand functionality as an implementation requirement thus creates additional implementation burden. This may be mitigated by the design of a cascading/adapting WFS which has access to both a WFS with WXXM data and a WFS with AIXM data (or has that already) and which realizes the AIXM feature as geometry operand functionality.

7.4.5 Event Service Update Intervals

The goal was to develop, test and document how a client could receive notifications from the Event Service in specific update intervals as defined for the according subscription.

7.4.5.1 Problem Statement

As previously stated, the economic use of bandwidth plays an important role in Aviation-specific applications. Therefore, a client should be able to define a behavior on how and how often to retrieve notification through an ES instance. This particularly applies to situations where an immediate update is not always required. If a client is able to define intervals in which it would like to retrieve new data it would not be overextended with massive amounts of data and would be able to focus on the most relevant information.

7.4.5.2 Concept and Methods

Update Intervals are a specialized form of Notification dissemination to an Event Service client. They are used to define a time window in which an ES implementation collects all Notifications which match a subscription. Depending on specified parameters it was decided that if only the latest event within this time window was forwarded to the client or whether a data format batching notifications (e.g. multiple TimeSlices for one feature) would be applied. By defining these intervals, a client could benefit from reduced bandwidth load. Update Intervals could be applied to any subscription made at an Event Service. Client software should make the decision on what subscriptions it would like to receive information at specific rates.

7.4.5.2.1 Requirements

An Update Interval implementation must fulfill the following requirements.

- When subscribing, the client software should be able to define if it is interested in receiving all messages ("batching") or only the latest event of the specified time window.
    - The "latest event" option has to be applied with great care. In the case of multiple new TimeSlices for one feature (e.g. a Runway operational status) in the time window only the latest TimeSlice would be delivered to the client. If situations occur where another property (besides the operational status) of that Runway has changed previously the "latest event" option will omit the dissemination of that TimeSlice and only provide the information on the operational status. Client software should always be aware of this fact and therefore define subscription filters in a way that such situations are not likely to occur.
    - Identifying duplicate events should be applied to the batching option. Examples of duplicate events include single service duplication (error), multiple service duplication (e.g., AIM issues TFR for a presidential event and then a conflation service also publishes the same TFR – both services are assumed to be consumed by the client), or multi-agency duplication (e.g., both the military and the FAA maintain a set of NOTAMs and both can be considered authoritative (governance issue). If a client subscribes to both feeds it will receive multiple events). It is the decision of the ES

implementation on how to detect duplicate events. One approach could be the comparison of gml:identifiers but in some cases more complex algorithms need to be applied.

- The capability of a service to detect different features (e.g. two different Airports) within a time window should be specified in the service capabilities. For OWS-9 an implementation will not be available and will be proposed for future work.

- If no update has been received within the interval, an Update Interval implementation should provide a system message stating that no event has been received.
- An Update Interval cannot be changed after the subscription has been created. If a client wants to change it, it should cancel the existing subscription and create a new one with the adjusted Update Interval.

7.4.5.2.2 Use Case

To clarify and illustrate the benefits of update intervals for ES subscriptions a brief scenario example is presented within this section. The illustrated example highlights the need for a well-defined interface that builds upon existing interface methods. Therefore, changing the overall interface of the ES is not necessary. The implementation of Update Intervals within OWS-9 is based on Subscription Policies as defined by the WS Notifications standards family.

The exemplary scenario builds on the experience of former OWS test beds. A pilot might have a high interested in events which are "spatially near" the aircraft, hence intersect a buffer of the flight route or any RouteSegment. In contrast, he might also have information about events which are a bit more spatially distant. Though, not every single DNOTAM might be of interest, an ES can define an interval at which the information should be disseminated via the Update Intervals functionality.

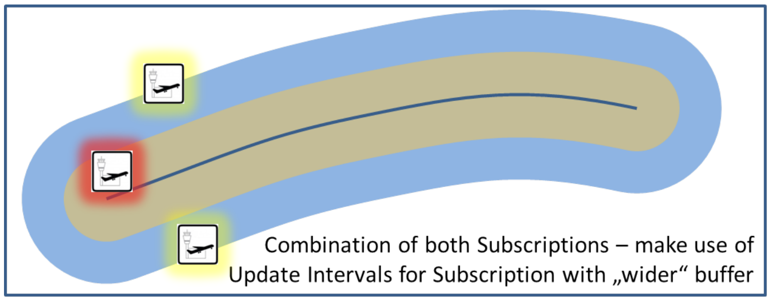

Combination of both Subscriptions – make use of
Update Intervals for Subscription with „wider" buffer

Figure 17 – Update Interval Subscription Illustration.

Technically, the client defines two separate Subscriptions. This allows the ES to distinguish between the dissemination methods. The first Subscription is a common one, defining no update interval policy. The second one uses the wider buffer, excludes the first buffer and defines an Update Interval. Both subscriptions create two different streams of data (see **Error! Reference source not found.**), one providing all information on events available (e.g. the airport with red shine), the other providing information on events at the specified interval (e.g. airports with yellow shine).

7.4.5.3 Implementation

Based on the requirements a schema has been developed which provides the encoding of update intervals as a subscription policy (see Annex B). A client can discover if an ES instance supports Update Intervals via its Capabilities. The schema defines the `SubscripionPolicyCapabilities` (see Listing 8**Error! Reference source not found.**) element which must be included into the `ows:OperationsMetadata/ows:ExtendedCapabilites`.

Listing 8 – Update Intervals Capabilities Portion.

```
...
<essp:SubscriptionPolicyCapabilities xmlns:essp="http://www.opengis.net/es-sp/0.0">
    <essp:SupportedPolicies>
        <essp:UpdateIntervalPolicy>
            <essp:BatchingSupported>true</essp:BatchingSupported>
            <essp:NonRelatedEventTreatmentSupported>false
            </essp:NonRelatedEventTreatmentSupported>
        </essp:UpdateIntervalPolicy>
    </essp:SupportedPolicies>
</essp:SubscriptionPolicyCapabilities>
...
```

The following example illustrates a Subscription with an Update Interval of 10 Minutes, “Batching” as the dissemination method and disabled non-related event treatment.

Listing 9 – Example Update Interval Subscription.

```
<wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
...
    <wsnt:Filter>
...
    </wsnt:Filter>
    <!-- UPDATE INTERVAL PART -->
    <wsnt:SubscriptionPolicy>
        <essp:EventServiceSubscriptionPolicy
            xmlns:essp="http://www.opengis.net/es-sp/0.0">
            <essp:UpdateInterval>
                <essp:IntervalDuration>PT10M</essp:IntervalDuration>
                <essp:DisseminationMethod>batching</essp:DisseminationMethod>
                <essp:NonRelatedEventTreatment>ignore
                </essp:NonRelatedEventTreatment>
            </essp:UpdateInterval>
        </essp:EventServiceSubscriptionPolicy>
    </wsnt:SubscriptionPolicy>
    <!-- END: UPDATE INTERVAL PART -->
</wsnt:Subscribe>
```

7.4.6 Selective Metadata Retrieval

The goal of this new functionality was to test how services (WFS-T & ES) can dynamically provide aeronautical and weather information to clients - with and without metadata as requested by the client (e.g. via an appropriate parameter in a WFS-T 2.0 GetFeature request or in an ES subscription).

7.4.6.1 Problem Statement

Metadata is encoded in properties on message, feature or time slice level. As ordinary properties, metadata elements are always returned inline together with the other contents of a feature. Some applications might not process metadata at all and, as it might be huge in size because of its verbosity, clients should be able to request it on demand. This is not covered by the WFS/FES standards.

7.4.6.2 Concept and Methods

The goal was to provide a method for the client to request feature data without metadata. This can be achieved by the introduction of a projection clause. In the WFS Temporality Extension, the PropertyExclusion projection allows the client to specify elements to exclude from a response by an XPath expression. If the XPath points to the metadata elements to remove, the requirement is fulfilled.

7.4.6.3 Implementation

For WFS servers the implementation of this feature is described in the WFS Temporality Extension ER,[OGC 12-146]. As the Event Service Subscribe request only uses the “Filter” element of FES no direct support for FES projection clauses was available. Therefore, an extension schema for the Subscribe method was developed to provide a similar semantic and syntax as used in the WFS implementation. Listing 10 illustrates an encoding of a subscription with a PropertyExclusion projection defined. The schema for the FilterWithProjectionClause element is attached to this document (see Annex B).

Listing 10 – Example Metadata Exclusion Subscription.

```
<wsnt:Subscribe>
    <wsnt:ConsumerReference>
        <wsa:Address>${consumer}</wsa:Address>
    </wsnt:ConsumerReference>
    <wsnt:Filter xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
        <wsnt:MessageContent Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116"
xmlns:xlink="http://www.w3.org/1999/xlink">
                    //aixm:Runway
        </wsnt:MessageContent>
        <wsnt:MessageContent Dialect="http://www.opengis.net/ses/filter/level2">
            <es-pc:FilterWithProjectionClause
xmlns:gmd="http://www.isotc211.org/2005/gmd" xmlns:fes-te="http://www.opengis.net/fes-
te/1.0" xmlns:es-pc="http://www.opengis.net/es-pc/0.0">
                <fes-te:PropertyExclusion>
                    <fes-te:propertyName>aixm:featureMetadata</fes-te:propertyName>
                </fes-te:PropertyExclusion>
                <fes-te:PropertyExclusion>
                    <fes-te:propertyName>aixm:timeSliceMetadata</fes-te:propertyName>
                </fes-te:PropertyExclusion>
            </es-pc:FilterWithProjectionClause>
        </wsnt:MessageContent>
    </wsnt:Filter>
</wsnt:Subscribe>
```

7.5 Service Discovery Based on Spatial Extent

Within the scope of service discovery an additional goal was the capability to identify services which provide data for a certain spatial extent (e.g. Europe, US). For WFS instances this is achieved by providing an ows:BoundingBox for every item in the FeatureTypeList.

As an ES theoretically can take any data as an input it naturally does not have a spatial limitation. To support discovery based on the spatial extent, the WFS's integrated an additional Capabilities element to point to the ES endpoint where DNOTAMs are sent to (see Listing 11**Error! Reference source not found.**). This way, the registry is able to map the ES instances to the WFS instances, hence providing discovery by spatial extent.

Listing 11 - Exemplary Event Service Reference for the IfGI ES.

```
<ows:ExtendedCapabilities>
  <es:EventServiceReference>
    <es:Publisher>
        <es:serviceName>SesService</es:serviceName>
        <es:WSDL>http://v-tml.uni-
muenster.de:8080/EventService/services/Broker?wsdl</es:WSDL>
    </es:Publisher>
  </es:EventServiceReference>
</ows:ExtendedCapabilities>
```

7.6 Accomplishments

Within this OWS, several new aspects and features which improve the data provision services were designed and implemented. Most of these new features were designed to make the life of clients using these services easier (e.g. the introduction of Stored Queries/Filter into the Aviation architecture, "AIXM Features as Geometry Operands" as well as "Spatial Filtering on Non-spatial Features"). Additionally, a special focus was laid on performance and bandwidth improvement. This has been achieved for instance by defining the explicit exclusion of metadata when receiving data on AIXM features but also through the introduction of "feature provision by reference" (through the gml:identifier) and the design of "Update Intervals" for Event Service subscriptions. The latter also helps a client (e.g. a pilot) to focus on the most relevant information.

In contrast to previous OWS test beds efforts on creating filters which take the vertical extent of features into account have been made. In particular, the design of simple altitude queries forms a valuable basis for future scenarios which require the filtering based on altitude limits.

7.7 Lessons Learned

Taking the vertical extent of AIXM features into account when applying filtering has been identified as a challenging task. Especially when dealing with complex Airspace geometries or three-dimensional flight routes a simple approach was not sufficient. The work on altitude queries improved the overall comprehension on complex AIXM features and is reflected in a matured filtering capabilities.

The work on Selective Metadata Retrieval highlighted the complexity of metadata handling within data provision and Aviation architecture in general. A first approach planned to provide functionality on both inclusion and exclusion of metadata. This could not easily be achieved as remote metadata should ideally not be provided through data provision services but through a CSW. Additional efforts on embedding a CSW into the data provision workflow are required to foster this work area. Additionally, other encodings for Selective Metadata Retrieval have been discussed (e.g. defining the PropertyExclusion outside of the wsnt:Filter element). The agreement on the provided encoding has been made as it can be used in combination with Stored Filters. A definition outside of wsnt:Filter is currently not supported through Stored Filters as these only provide a shortcut to the wsnt:Filter element. Hence, one future work item is to develop a more general projection clause support for ES subscriptions (e.g. in alignment with WFS queries).

7.8 Future Work

During the design and implementation of the advanced filtering functionality several future work items have been identified. The following list provides a summary on these.

- **Stored Filters** – Currently, the design of Stored Filter only takes the wsnt:Filter part of an Event Service subscription into account. In order to provide other aspects of a subscription through a stored mechanism, a more general approach should be designed, namely Stored Subscriptions. Thus it would be possible to define policies and projection clauses in addition to the filter part.

- **Selective Metadata Retrieval** – The current design of Selective Metadata Retrieval only provides capabilities for the exclusion of metadata. One future work item is to develop a mechanism to allow explicit inclusion of metadata (inline or by reference). Here, several aspects such as the dynamic creation of remote metadata instances (e.g. through a catalog service) need to be considered. Another future work item is the general definition of projection clauses for Event Service subscriptions. The current approach has the drawback that it cannot be used without a Filter.

- **Altitude Queries** – A transition from “simple” to “complex” altitude queries is a very interesting field of work. This requires detailed work on the various ways of how to encode altitude limits within AIXM (e.g. taking the unit of measurements into account). The most complex situations occur when one of two geometry operands provides altitudes as flight levels. Here, barometric pressure calculations need to be taken into consideration. A complex altitude query design could also be aligned with future work on the Geometry Processing field of work.

- **AIXM Features as Geometry Operands** – It has been identified within this test bed that an AIXM might have multiple geometries defined. The current design does not take such cases into consideration. Future work should focus on this by introducing, for instance, a “scenario” parameter which defines how the geometry should be computed (e.g. simple locating vs. complex visualization of a feature). Such work should also be aligned with Geometry Processing work areas.

- **Event Service Update Intervals** – The design of Update Intervals already provides encodings to define the behavior on how to treat non-related feature events as well as the detection of duplicate events. Nevertheless, the development of algorithms for these cases is a complex venture and thus has been postponed. Future work on this feature should focus on the design and implementation of such algorithms.

8 Geometry Processing via Web Processing Service

8.1 Overview

Web Processing Services (WPS) encapsulate a set of specific processing functionality and make it available to clients via the web. Processing tasks common to a given domain can therefore be performed by a dedicated and re-usable component. This approach offers a way to have computing heavy tasks performed by the WPS on behalf of clients that may only have limited resources. This section describes the approach developed within OWS-9 to integrated WPS for geometry processing into the service environment. It provides a summary of the developed WPS profiles, including the defined processes (with inputs and outputs). Additionally, relevant use cases for Aviation-specific WPS utilization are presented and incorporated into possible future work items.

8.2 Applications Profiles

As stated by [OGC 05-007r7], a WPS server provides a significant set of web service interfaces which can be reused by client applications. Nevertheless, the actual functionality of a WPS instance is encapsulated in its processes. To achieve real interoperability among different WPS instances they need to implement a certain profile (of a process) also denoted as an *Application Profile.* Such a profile must consist of a unique identifier (represented as a URL) as well as a reference response to a DescribeProcess request.

Within this test bed three Application Profiles were developed for the Aviation domain. Besides geometry operations (intersection calculation, geometry touching) an additional profile was introduced to compute/resolve the GML geometry of an AIXM feature. All processes took AIXM 5.1 data as input. As AIXM 5.1 uses GML 3.2.1 for geometry representations, it is used if a geometry is the result of a process.

Within this OWS, the processing capabilities of all developed processes were limited to 2D computations. See the “Future Work” section for plans on how to deal with 2.5/3D geometries. All calculations were performed geodetically for increased accuracy for the following profiles. Especially when dealing with geometries of a wide spatial extent (e.g. airspaces spanning over multiple countries and intercontinental flight routes) taking the curvature of the earth (e.g. Great Circle vs. Rhumbline interpolation) into account is inevitable.

As all designed ProcessDefinitions took AIXM features as inputs, implementations needed to be able to resolve the actual geometry of an AIXM feature. See section 7 for details on how this was achieved.

The following sections provide information for the profiles developed within this test bed. Additionally, these can be considered as the *human-readable document that describes the process and its implementation* (see OGC 05-007r7, clause 6.4) which are denoted as recommended for each Application Profile.

8.2.1 Intersection Calculation

The intersection profile describes the intersection of two AIXM 5.1 Features based on their geometries. The output was a GML3.2 MultiGeometry element that contained the intersection results. The MultiGeometry can be empty if no intersection occurs. The profile imposed no limit on the type of AIXM 5.1 feature that could be supplied to the service, but, for this OWS, using intersection testing between Airspace and RouteSegment elements were developed and tested. These features usually contained a single geometry element, unlike some other top-level AIXM features that don't contain geometry. Airspaces contain surfaces such as polygons) and routes contain polylines.

8.2.1.1 Unique identifier

The URL for AIXM Intersection Calculation was

```
http://www.opengis.net/ows9/aviation/wps/intersection
```

8.2.1.2 Inputs

The Inputs for this Process were two AIXM features. Within the scope of this OWS, both WPS implementations provided support for aixm:Airspace and aixm:RouteSegment.

The ProcessDefinition defined one wps:Input element (identifier “Feature”) with exactly two occurrences which allowed every AIXM feature defined in AIXM_Features.xsd as inline content.

8.2.1.3 Outputs

The result of this Process was defined by a schema developed for this purpose. The schema contained one element of type gml:MultiGeometry. This allowed the encoding of results with all geometric dimensions. The identifier for the result was simply “Result”. An example result is illustrated by Figure 18.

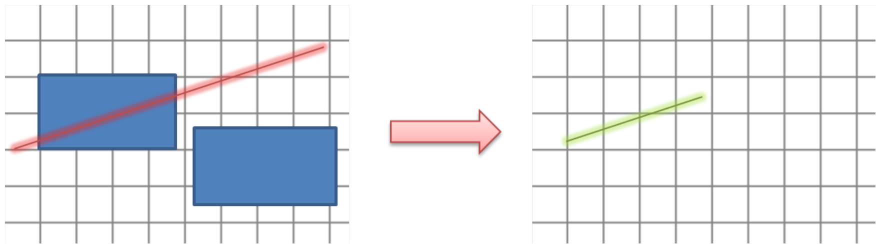


Figure 18 – Intersection of Airspace and RouteSegment.

8.2.1.4 Reference ProcessDescription

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessDescriptions xmlns="http://www.opengis.net/wps/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd" xml:lang="en-US"
service="WPS" version="1.0.0">
    <ProcessDescription xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink" processVersion="1.0.0" storeSupported="false"
statusSupported="false">
        <ows:Identifier>AIXMIntersection</ows:Identifier>
        <ows:Title>Calculate Airspace-Route Intersection</ows:Title>
        <ows:Abstract>
      Calculates the intersection points for two given AIXM features.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="intersects"/>
        <ows:Metadata xlink:title="AIXM"/>
        <ows:Metadata xlink:title="GML"/>
        <DataInputs>
            <Input minOccurs="2" maxOccurs="2">
                <ows:Identifier>Feature</ows:Identifier>
                <ows:Title>Input Feature</ows:Title>
                <ows:Abstract>The input features.</ows:Abstract>
                <ComplexData>
                    <Default>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</Schema>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</Schema>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Multipoint geometry</ows:Title>
                <ows:Abstract>The output, defined as a GML3.2 MultiPoint element.
If no intersections occurred,
          the MultiPoint will be empty.
        </ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>text/xml</MimeType>
                            <Schema>http://test.schemas.opengis.net/ows-
9/aviation/wps/aixmIntersectionResult.xsd</Schema>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>text/xml</MimeType>
                            <Schema>http://test.schemas.opengis.net/ows-
9/aviation/wps/aixmIntersectionResult.xsd</Schema>
                        </Format>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://schemas.opengis.net/gml/3.2.1/geometryAggregates.xsd
</Schema>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</ProcessDescriptions>
```

8.2.2 Spatial Relation

The spatial relation profile describes 9 distinct spatial relation operations between geometries. The input for each of the described processes was 2 AIXM features (identifiers "Feature1" and "Feature2"), and the output was a Boolean literal (identifier "Result"). The various operations were:

- Contains.
- Covers.
- Crosses.
- Disjoint.
- Equals.
- Intersects.
- Overlaps.
- Touches.
- Within.

The processes did not mandate any specific features, but we proposed using only Airspace and RouteSegment elements for this thread.

Input features may contain multiple disjoint geometries (e.g. Airspace with multiple ElevatedSurfaces). Spatial relation was computed on all geometries of both input features (e.g. feature 1 intersects with feature 2 if one or more of the geometries in feature 1 intersect with one or more of feature 2). Especially for Airspaces with multiple AirspaceGeometryComponents, the geometric operations (e.g. "UNION") used to define the complex geometry were taken into consideration in the process evaluation.

Figure 19 illustrates the inputs and results of the "Intersects" Process.

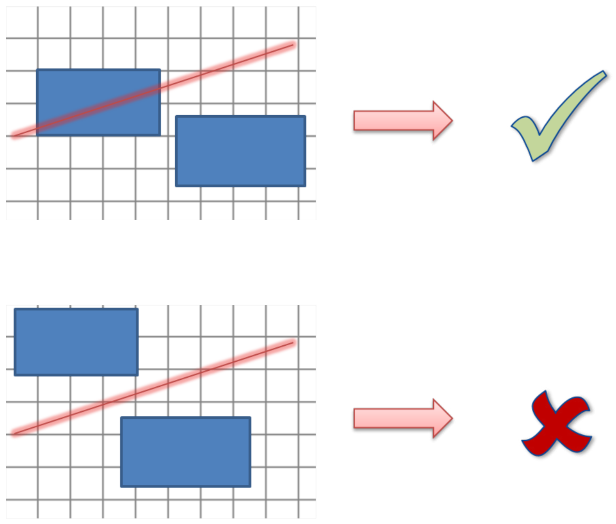


Figure 19 – Spatial Relation "Intersects" for Airspace and RouteSegment.

8.2.2.1 Unique identifier

The URL for AIXM Spatial Relation was

```
http://www.opengis.net/ows9/aviation/wps/spatialRelation
```

8.2.2.2 Reference Process Descriptions

Due to the high amount of XML markup, the process descriptions are documented in Annex D of this document.

8.2.3 Geometry Retrieval

The geometry retrieval profile can be used to resolve the geometry from an AIXM feature. As AIXM features can vary over time (represented by multiple TimeSlices), the point in time for geometry resolving can be defined within a request. If such a parameter is not provided, the geometry is calculated for the current time.

Implementations of this profile should also be able to resolve a geometry from AIXM features which do not define a geometry themselves. Such feature geometry could be computed from associated features. A valid example for such a situation is a Runway (defining no geometry) with associated RunwayElements. The implementation should be able to request additional information (e.g. from a WFS) if the input data is not sufficient.

One specialty of AIXM is that airspaces contain surfaces (AirspaceVolumes) that can be formed using geometric operations (difference, union, intersection). It is up to the WPS server to return a set of disjoint GML surfaces that represent the original airspace. Additionally, airspaces contain altitude data. This cannot be modeled by GML. Currently, the team is discussing the creation of a custom GML schema that explicitly models the altitude. Therefore the support of altitudes within this profile had no absolute guarantee within OWS-9.

8.2.3.1 Inputs

The Inputs for this process are one AIXM feature and an optional date time. The date time is used to resolve the geometry of the feature for a specific valid time. If it is not provided, this input defaults to the current time.

The input AIXM feature has two supported variants: Inline or by reference. The inline variant (the default) allows clients to supply the AIXM feature in its entirety, embedded into the WPS execute.

The 'by reference' variant makes use of a newly developed AIXM extension schema. This schema allows AIXM features to be referenced to by their GML identifier. Section 8.2.3.6 contains the proposed AIXM extension schema.8.2.3.6

See Section 7 for the list of supported input AIXM features.

8.2.3.2 Outputs

The output of this process is an element as defined within the Geometry Aggregates subschema of GML 3.2.

8.2.3.3 Algorithm

The geometry retrieval algorithm consists of several phases.

First, if the given input feature is a reference (i.e. If it uses the schema in section 8.2.3.6), the feature for the reference is resolved on the server. This is accomplished with a GetFeature query that contains a list of all feature type names supported by the server. This feature is needed because based on the reference, we cannot know what type of feature we are dealing with. It is important to know the feature type, as different features will link to geometry in different ways.

Next, using the resolved (or given) feature, we first have a look at the type of feature we are dealing with. This part of the algorithm is highly format dependent. For example, a Taxiway feature contains an associated AirportHeliport, which contains a point-based geometry. However, this geometry is only secondary information. What we're really interested in is the collection of TaxiwayElement features that have the given Taxiway as associated Taxiway (through reverse association).

Another example would be a RouteSegment, which contains a GML curve, as well as separate start and end positions. Generally speaking, we are only interested in obtaining the GML curve that represents the RouteSegment.

We make a distinction between AIXM features that contain their primary geometry directly versus AIXM features that do not contain their primary geometry directly. Two examples of AIXM features that contain their primary geometry directly are Airspaces and RouteSegments.

When a given feature does not directly contain its primary geometry, we have to perform an additional query on the WFS server to resolve a set of new features that do contain the necessary geometry. Many of the examined features made use of reverse associations. For example, Runway has no links to RunwayElements, but RunwayElement features do have links to Runways in the form of associations. For OWS9, we have implemented support for the following features:

- Runway → RunwayElement
- Apron → ApronElement
- Taxiway → TaxiwayElement

For this we make use of the wfs:valueOf() function, as documented in paragraph 7.3.2 of the WFS 2.0 specification. The wfs:valueOf() allows us to retrieve the XLink value of a property. This means that we can easily create a filter that finds all RunwayElements with a link to our original Runway feature. An example of what this WFS query would look like can be seen in the query below:

```
  <wfs:Query xmlns:aixm="http://www.aixm.aero/schema/5.1" typeNames="aixm:RunwayElement">
    <fes:Filter>
      <fes:PropertyIsEqualTo>
<fes:ValueReference>wfs:valueOf(*/*/aixm:associatedRunway)/*/gml:identifier</fes:ValueRef
erence>
        <fes:Literal>009AA70D-C240-4A66-9873-9ADC1F744C76</fes:Literal>
      </fes:PropertyIsEqualTo>
   </fes:Filter>
  </wfs:Query>
```

Finally, after all necessary features are retrieved, we generate a SNAPSHOT timeslice on the results, based on the given input date. For more information on SNAPSHOT generation, please see the AIXM 5.1 temporality model documentation.

8.2.3.4 Unique identifier

The URL for AIXM Geometry Retrieval was

`http://www.opengis.net/ows9/aviation/wps/geometryRetrieval`

8.2.3.5 Reference Process Description

```
<ProcessDescriptions xmlns="http://www.opengis.net/wps/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd" xml:lang="en-US"
service="WPS" version="1.0.0">
    <ProcessDescription processVersion="1.0.0" storeSupported="false"
statusSupported="false">
        <ows:Identifier>ResolveAIXMFeatureGeometry</ows:Identifier>
        <ows:Title>Retrieve Geometry of AIXM feature</ows:Title>
        <ows:Abstract>Retrieves the geometry of the given AIXM feature for a given time
instance.</ows:Abstract>
        <ows:Metadata xlink:title="geometry"/>
        <ows:Metadata xlink:title="retrieve"/>
        <ows:Metadata xlink:title="AIXM"/>
        <ows:Metadata xlink:title="GML"/>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature</ows:Identifier>
                <ows:Title>Input Feature</ows:Title>
                <ows:Abstract>The feature from which to retrieve the
geometry.</ows:Abstract>
                <ComplexData>
                    <Default>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</Schema>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</Schema>
                        </Format>
                        <Format>
                            <MimeType>text/xml</MimeType>
                            <Schema>http://test.schemas.opengis.net/ows-
9/aviation/aixm/aixm-extension.xsd
</Schema>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
            <Input minOccurs="0" maxOccurs="1">
                <ows:Identifier>Date</ows:Identifier>
                <ows:Title>Date</ows:Title>
                <ows:Abstract>The date for which to retrieve a geometry (validTime of a
timeslice). If not provided, the current time is the default.</ows:Abstract>
                <LiteralData>
                    <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-
2/#dateTime">dateTime</ows:DataType>
                    <ows:AnyValue/>
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Geometry</ows:Title>
                <ows:Abstract>The output geometry.</ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>text/xml</MimeType>

    <Schema>http://schemas.opengis.net/gml/3.2.1/geometryAggregates.xsd
</Schema>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
```

```
                                <MimeType>text/xml</MimeType>

    <Schema>http://schemas.opengis.net/gml/3.2.1/geometryAggregates.xsd
</Schema>
                            </Format>
                        </Supported>
                    </ComplexOutput>
                </Output>
            </ProcessOutputs>
        </ProcessDescription>
</ProcessDescriptions>
```

8.2.3.6 AIXM extension schema

The AIXM extension schema to reference AIXM features by GML identifier in the geometry retrieval profile can be seen here:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:aixm-
ext="http://www.opengis.net/ows9/aviation/aixm/extension"
targetNamespace="http://www.opengis.net/ows9/aviation/aixm/extension"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://www.opengis.net/gml/3.2"
schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <xs:element name="FeatureIdentifier" type="aixm-ext:FeatureIdentifierType">
        <xs:annotation>
            <xs:documentation>The FeatureIdentifier to enable feature-by-reference
through gml:identifier
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="FeatureIdentifierType">
        <xs:sequence>
            <xs:element name="identifier" type="gml:CodeWithAuthorityType"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

8.2.4 Limitations

The implementations of the geometric calculation profiles within OWS-9 were limited to support aixm:Airspace and aixm:RouteSegment as inputs. This could easily be extended with additional development effort. Currently, the geometric calculations only support the horizontal extent of these features.

The WPS for the “Geometry retrieval” profile is currently limited to aixm:Airspace, aixm:RouteSegment, aixm:AirportHeliport, aixm:Runway, aixm:RunwayElement, aixm:Taxiway, aixm:TaxiwayElement, aixm:Apron and aixm:ApronElement. This could as well be extended with additional effort.

8.3 Accomplishments

Within this OWS-9, the implementation of two separate WPS instances for specific Aviation processing tasks was achieved. Based on well-defined ProcessDefinitions, both implementations were truly interoperable, thus enabling an easy integration into existing client and server software.

Additionally, several interesting fields of work were identified where the integration of a WPS into the overall Aviation architecture seemed of great value.

By integrating generic processing capabilities into the Aviation architecture, client and service instances can leverage a wide range of possible functionality. Within this OWS, the benefits of geometric calculations which have been used by the Luciad client and the Event Services were demonstrated. The main benefit was in the capability to take complex AIXM features as inputs and provide simple responses. Thus, clients and services would not need to take care of the internal data structure of a specific AIXM feature. For instance, Event Services could make use of this feature when receiving a Subscription and processing Digital NOTAMs.

From the client perspective, the WPS approach offers many opportunities. As spatial computations are likely to be very costly clients running on handhelds or similar devices can focus on other important tasks by outsourcing the costly processing to a WPS.

8.4 Lessons Learned

- When designing a WPS ProcessDefinition the format for inputs and outputs can be defined in a lax way by pointing to a commonly available XML Schema. By implication a WPS implementing such ProcessDefinition must support all available root elements as inputs/outputs. To enable true interoperability among different WPS instances, one should be as exact as possible defining the inputs/outputs. Within OWS, we decided to define a new schema for the result output. The schema actually subsets GML 3.2 to only allow one element of gml:MultiGeometry. It should be considered in future revisions of the WPS standard if a sub-setting of an XML schema (e.g. via a list of supported QNames) could be supported within a ProcessDefinition.

8.5 Future Work

- **2.5D, 3D calculations** – The current OWS-9 implementations of the WPS only support the horizontal extent of AIXM features. One future work item should focus on the integration of the vertical extent. Here, it is to be decided if a general approach for transforming AIXM 2.5D geometries into GML 3D geometries should be developed.

- **Additional Profiles** – Besides the three described, additional profiles should be developed. This includes but is not limited to the calculation of actual altitudes (e.g. based on barometric pressure, DEMs), calculation of the containing circle for an Airspace border (used for a Digital NOTAM's Q line), metadata/provenance resolving.

- **Bulk Operations –** The current profiles for spatial operations and intersection calculations only allow you to do individual geometric operations on 2 AIXM features. In the demo we demonstrated the calculation of one RouteSegment with multiple Airspaces at the same time. Each of these operations required a separate query to the server. A performance optimization would be to allow a client to send a list of airspaces, instead of individual airspaces. This would make the result more complex though.

- **Catalogue Service for referenced features** – One potential improvement for geometry retrieval by reference would be the use of a Catalogue Service to retrieve the server on which a feature with a given gml:identifier exists. Even better would be if the Catalogue Service could also supply the feature type in the same request. This would reduce the overhead of having to perform multiple WFS GetFeature queries to the same WFS service.

9 FAA SWIM Compliance Analysis

9.1 Introduction

The purpose of the SWIM Compliance Assessment is to provide information to the OGC and FAA on:

- The alignment of FAA SWIM compliance requirements and OGC standards.
- The applicability of a given FAA SWIM requirement to OWS-9 services, clients, or components.

The source for FAA SWIM compliance requirements used in the assessment is http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/atc_comms_services/swim/documentation/media/compliancy/SWIM%20Service%20Compliance%20Requirements.pdf. For the remainder of this chapter, "SWIM" always refers to "FAA SWIM".

In recognition that some SWIM compliance requirements appear to be focused on FAA programs that operate in the SWIM environment, the assessment was conducted from the perspective of envisioning a future where there is a bi-directional flow of information between OGC services and FAA programs, and where OGC services are used by FAA programs in the FAA SWIM environment. From this perspective, the assessment process was designed to solicit input on SWIM requirements that would not be applicable to web services data exchanges involving external entities.

9.2 Analysis Process

The analysis process flowed from transferring SWIM Segment 1 requirements to a matrix form that was used to collect data from the component developers as shown in Figure 20.

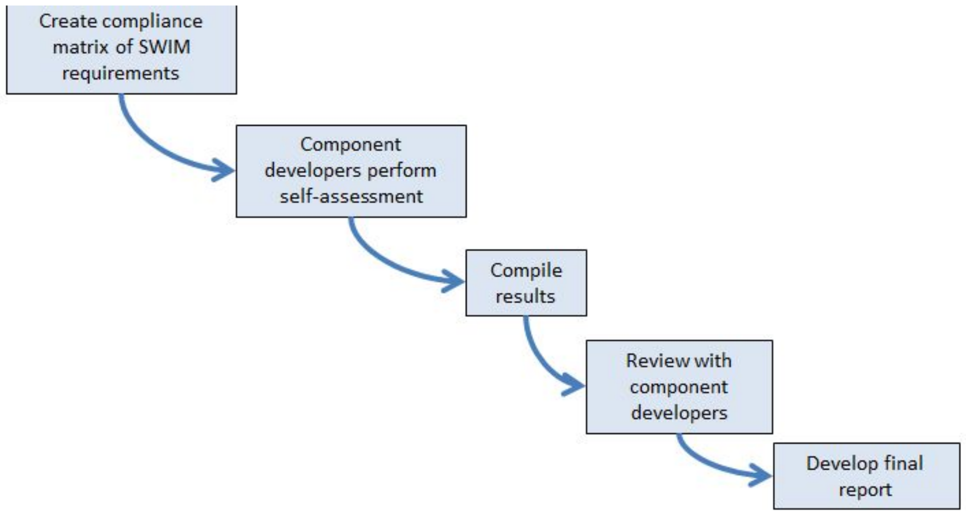

Create compliance matrix of SWIM requirements
Component developers perform self-assessment
Compile results
Review with component developers
Develop final report

Figure 20 SWIM Compliance Assessment Process

The assessment checklist located in Annex A was developed to incorporate relevant aspects of SWIM requirements published by the FAA. This checklist was completed by the services teams in the Aviation Thread. Their inputs were compiled and analyzed to produce the findings and observations and final report of the level of FAA SWIM Compliance and associated recommendations. These teams included:

- 52 North.
- COMSOFT.
- Envitia.
- Galdos.
- Luciad.
- IDS.
- IfGI.
- Snowflake.

9.3 Assessment Results

The following table summarizes the results of the compiled assessments.

Reference	Requirement	Summary Assessment
4.1 Technology Acquisition		
SWIM-SC-0001	SWIM services SHALL be implemented using standardized service container products and product versions that are specified in the SWIM Product Standardization (SPS) document and provided by the SWIM Program Office and managed as part of the SWIM COTS Product Repository (ref: SWIM COTS Product Management Plan).	**Not Applicable** - Applies to technology acquisition officials
SWIM-SC-0002	Service implementing programs SHALL acquire approved Commercial-Off-The-Shelf (COTS) products only through the SWIM COTS Product Repository (SCPR) in accordance with the SWIM COTS Product Management Plan.	**Not Applicable** - Applies to technology acquisition officials
4.2 Interoperability, Reuse, and Standards		
4.2.1 Use of Open Standards		
SWIM-SC-0010	SWIM services SHOULD apply open data standards, schemas, and message interfaces where possible, instead of creating new schema definitions.	**OGC Standards Perspective - Fully Compliant** OGC standards are based on open standards and are themselves open standards. **Product Implementation Perspective - Fully Compliant** Most of the services deployed in OWS-9 are defined by approved OGC standards (WPS 1.0, WFS 2.0, WMS/FPS, CSW ebRIM). The Event Service is a Broker according to the OASIS WS-Notification set of standards. However, for the test bed, some processing functionality may not be specified by approved open standards, (e.g. new WPS profiles or Event Service filtering). This functionality will be picked up by the OGC's standardization process (one or more OGC Standards Working Groups would pick up the new functionality) to eventually become an approved open standard.

SWIM-SC-0020	SWIM services SHALL provide an interface that supports one or more of the following message format, message protocol, and transport protocol combinations: • SOAP-over-HTTP/HTTPS • XML-over-HTTP/HTTPS, inclusive of the Representative State Transfer (REST) interface pattern • SOAP-over-JMS • XML-over-JMS	**OGC Standards Perspective - Fully Compliant** XML-over-HTTP is the most commonly supported format/protocol/transport combination. Exchanging web service messages via JMS should also be possible using the W3C SOAP JMS binding. **Product Implementation Perspective – Fully Compliant** Each of the providers supports at least one of the combinations and could support more if required.
SWIM-SC-0021	SWIM services MAY use HTTPS or TLS as the transport protocol in place of HTTP or TCP for service interface interactions.	**OGC Standards Perspective - Fully Compliant** OGC services are usual web services and thus all components should be able (at least with minor modifications) to support HTTPS/TLS. **Product Implementation Perspective - Fully Compliant** Security is generally implemented by OGC services at the HTTPS, SOAP, REST layers as a common practice.
4.2.3 SOAP Message Processing		
SWIM-SC-0030	SOAP Messages in the SWIM environment SHALL be processed using the FUSE Service Framework library.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** We have results from previous studies (see OWS-7 Engineering Report) that tested the deployment of OGC compliant web services in a FUSE environment with the service deployed as a web application. Revamping the implementation of the service itself should be possible if the product was really required to use the FUSE libraries.
4.2.4 SOAP Messaging WS-I Compliance		
SWIM-SC-0040	SWIM Service endpoints SHALL meet the messaging compliance requirements of the SWIM Interoperability Basic Profile (a SWIM-annotated version of the Web Services Interoperability Basic Profile, Section 3).	**OGC Standards Perspective - Fully Compliant** OGC web service standards appear to be compatible with the WS-I Basic Profile and also the SWIM-annotated WS-I Basic Profile. **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.

4.2.5 Binary Attachments in SOAP Messages		
SWIM-SC-0045	Any binary attachment that is sent with a SOAP message SHALL be attached and processed using the Message Transmission Optimization Mechanism (MTOM), in accordance with the SWIM Governance Plan.	**OGC Standards Perspective - Fully Compliant** This requirement does not appear to conflict with OGC standards and practices. The OGC "OWS 5 SOAP/WSDL Common Engineering Report" Public Engineering Report recommends the use of MTOM in a SOAP binding to transfer large binary data. **Product Implementation Perspective - Provisionally Compliant** MTOM is not being used for any of the OWS-9 components but could be used as an implementation choice if required.
4.2.6 Data Retrieval Protocol		
SWIM-SC-0048	Service providers MAY use transport protocols and message formats other than those defined in SWIM-SC-0020 for binary and large file data delivery and retrieval. For example: FTP, SMTP, or SCP may be applied to delivery large binary files. The alternate protocol SHALL NOT be used for service requests or notifications, but MAY be used for content delivery.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** The WPS Schema supports this requirement by pointing to the binary file (image) location. If required for the implementation, this requirement can be satisfied within the OGC standards set.
4.2.7 JMS Provider Standardization		
SWIM-SC-0050	SWIM JMS message producers SHALL use FUSE Message Broker as the JMS Provider for JMS destinations	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** JMS hasn't been widely implemented by the OGC community or its products due an emphasis on broad interoperability through use of common protocols and simple message exchange patterns. As requirements for JMS messages appear, OGC participant's products will evolve as needed.
SWIM-SC-0051	SWIM JMS clients MAY institute a JMS bridge between FUSE Message Broker and the service provider's message-oriented-middleware (MoM).	**Not Applicable** to OGC standards This requirement addresses the establishment of a communications link between services.
SWIM-SC-0052	SWIM JMS Message Brokers SHALL be configured to support the OpenWire protocol over TCP or SSL.	**Not Applicable** to OGC standards This is a requirement against the JMS Message Broker and does not apply to OGC service components.
SWIM-SC-0053	SWIM JMS Message Brokers MAY be configured to support the Simple Text Object Messaging Protocol (STOMP) protocol over HTTP or HTTPS	**Not Applicable** to OGC standards This is a requirement against the JMS Message Broker and does not apply to OGC service components.

4.2.8 JMS Destination Names		
SWIM-SC-0055	JMS destination names SHALL include the FAA Business Context Identifier (FBCI) of a SWIM-registered namespace as the first prefix for the destination name, as described by [STD063]. For example, the namespace: urn:us:gov:dot:faa:AviationSafety may correspond to the JMS destination name: "AviationSafety.topic.IncidentReports".	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.
4.2.9 JMS Message Type		
SWIM-SC-0060	Messages sent using JMS SHALL use an XML format, and MAY use the SOAP message format.	**OGC Standards Perspective - Fully Compliant** With the W3C standard "SOAP over Java Message Service 1.0" OGC based services should be able to exchange XML (more specifically: SOAP) messages via JMS. **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.
SWIM-SC-0061	SOAP and XML messages that are sent over JMS SHALL use the JMS TextMessage type.	**OGC Standards Perspective - Fully Compliant** The W3C "SOAP over Java Message Service 1.0" standard requires that TextMessage is supported. It also lists some concerns but in general it looks that with the W3C standard OGC based services are able to exchange SOAP messages via JMS using the JMS TextMessage type. **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.

4.3 Registry / Repository		
4.3.1 Interface Discoverability		
SWIM-SC-0070	WSDL documents corresponding to SWIM service endpoints SHALL be registered with the SWIM Service Registry / Repository in accordance with TBD. One WSDL document may describe many endpoints.	**OGC Standards Perspective - Fully Compliant** The OGC service standards define web service operations and schema, which can easily be integrated in a WSDL description of the service. **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.
SWIM-SC-0071	SWIM service WSDL documents SHALL NOT be published through any mechanism other than the SWIM Service Registry / Repository. Those mechanisms prohibited include, but are not limited to direct publishing using HTTP from the service provider site.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied by service providers.
4.3.2 Interface Categorization		
SWIM-SC-0080	SWIM services SHALL be categorized in the SWIM Service Registry/Repository as described in FAA-STD-064 and FAA-STD-066.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to create this information.
SWIM-SC-0082	XSD schemas that define the structure for SWIM service messages SHALL be categorized in the SWIM Service Registry/Repository using SWIM service taxonomy categories as described in TBD.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to create this information.
4.4 Namespace and Schema		
SWIM-SC-0090	SWIM service WSDL documents SHALL define services within a namespace that has been registered by the service provider in the FAA Data Registry (FDR).	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the

		standards to create this information.
SWIM-SC-0091	SWIM service message schemas SHALL use namespaces that have been registered in the FAA Data Registry (FDR).	**OGC Standards Perspective - Fully Compliant** Schemas are defined in OGC standards for the information exchanges and are registered in the namespace defined by the OGC. These namespaces would need to be registered with the FAA Data Registry FDR (by an implementer, FDR administrator or other entity). **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set.
4.5 Service Interface Design		
SWIM-SC-0100	SWIM service interfaces SHALL be described by a Web Service Definition Language (WSDL) v2.0 document.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to create this information.
SWIM-SC-0101	SWIM service interfaces MAY be described by a WSDL v1.1 document.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to create this information.

SWIM-SC-0102	The message content that may be sent or received by a SWIM service SHALL be described by one or more XML Schema Definition (XSD) documents.	**OGC Standards Perspective - Fully Compliant** However, it is appropriate to note that even if there is a schema, its contents still do not define all necessary details to make use of messages that comply to it. The schema may allow for dynamic or even any content (using substitution groups or elements of anyType, or "any" elements). In addition the semantic of the encoded data is usually described elsewhere. **Product Implementation Perspective - Fully Compliant**) This is certainly fulfilled by all OGC compliant services in implementation.
SWIM-SC-0103	The message content schema for messages that may be sent or received by a SWIM service SHALL NOT be defined in the WSDL document, and SHALL be in a separate XSD.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to create this information.
SWIM-SC-0104	SWIM services shall be described by an FAA Web Service Definition Document (WSDD) in accordance with STD065, P*reparation of Web Service Description Documents*.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** A WSDD is a more detailed service description and is designed to communicate service capabilities to human readers. If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer can develop this information if required.

4.5.2 Service Interface WS-I Compliance		
SWIM-SC-0110	SWIM service WSDL interface descriptions SHALL be compliant with the Service Description requirements defined in the SWIM Interoperability Basic Profile (a SWIM-annotated version of the Web Services Interoperability Basic Profile v1.2, section 4).	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** Although some participants indicate that they support this requirement, most products do not at this time. If required for the implementation, this requirement can be satisfied within the OGC standards set. The product developer has the tools within the standards to satisfy this requirement.
4.6 Information Security		
SWIM-SC-0120	SWIM services shall implement security consistent with NIST Special Publication 800-95 Guide to Secure Web Services [NIST800-95].	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** Security is generally implemented by OGC services at the HTTPS, SOAP, REST layers as a common practice. Higher level application layer security including registry security, e.g. identity management and access controls, are usually applied outside of the services to ensure that the user is authorized to access the service and has the permissions for the types of data requested. Additional security measures can be implemented depending upon the requirements.

SWIM-SC-0121	SWIM services shall be compliant with the requirements defined in the SWIM Interoperability Basic Security Profile (a SWIM-annotated version of the Web Services Interoperability Basic Security Profile).	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** Security is generally implemented by OGC services at the HTTPS, SOAP, REST layers as a common practice. Higher level application layer security including registry security (e.g. identity management and access controls) are usually applied outside of the services to ensure that the user is authorized to access the service and has the permissions for the types of data requested. Additional security measures (e.g. those in the SWIM WS-I Basic Security Profile) can be implemented depending upon the requirements.
4.7 Service Management		
SWIM-SC-0130	SWIM services SHALL use Java Management Extensions (JMX) for management and monitoring services at runtime.	**OGC Standards Perspective - Fully Compliant** **Product Implementation Perspective - Provisionally Compliant** Some service providers support JMX already at some level but most do not, however their products can be extended to support this requirement. If required for the implementation, this requirement can be satisfied within the OGC standards set.

9.4 Findings and Observations

This section contains topics of interest resulting from the analysis of the inputs on the SWIM Compliance Matrix provided by the component development teams. Analysis of their responses indicated that three categories of compliance needed to be defined:

- OGC Standards Perspective – to identify those SWIM compliance requirements which conflict with OGC standards. The possible ratings in this category were "fully compliant," "provisionally compliant," or "not compliant".

- Product Implementation Perspective – to identify those SWIM compliance requirements that may not be commonly supported in current services but which could be supported if required, i.e. there are no limitations from an OGC standards perspective on their implementation. The possible ratings in this category were "fully compliant," "provisionally compliant," or "not compliant".

- Not Applicable – was used to identify those SWIM compliance requirements which were either not relevant to service implementations (e.g. SWIM-SC-0001 and SWIM-SC-0002) which apply to technology acquisition officials, or SWIM-

SC-0051, SWIM-SCV-0052 and SWIM-SC-0053 which apply to communications among message brokers – a SWIM internal activity.

9.4.1 SWIM Requirements

The FAA SWIM Compliance document lists 31 requirements for compliance with SWIM. Two of those requirements, SWIM-SC-0001 and SWIM-SC-0002, were determined to be not applicable to OGC products in that they address technology acquisition requirements. The following table describes the levels of compliance with the remaining requirements.

Summary Assessment	Number of Requirements
OGC Standards Perspective	
Fully Compliant	26
Provisionally Compliant	0
Not Applicable	3
Product Implementation Perspective	
Fully Compliant	4
Provisionally Compliant	22
Not Applicable	3

9.4.2 Observations

- It was noted that the SWIM Requirements were defined for SWIM Segment 1 and the FAA is transitioning to SWIM Segment 2. The OWS-9 team confirmed with FAA representatives that the technical requirements and standards would be relatively unchanged from Segment 1 to Segment 2 so the assessment would still provide a valid indicator for the support of SWIM (compliance) requirements by OGC standards in general and OWS-9 service implementations in particular.

- The SWIM security requirements provided an interesting topic of discussion. A review of the SWIM-annotated Web Services – Interoperability Basic Security Profile Version Number 1.0 provided insight into three SWIM requirements that would be in addition to the WS-I Basic Security Profile Version 1.1. These requirements deal with replay of the username token and encryption with signatures, both aspects that can be implemented in a secure way within the OGC standards framework to prevent a security risk. In OGC Web Services, security functionality is separated as much as possible from service implementation. This practice enables the provision of security capabilities through separate security services (e.g. authentication, authorization and audit services) which can be flexibly combined and used in different configurations to suit the requirements of a specific implementation. This provides scalable solutions that allow new security services to be implemented without affecting other services (e.g. WFS, ES, WPS). More information on OWS security implementation can be located at:

- OWS-8 Aviation Architecture Engineering Report
- OWS-6 Security Engineering Report

9.5 Summary and Recommendations

In general, it appears that most of the SWIM compliance requirements can be supported within the framework of OGC standards. While many of the SWIM compliance requirements have not been implemented in products of OWS-9 Aviation service providers, the general consensus was that they could be implemented if required. Some recommendations that flowed out of the analysis and discussions about SWIM standards included:

- The path and processes for submitting products for consideration for inclusion on the SWIM product list need to be more accessible and available for OGC partners.
- Clarification of the relationship between the technology requirements and standards for SWIM Segment 2 needs to be provided to the community to enable continuing development of OGC and SWIM compliant products.

10 Scenario

10.1 Introduction

The scenario used for OWS-9 is a variation of the scenario used for OWS-8. As shown in Figure 21, the scenario describes a flight originating from a European airport, Donlon (EADD , deviating around en route weather and then receiving a redirection to an alternate airport, Milwaukee International (MKE), based on a fire event at its destination. OWS-9 provides information services for pre-flight planning, en route planning, and approach information about the destination airport.

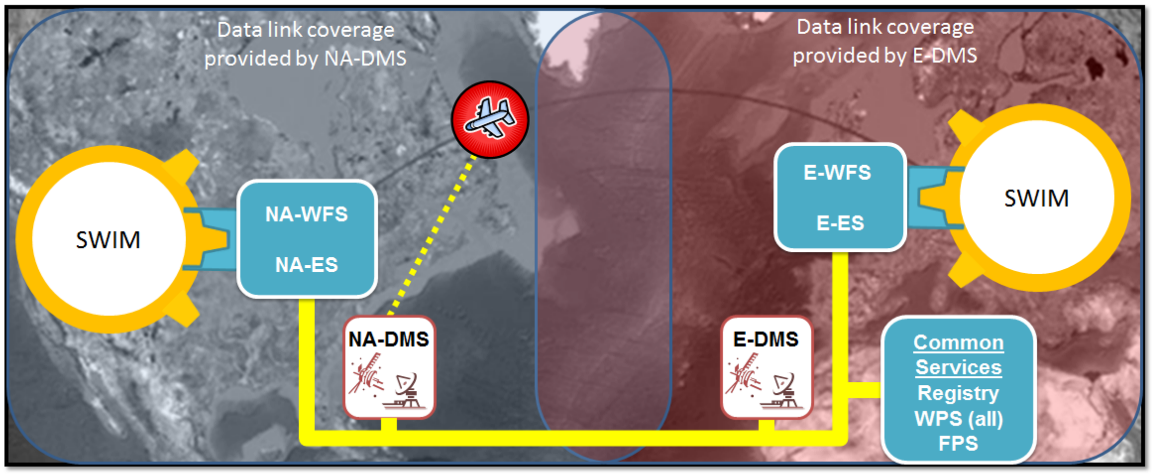

Data link coverage provided by NA-DMS
Data link coverage provided by E-DMS
SWIM
NA-WFS
NA-ES
NA-DMS
E-WFS
E-ES
SWIM
E-DMS
Common Services Registry WPS (all) FPS

Figure 21 OWS-9 Operational Scenario

10.2 Component assignments

Component identifier	**Provider**
Dispatch client NOTE: this client does not use the DMS(s) to communicate with the ground services.	Luciad
Aircraft client NOTE: here communication with the ground services is performed via the DMS(s).	Luciad
N-client (NOTAM generation client)	Luciad, using JMeter
DS-client (dispatch synchronization client)	ATM-TGS (JMeter as backup)
E-DMS	ATM-TGS (Harris DMS as backup)
E-ES	IDS (IfGI ES as backup)
E-WFS	Snowflake (COMSOFT WFS as backup)
ePIB client NOTE: this client does not use the DMS to communicate with the ground services.	Envitia
FPS	Luciad
GP-WPS (Geometry Processing WPS)	Luciad or 52N

NA-DMS	Harris (ATM-TGS DMS as backup)
NA-ES	IfGI (IDS ES as backup)
NA-WFS	COMSOFT (Snowflake WFS as backup)
Registry	Galdos
CCI-WPS (CCI Semantic Mediation WPS)	Envitia

10.3 Detailed Scenario

The following table describes the detailed scenario and associates key parts of the scenario with more detailed demonstration descriptions.

Scenario	**Main Demo**
Pre-Departure (Flight Planning) **OWS-9 Flight 324** is scheduled to depart from Airport Donlon (EADD) for Airport Chicago O'Hare (ORD). The usual alternate airports for the flight route are Reykjavik (RKV) (an airport close to the filed route of flight to be used in an emergency occurring during flight prior to the equal time point (ETP)) and Milwaukee International (MKE) (an alternate airport to be used in the event that a landing at ORD cannot be made). NOTE: the dispatcher already has an initial flight route because this is a common flight. So the demo can start showing the route.	
1. The dispatcher retrieves aeronautical information, including both static data and NOTAMs relevant to the route of flight, e.g. airspace restrictions.	The **dispatch client** queries the **CCI-WPS** for WFSs that serve aeronautical data corresponding to the dispatcher's interest. The dispatcher can enter a term using his/her own terminology. NOTE: the dispatch client could have also queried the Registry for WFSs that serve aeronautical information for a relevant spatial region along the flight route (100 mile buffer), though without semantic mediation of search terms. This is shown in the Registry mini demo. NOTE: when looking up services that provide specific data via the CCI-WPS, the search term "Air Traffic Control Preferred Routes" would for example query the AIXM feature types RouteDME, Route, RouteSegment and

Scenario	Main Demo
	Airspace. The **dispatch client** queries each of the returned **WFSs** for relevant AIXM feature data that intersects a spatial bounding box around the planned flight (200 mile buffer around the flight route). NOTE: for this scenario retrieval of airspace data as well as airport feature data (e.g. the addition of a crane at ORD) to visualize the updates is sufficient. NOTE: the dispatch client retrieves full AIXM feature data from the WFSs. In a future activity the Temporality Extension could be used to only retrieve feature data that is relevant to the time of flight, not the complete dynamic feature data).
2. The dispatcher also subscribes to receive updates from this data source.	(*narrative*)
3. The dispatcher reviews the information to determine whether the route needs to be amended. The dispatcher therefore initiates a check for conflicts, for example with relevant airspaces or at relevant airports.	The **dispatch client** uses the **GP-WPS** to identify the airspaces that spatially intersect (in 2D) the planned route. Two (or more) airspaces intersect the route – and are highlighted by the client. The **dispatch client** also computes (again using the **GP-WPS**) and highlights the actual intersection points. Then the **dispatch client** checks if one of these airspaces is planned to be active / in use during the time of the planned flight. It turns out that none of these airspaces are currently planned to be active when the flight is scheduled to cross it. NOTE: this is achieved by simulating the flight and visually checking if at the time that the flight crosses the airspace it is active / in use. The dispatcher also reviews information about the addition of a crane at ORD. The dispatcher decides that this update is non-critical for the flight.

Scenario	**Main Demo**
	NOTE: this update will also show up in the ePIB map.
4. The analysis of the aeronautical information showed that the flight route does not need to be amended. The dispatcher thus creates and files the flight plan with the ANSP.	(*narrative*)
5. The dispatcher then proceeds to create the Pre-Flight Information Briefing package. He generates the ePIB maps including airport images and DNOTAM overlays.	Using the **ePIB client** the dispatcher requests airport maps for the flight relevant airports (departure, destination, and alternate(s)). The ePIB client allows the user to specify an airport and search for DNOTAMs applicable to that airport. The user can then select the DNOTAMs of interest and request an airport map from the ePIB WPS. The resulting map shows a graphical representation of the selected DNOTAMs on top of an airport image. Using the **ePIB client** the dispatcher (*relatively quickly*) browses through the generated maps and stops at the ORD map, highlighting / zooming in to the crane NOTAM information displayed there.
6. The dispatcher then proceeds to configure the connection between the aircraft client and the DMS(s). NOTE: with information on the full flight plan the aircraft client could also configure the connection(s) automatically.	*(narrative) The dispatcher knows that there are two DMSs – the E-DMS and the NA-DMS – that satisfy the communication requirements of the flight. The E-DMS supports communication over Europe while the NA-DMS supports communication over North America. Both DMSs support required functionality but also modules that are optional for this flight.* The **dispatch client** retrieves information on module functionality (*as well as the spatial coverage of available data links*) supported by both the E-DMS and NA-DMS. The **dispatch client** then sets up the module options for both DMSs: - The following required functionality supported by both DMSs is enabled:

Scenario	**Main Demo**
	o basic filtering o reliable messaging For the NA-DMS he also enables: provenance tracking *(narrative) The dispatch client then loads these settings to the aircraft client which automatically uses them to initiate connections – first with the E-DMS and later on with the NA-DMS.*
7. The dispatcher turns on DMS synchronization in order to be notified of the data that the aircraft client receives.	The **DS-client** subscribes to the aircraft client (sessions) at the **E-DMS and NA-DMS**. NOTE: this will ensure that the DS-client receives copies of all messages that the two DMSs send to the aircraft client.
8. In addition, the dispatcher subscribes the aircraft client for aeronautical updates relevant for the flight.	Using stored filters the **dispatch client** creates subscriptions on behalf of the aircraft client at the **E-ES** and **NA-ES** to receive aeronautical information updates relevant to the flight. NOTE: requires that stored filters have been created at the ESs. NOTE: under the hood, the dispatch client requests the consumer reference endpoints for the aircraft client at both DMSs. These endpoints are required for creating subscriptions on behalf of the aircraft client, because the Event Service(s) will send notifications to these endpoints. Notification messages received by the DMSs at these endpoints can be delivered to the aircraft client. The dispatch client knows the Client ID with which to query the consumer reference endpoints at both DMSs. The aircraft client will automatically configure its subscriptions based upon which DMS sessions are actually enabled. NOTE: the E-DMS is used initially because it provides data link coverage for the first half of the flight (which starts in Europe). Before the aircraft leaves the area of the world where the E-DMS provides data link coverage, the aircraft client will need to switch communications to use the NA-DMS. More specifically, the following subscriptions are created at both Event Services:

Scenario	**Main Demo**
	NOTE: in an operational environment subscriptions would target much more information; the following subscriptions only cover the information updates relevant for the demonstration in OWS-9 a) subscription to updates for all Runway related events (not using the non-spatial feature query but using selective metadata retrieval to remove any feature and time slice metadata) NOTE: in this scenario this subscription is only relevant at the E-ES. b) subscription to updates for all AirportHeliport related events (not using the non-spatial feature query) NOTE: in this scenario this subscription is only relevant at the NA-ES. c) subscription to updates for all Taxiway events (using the non-spatial feature query in a 100 mile buffer around the flight route) NOTE: in this scenario this subscription is only relevant at the NA-ES.
9. The aircraft client will now automatically be notified of relevant information updates. Due to the dispatcher automatically receiving copies of such updates as well, the dispatcher cancels the subscriptions that he created for himself.	(*narrative*)
10. Finally, the dispatcher loads the Pre-Flight Information Briefing package onto the aircraft client. The package contains the ePIB data as well as relevant aeronautical data.	*(narrative)*
11. As part of his or her pre-flight preparation, the pilot reviews the data contained in the Pre-Flight Information Briefing package on the aircraft client. The dispatcher thereby provides the verbal part of	*(narrative)*

Scenario	**Main Demo**
the briefing. After the briefing the pilot heads towards the aircraft.	
Pilot User (Update for Alternate Airport) (*narrative) The pilot enters the aircraft after his briefing.*	
12. The pilot turns on his client to retrieve relevant information updates that were published since the briefing. He receives a NOTAM informing him that one of the EADD runways is temporarily closed due to removal of surface contamination (oil). Until the contamination is cleared, only one runway remains at EADD, which causes a slight departure delay.	The **aircraft client** enables the connection with the **E-DMS**. The NOTAM XML is shown by the **N-client** which briefly highlights that the NOTAM contains lots of metadata. The NOTAM information is sent (via the **N-client**) to the **E-ES** as a new tempdelta for the Runway. NOTE: this NOTAM is used to demonstrate the selective metadata retrieval of the E-ES. The **E-ES** matches the DNOTAM against its subscriptions (thereby applying selective metadata retrieval) and sends the (modified) DNOTAM to the **E-DMS**. The **E-DMS** filters the message and transmits the Runway DNOTAM to the **aircraft client**. The **aircraft client** displays the DNOTAM and the pilot inspects it.
13. The dispatcher receives the NOTAM as a synchronization update from the DMS. As only a slight delay is caused, no further action is required by the dispatcher.	The **E-DMS**, after having received the acknowledgement from the aircraft client that the filtered Runway DNOTAM was successfully transmitted, sends a copy of the filtered Runway DNOTAM to the **DS-client**. The **DS-client** shows the incoming message, especially highlighting the reduced size (due to filtering performed by the E-DMS and selective metadata retrieval performed by the E-ES).

Scenario	Main Demo
14. After a slight delay the ground controller at EADD airport clears **OWS-9 Flight 324** to taxi for departure.	(*narrative*)
Transition to a different DMS	
15. While the aircraft approaches Greenland, it switches ground service communications to use a DMS that provides data link coverage for the rest of the flight.	The aircraft leaves the area of the world where the E-DMS provides data link coverage, thus the **aircraft client** switches communications to use the NA-DMS. The **aircraft client** therefore disables the connection with the **E-DMS** and enables the connection with the **NA-DMS**. NOTE: the aircraft client automatically updates its subscriptions to be routed via the NA-DMS instead of the E-DMS. It therefore creates all relevant subscriptions (see scenario step 8) at the E-ES and NA-ES using the aircraft client consumer reference at the NA-DMS and cancels all subscriptions that used the consumer reference at the E-DMS. NOTE: the client only needs to re-route the notifications from the Event Services to the NA-DMS. Because Event Services currently do not support the update of the notification target, the client has to create new subscriptions that use the NA-DMS consumer reference and cancel the old subscriptions. Developing ways to support simple re-routing of notifications is future work. From now on the aircraft client communicates exclusively via the NA-DMS.
Transition from departure data source to destination data source	
16. While on its way to its destination **OWS-9 Flight 324** is connected via the DMS to the ground services of both the departure ANSP data source in the European SWIM environment and the arrival ANSP data source in the U.S. SWIM environment. In both cases the ground services support interoperable OGC web service interfaces which allow the aircraft client to communicate with all	(*narrative*)

Scenario	**Main Demo**
ground services in a uniform way.	
En-Route Data Exchange	
17. A NOTAM is issued for MKE, informing that one of its taxiways is temporarily closed due to maintenance. The pilot receives and inspects the update. The taxiway closure does not prevent a potential landing at the alternate airport. Because OWS-9 Flight 324 is still on its way to ORD as planned, the pilot files this update away for now, knowing that he can bring it back on in preparation for taxiing at MKE in case that he needs to amend the route.	The NOTAM information is sent (via the **N-client**) to the **NA-ES** as a new tempdelta for the Taxiway. The **NA-ES** matches the DNOTAM against its subscriptions (applying non-spatial feature query) and sends the DNOTAM to the **NA-DMS**. The **NA-DMS** filters the message and transmits it to the **aircraft client**. The **aircraft client** displays the DNOTAM and the pilot inspects it.
Arrival Planning Data Exchange	
18. While approaching North America, the data link to the aircraft is broken.	The **aircraft client** is disconnected from the network.
19. During the time that the aircraft client is not connected to the ground services, a critical NOTAM is issued for ORD, informing that ORD is closed until further notice. Fires in close proximity to ORD are producing extensive smoke over the airport.	The NOTAM information is sent (via the **N-client**) to the **NA-ES** as a new tempdelta for the ORD AirportHeliport. The **NA-ES** matches it against its subscriptions and sends the DNOTAM to the **NA-DMS**. The **NA-DMS** filters the message but cannot transmit it to the aircraft client because the aircraft client is not connected to the network. The NA-DMS therefore stores the filtered message and tries to re-send it according to its reliable messaging settings.
20. Later on the data link to the aircraft is re-established. The pilot receives the NOTAM and inspects it.	The **aircraft client** connects to the network. The **NA-DMS** is able to successfully transmit the AirportHeliport DNOTAM to the **aircraft client**.

Scenario	Main Demo
	The **aircraft client** displays the DNOTAM and the pilot inspects it.
21. Because the destination airport is closed until further notice the pilot decides to fly to the alternate airport. The pilot requests redirection of the flight to MKE via voice and receives clearance from ATC.	(*narrative*)
22. The pilot checks if there are any conflicts with airspace limitations on the new route.	The pilot uses the **aircraft client** to create a new route from the current location of OWS-9 Flight 324 to MKE. The **aircraft client** performs an altitude query at the **NA-WFS** to retrieve detailed information – especially updates that may have occurred since aeronautical data was loaded on the aircraft client by the dispatcher - on the airspaces that actually intersect the new flight route in 2.5D and are active, available for activation or in use during the remaining flight time. A single airspace is returned. The **aircraft client** displays this airspace in 3D and the pilot inspects if the airspace is in use or going to be active for the time that the flight is scheduled to cross the airspace. It turns out that this is not the case.
OWS-9 Flight 324 continues to MKE and lands without incident.	(*narrative*)

10.4 Clarifications

- In this demo the aircraft client actually always communicates via the same network. Its IP address does not change. Only its connection to the network may be broken, which is handled by DMS via the reliable messaging functionality.
- In this demo the DMS does not actually manage different data links. All mentioning of data link change is pure narration. Actual data link management by DMS is a future work item.

10.5 Aspects Not Covered by Main Scenario

The aspects described in the following subsections were removed from the main scenario because the actions they define are considered to have already been performed. In other words, the results of these actions are preconditions for the main scenario.

11 Lessons Learned

11.1 Inconsistent Data Publication Models

11.1.1 Problem Statement and Description

As OGC standards, such as AIXM 5.1, are being increasing deployed in operational production environments by data publishers, differing interpretations of the standards become an obstacle to interoperability. It appears that the data publishers are interpreting the standards to create their own publication mechanisms and the complexity of the standards and the nuances in this interpretation lead to incompatibility among the data publishers. A few examples illustrate the point:

1. A data publisher exchanges its data as AIXM 5.1 to a wide range of stakeholders and assigns each feature a UUID. The consumers of that data assign a new UUID to satisfy their internal data manipulation needs. The results are multiple sets of UUIDs for describing the same features. As data publishers continue to expand their own data exchanges with other stakeholders, the probability that a consumer may end up with multiple instances of the same feature data in its database. In the near term, UUID look-up tables can resolve the issue but other long-term mechanisms need to be put into place.
2. A related but different example may be found in a comparison of data sets from various data publishers. In this instance, different interpretations of the XLink href's can render the data sets incompatible. The error is due to one publisher encoding the value of the XLink href not with the value assigned in the gml:id but the gml:identifier UUID plus a prefix urn:uuid. In order to use the data, another publisher would have to strip out the urn:uuid: prefix and then use the UUID value to generate a join back to the relevant feature so it could be published with a valid XLink href value. To further illuminate the problem, the Xlinks used in the data follow the specification of "Abstract references" in

   http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_Feature_Identification_and_Reference-1.0.pdf.

   Section 3.4.1 which points to the gml:identifier, not the gml:id (though there is a recommendation on how the gml:id should be chosen given the gml:identifier of the feature). The result of interpreting the nuances of the standard is incompatible data sets.

11.1.2 Recommendations

The evolution of the business needs that drive technical standards continually requires the technical community to make choices about customization and tailored applications that manipulate data flowing from a standards-based web service. While these efforts satisfy customer requirements, they tend to create intrinsic incompatibilities when the data is stored in a standards-based data model and then redistributed. The challenge for standards bodies such as the OGC is to guide the standards working groups into creatively resolving incompatibilities among implementations without making the standard more complex than absolutely necessary. Following this theme, it is recommended that OGC expand and formalize its initiatives designed to reduce the number of standards interpretation incompatibility issues, such as:

- Conformance testing - OWS-9 participants are building and/or testing Compliance and Interoperability Test (CITE) reference implementations for the following services:
    - SPS 2.0.
    - WMS 1.3 client and server.
    - WFS 2.0.
    - WFS EO 1.0.

  As these components are tested, validated, and incorporated into implementation integration testing, it will be easier to determine whether a service really supports a certain set of functionality. Then the required set of functionality can be stated more precisely in procurement requirements and service providers will be better able to prove up-front which functionality they support.

- Governance and certification – OGC continues to maintain significant standards harmonization initiatives to help ensure the cross-pollination of good ideas across its many standards working groups. Expansion and formalization of these initiatives in conjunction with an active governance and certification process would help reduce the variations in standards interpretations with the result of improved interoperability.

12 Technical Challenges / Future Work

- Common Definition of WPS Process Definitions - When designing a WPS ProcessDefinition the format for inputs and outputs can be defined in a lax way by pointing to a commonly available XML Schema. By implication, a WPS implementing such ProcessDefinition must support all available root elements as inputs/outputs. To enable true interoperability among different WPS instances, one should be as exact as possible defining the inputs/outputs. Within OWS-9, the decision was to define a new schema for the result output. The schema actually subsets GML 3.2 to only allow one element of gml:MultiGeometry. It should be considered in future revisions of the WPS standard if a sub-setting of an XML schema (e.g. via a list of supported QNames) could be supported within a ProcessDefinition.

- Stored Filters – Currently, the design of Stored Filter only takes the wsnt:Filter part of an Event Service subscription into account. In order to provide other aspects of a subscription through a stored mechanism, a more general approach should be designed, namely Stored Subscriptions. Thus it would be possible to define policies and projection clauses in addition to the filter part.

- AIXM Features as Geometry Operands – It has been identified within this test bed that an AIXM might have multiple geometries defined. The current design does not take such cases into consideration. Future work should focus on this by introducing, for instance, a "scenario" parameter which defines how the geometry should be computed (e.g. simple locating vs. complex visualization of a feature). Such work should also be aligned with Geometry Processing work areas.

- Event Service Update Intervals – The design of Update Intervals already provides encodings to define the behavior on how to treat non-related feature events as well as the detection of duplicate events. Nevertheless, the development of algorithms for these cases is a complex venture and thus has been postponed. Future work on this feature should focus on the design and implementation of such algorithms.

- Calculation of the airspace extent in 2.5D - The correct calculation of composite airspaces, which in turn may be composed of other airspaces and which may in turn include references to geographical borders, turned out to be a challenging task, especially when taking the vertical dimension into account. As the specification is a mixture of AIXM and GML, a dedicated algorithm had to be developed, combining 2D calculations on the earth's ellipsoid with AIXM specific properties for the vertical extent and composition operators.

- Feature Portrayal Service Performance - One important and reoccurring challenge when working with an FPS in general is to get optimal performance. The high flexibility of being able to submit any kind of feature data and styling information comes with difficulties to achieve this. The AIXM format is verbose, so large amounts of bandwidth can be needed to provide the FPS with the necessary feature data. Reducing the amount of data (e.g. filtering properties to the bare minimum that is actually needed by the FPS) and/or applying compression are possible ways to resolve this. Additionally, a SE style can also have an impact on performance: complex styles with lots of rules / filter combinations can be defined that can slow down their interpretation and therefore impact rendering performance.

- Extend distance parameter of DWithin and Beyond operators - The Filter Encoding Specification defines two spatial operators with a distance parameter:

DWithin and Beyond. It is not explicitly mentioned what the distance metric is, as there are several options: distance in space, geodesic distance, distance on terrain, to name a few. One work item is to clarify this and find a sensible default. A use case in OWS-9 revealed that more flexibility is needed. When querying for airspaces that lie in a certain distance of a flight route, it must be possible to specify separate distance parameters for the horizontal and vertical axis (altitude) at least. Future test beds should elaborate on this issue to find a way to extend the filter encoding standard or develop work-arounds.

- Interoperability among ANSPs - The key challenge in the development of the Data Management Service (DMS) was creating a solution that can fully communicate with both the North American and European System Wide Information Management (SWIM) environments. Although the overarching system concepts for SWIM are the same in both North American and Europe, the actual SWIM implementations currently employ different technical standards for their communication interfaces. During the development of DMS for OWS-9, the web service technical standard is used as the communication protocol between aircraft, DMS, and grounds services. The use of web service as a communication protocol works well as connection to European SWIM ground services. Future OGC activity should focus on the integration of Message-Oriented Middleware (MOM), which is the communication protocol primarily used by North American SWIM ground services. The extent of the technical gap between what was created in OWS-9 and what needs to be developed for integration of FAA SWIM ground services, was documented by OWS-9 in the SWIM Compliance Assessment document. The creation of a solution that bridges the gap between North American and European SWIM messaging protocol is a strong candidate for future OGC work.

13 Accomplishments

13.1 Web Feature Service

- Data was loaded, validated, transformed and corrected from multiple sources, including splitting the data into two WFS instances, one for North American data and one for Middle-Northern Europe data.

- Calculation of airspace extents in full 2.5D, including composite airspaces (unions, intersections and subtractions) and support for altitude queries and calculation of extents of non-spatial feature types (runways, taxiways and aprons) and support for spatial filtering of non-spatial feature types.

13.2 Registry Service

The OWS-9 Aviation Registry was used to host resources shared between multiple OGC services (e.g. ISO metadata pertaining to datasets common to multiple WFS implementations (Snowflake, Comsoft), Styling information shared between multiple Feature Portrayal Services (FPS) implementations (Envitia, Luciad), and ISO Service metadata) describing the Services deployed in OWS-9. The efficient retrieval of metadata task was also implemented using the Aviation Registry by designing queries to retrieve only the relevant excerpts of the metadata desired by clients. The participants used Insert/Update/Get transactions to create User accounts, publish common resources, and efficiently retrieve resources using complex filtered queries. No functional or performance issues were encountered during the OWS-9 initiative.

13.3 Event Service

Compared to the ES used in the previous OGC test beds, major improvements have been implemented during OWS-9, particularly features summarized under the term “Advanced filtering functionality.” These included “Event Service Update Intervals”, “Stored Filters”, “AIXM Features as Geometry Operands”, “Spatial Filtering of Non-spatial Features”, “Simple Altitude Queries” as well as “Selective Metadata Retrieval”.

13.4 Web Processing Service

Implementation of WPS profiles to support ePIB map generation and geometry processing, including calculation of topological relations between two AIXM 5.1 features in ellipsoidal space. Supported topological relations: INTERSECTS, DISJOINT, EQUALS, TOUCHES, CROSSES, OVERLAPS, CONTAINS, WITHIN, COVERS, COVERED_BY.

13.5 Data Management Service

Within the OWS-9, a set of functionalities to provide reliable and efficient management of communications between aircraft and services located on the ground were implemented and tested:

- DMS service discovery; which mainly concerns the process of finding and setting the processing options to be done at the DMS by the aircraft and dispatch client.
- DMS basic pass-through; which handles the forwarding of request/response and notifications between aircraft client and OGC web services.
- Reliable messaging.
- Data compression / expansion
- Data filtering.

- Dispatch synchronization.

13.6 Aviation Client

Aviation clients were implemented and tested that provide map-centric displays with intuitive user interface giving access to data from entities such as Web Feature Service, Event Service, and the Data Management Service. The continuing evolution of these clients provided rich set of capabilities and features that help to demonstrate the OWS-9 Aviation scenario and to perform testing and integration with a wide variety of service components.

Annex A - SWIM Compliance Assessment

Introduction

The purpose of the SWIM Compliance Assessment is to provide information to the OGC and FAA on:

- The alignment of SWIM compliance requirements and OGC standards.
- The applicability of a SWIM requirement to OWS-9 services, clients, or components.

In recognition that some SWIM compliance requirements appear to be focused on FAA programs that operate in the SWIM environment, we need to envision a future where there is a bi-directional flow of information between OGC services and FAA programs. From this perspective, we would appreciate your input on SWIM requirements that would not be applicable to web services data exchanges involving external entities.

Completing the Matrix

Please enter a % compliance value from 0 to 100 in the Self-Assessment Column.

- Make your best subjective estimate of the % compliance.
- It is acceptable for you to enter less than 100% in the Self-Assessment column if your service, client, or component doesn't clearly meet the requirement due to the maturity of your service, client, or component solution.
- It is acceptable to enter N/A in the Self-Assessment Column with an explanation for why that requirement doesn't apply to the project or is not being complied with.

The Notes column of the Assessment Table provides you with the opportunity to:

- Provide the rationale for your assessment.
- Comment on the applicability of the SWIM requirement to your service, client, or component.

File Naming Convention

Please add a hyphen to the name of this file and then enter your company name and then upload to the SWIM Compliance Analysis folder on the OGC portal at OGC Web Services, Phase 9* / Thread 1 - Aviation / SWIM Compliance Analysis. For example – SWIM Compliance Assessment-SpeedSquared.docx

Assessment Identification:

Company: ____________________________

Point of Contact (Name and E-mail): ____

OWS-9 Service / Client / Component: ___________________

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
4.1 Technology Acquisition		N/A	Applies to technology acquisition officials
SWIM-SC-0001	SWIM services SHALL be implemented using standardized service container products and product versions that are specified in the SWIM Product Standardization (SPS) document and provided by the SWIM Program Office and managed as part of the SWIM COTS Product Repository (ref: SWIM COTS Product Management Plan).		
SWIM-SC-0002	Service implementing programs SHALL acquire approved Commercial-Off-The-Shelf (COTS)		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
	products only through the SWIM COTS Product Repository (SCPR) in accordance with the SWIM COTS Product Management Plan.		
4.2 Interoperability, Reuse, and Standards			
4.2.1 Use of Open Standards			
SWIM-SC-0010	SWIM services SHOULD apply open data standards, schemas, and message interfaces where possible, instead of creating new schema definitions.		
SWIM-SC-0020	SWIM services SHALL provide an interface that supports one or more of the following message format, message protocol, and transport protocol combinations: • SOAP-over-HTTP/HTTPS • XML-over-HTTP/HTTPS, inclusive of the Representative State Transfer (REST) interface pattern • SOAP-over-JMS • XML-over-JMS		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
SWIM-SC-0021	SWIM services MAY use HTTPS or TLS as the transport protocol in place of HTTP or TCP for service interface interactions.		
4.2.3 SOAP Message Processing			
SWIM-SC-0030	SOAP Messages in the SWIM environment SHALL be processed using the FUSE Service Framework library.		
4.2.4 SOAP Messaging WS-I Compliance			
SWIM-SC-0040	SWIM Service endpoints SHALL meet the messaging compliance requirements of the SWIM Interoperability Basic Profile (a SWIM-annotated version of the Web Services Interoperability Basic Profile, Section 3).		
4.2.5 Binary Attachments in SOAP Messages			
SWIM-SC-0045	Any binary attachment that is sent with a SOAP message SHALL be attached and processed using the Message Transmission Optimization Mechanism (MTOM), in accordance with the SWIM Governance Plan.		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
4.2.6 Data Retrieval Protocol			
SWIM-SC-0048	Service providers MAY use transport protocols and message formats other than those defined in SWIM-SC-0020 for binary and large file data delivery and retrieval. For example: FTP, SMTP, or SCP may be applied to delivery large binary files. The alternate protocol SHALL NOT be used for service requests or notifications, but MAY be used for content delivery.		
4.2.7 JMS Provider Standardization			
SWIM-SC-0050	SWIM JMS message producers SHALL use FUSE Message Broker as the JMS Provider for JMS destinations		
SWIM-SC-0051	SWIM JMS clients MAY institute a JMS bridge between FUSE Message Broker and the service provider's message-oriented-middleware (MoM).		
SWIM-SC-0052	SWIM JMS Message Brokers SHALL be configured to support the OpenWire protocol over TCP or SSL.		
SWIM-SC-0053	SWIM JMS Message Brokers MAY be configured to support the Simple Text Object		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
	Messaging Protocol (STOMP) protocol over HTTP or HTTPS		
4.2.8 JMS Destination Names			
SWIM-SC-0055	JMS destination names SHALL include the FAA Business Context Identifier (FBCI) of a SWIM-registered namespace as the first prefix for the destination name, as described by [STD063]. For example, the namespace: urn:us:gov:dot:faa:AviationSafety may correspond to the JMS destination name: “AviationSafety.topic.IncidentReports”.		
4.2.9 JMS Message Type			
SWIM-SC-0060	Messages sent using JMS SHALL use an XML format, and MAY use the SOAP message format.		
SWIM-SC-0061	SOAP and XML messages that are sent over JMS SHALL use the JMS TextMessage type.		
4.3 Registry / Repository			
4.3.1 Interface Discoverability			

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
SWIM-SC-0070	WSDL documents corresponding to SWIM service endpoints SHALL be registered with the SWIM Service Registry / Repository in accordance with TBD. One WSDL document may describe many endpoints.		
SWIM-SC-0071	SWIM service WSDL documents SHALL NOT be published through any mechanism other than the SWIM Service Registry / Repository. Those mechanisms prohibited include, but are not limited to direct publishing using HTTP from the service provider site.		
4.3.2 Interface Categorization			
SWIM-SC-0080	SWIM services SHALL be categorized in the SWIM Service Registry/Repository as described in FAA-STD-064 and FAA-STD-066.		
SWIM-SC-0082	XSD schemas that define the structure for SWIM service messages SHALL be categorized in the SWIM Service Registry/Repository using SWIM service taxonomy categories as described in TBD.		
4.4 Namespace and Schema			

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
SWIM-SC-0090	SWIM service WSDL documents SHALL define services within a namespace that has been registered by the service provider in the FAA Data Registry (FDR).		
SWIM-SC-0091	SWIM service message schemas SHALL use namespaces that have been registered in the FAA Data Registry (FDR).		
4.5 Service Interface Design			
SWIM-SC-0100	SWIM service interfaces SHALL be described by a Web Service Definition Language (WSDL) v2.0 document.		
SWIM-SC-0101	SWIM service interfaces MAY be described by a WSDL v1.1 document.		
SWIM-SC-0102	The message content that may be sent or received by a SWIM service SHALL be described by one or more XML Schema Definition (XSD) documents.		
SWIM-SC-0103	The message content schema for messages that may be sent or received by a SWIM service SHALL NOT be defined in the WSDL		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
	document, and SHALL be in a separate XSD.		
SWIM-SC-0104	SWIM services shall be described by an FAA Web Service Definition Document (WSDD) in accordance with STD065, P*reparation of Web Service Description Documents*.		
4.5.2 Service Interface WS-I Compliance			
SWIM-SC-0110	SWIM service WSDL interface descriptions SHALL be compliant with the Service Description requirements defined in the SWIM Interoperability Basic Profile (a SWIM-annotated version of the Web Services Interoperability Basic Profile v1.2, section 4).		
4.6 Information Security			
SWIM-SC-0120	SWIM services shall implement security consistent with NIST Special Publication 800-95 Guide to Secure Web Services [NIST800-95].		
SWIM-SC-0121	SWIM services shall be compliant with the requirements defined in the SWIM Interoperability Basic Security Profile (a SWIM-annotated version of the Web Services		

Reference	Requirement	Self-Assessment (% compliance)	Notes / Comments
	Interoperability Basic Security Profile).		
4.7 Service Management			
SWIM-SC-0130	SWIM services SHALL use Java Management Extensions (JMX) for management and monitoring services at runtime.		

Annex B - Extended Event Service XML Schema

Stored Filter Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/es-sf/0.0"
xmlns:essf="http://www.opengis.net/es-sf/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified" xml:lang="en">
    <!-- REQUESTS -->
    <xs:element name="DescribeStoredFilter">
        <xs:annotation>
            <xs:documentation>Used to request detailed stored filter description.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="essf:StoredFilterID"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="StoredFilterID" type="xs:token">
        <xs:annotation>
            <xs:documentation>Unique ID for every available stored filter.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="DescribeStoredFilterResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="essf:StoredFilterDescription"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ListStoredFilters">
        <xs:annotation>
            <xs:documentation>Used to request the available stored filter IDs.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType/>
    </xs:element>
    <xs:element name="ListStoredFiltersResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="essf:StoredFilterID" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- DESCRIPTION -->
    <xs:element name="StoredFilterDescription" type="essf:StoredFilterDescriptionType"/>
    <xs:complexType name="StoredFilterDescriptionType">
        <xs:sequence>
            <xs:element ref="essf:Title"/>
            <xs:element ref="essf:Abstract"/>
            <xs:element ref="essf:Parameter" minOccurs="0"/>
            <xs:element ref="essf:FilterExpressionText" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>ID for this stored filter (as provided in the
                    capabilities contents).</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Abstract" type="xs:string"/>
    <xs:element name="Parameter">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="essf:Abstract"/>
            </xs:sequence>
```

```
                <xs:attribute name="name" type="xs:string" use="required">
                    <xs:annotation>
                        <xs:documentation>Local name for this parameter.</xs:documentation>
                    </xs:annotation>
                </xs:attribute>
                <xs:attribute name="type" type="xs:string" use="required">
                    <xs:annotation>
                        <xs:documentation>An xml type with namespace prefix. The prefix
                            must be defined within an xml instance.</xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
        <xs:element name="FilterExpressionText">
            <xs:annotation>
                <xs:documentation>This holds the actual filter expression. Due to the
                    fact that the markup
                    probably has placeholders for the parameters, it is useful to provide
the
                    expression as
                    escaped XML (e.g. using CDATA or escaped characters).</xs:documentation>
            </xs:annotation>
            <xs:complexType mixed="true">
                <xs:choice>
                    <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:any namespace="##targetNamespace" processContents="skip"
minOccurs="0" maxOccurs="unbounded"/>
                </xs:choice>
                <xs:attribute name="language" type="xs:anyURI" use="required"/>
                <xs:attribute name="isPrivate" type="xs:boolean" default="false"/>
            </xs:complexType>
        </xs:element>
        <!-- SUBSCRIPTION -->
        <xs:element name="StoredFilterSubscription" type="essf:StoredFilterSubscription"/>
        <xs:complexType name="StoredFilterSubscription">
            <xs:sequence>
                <xs:element ref="essf:ParameterValue" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required">
                <xs:annotation>
                    <xs:documentation>ID for this stored filter (as provided in the
                        capabilities contents).</xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:complexType>
        <xs:element name="ParameterValue">
            <xs:complexType>
                <xs:sequence>
                    <xs:any processContents="lax"/>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" use="required"/>
            </xs:complexType>
        </xs:element>
        <!-- CAPABILITIES -->
        <xs:element name="StoredFilterCapabilities"
type="essf:StoredFilterCapabilitiesType"/>
        <xs:complexType name="StoredFilterCapabilitiesType">
            <xs:sequence>
                <xs:element name="SupportedOperations">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="ListStoredFilters">
                                <xs:complexType/>
                            </xs:element>
                            <xs:element name="DescribeStoredFilter">
                                <xs:complexType/>
                            </xs:element>
                            <xs:element name="CreateStoredFilter" minOccurs="0"
maxOccurs="1">
                                <xs:complexType/>
                            </xs:element>
                            <xs:element name="RemoveStoredFilter" minOccurs="0"
maxOccurs="1">
                                <xs:complexType/>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Update Intervals Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:essp="http://www.opengis.net/es-sp/0.0"
targetNamespace="http://www.opengis.net/es-sp/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified" xml:lang="en">
    <!-- Subscription Policy Elements -->
    <xs:element name="EventServiceSubscriptionPolicy">
        <xs:annotation>
            <xs:documentation>Base element of an Event Service Subscription
                policy.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="UpdateInterval" type="essp:UpdateIntervalType"
minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Specify an update interval for a
subscription.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- Types -->
    <xs:complexType name="UpdateIntervalType">
        <xs:sequence>
            <xs:element name="IntervalDuration" type="xs:duration"/>
            <xs:element name="DisseminationMethod">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="batching">
                            <xs:annotation>
                                <xs:documentation>
                                    Use if all events occurred within the update
interval should be

                                    disseminated in one aggregated message.
                                </xs:documentation>
                            </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="latest">
                            <xs:annotation>
                                <xs:documentation>
                                    Use if only the latest event within the update
interval should

                                    be disseminated.
                                </xs:documentation>
                            </xs:annotation>
                        </xs:enumeration>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="NonRelatedEventTreatment" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="ignore">
                            <xs:annotation>
                                <xs:documentation>
                                    Ignore non-related (different features) events
within the update

                                    interval.
                                    When using "latest" as DisseminationMethod only the
actual latest

                                    event for the subscription regardless of
                                    the associated feature will be provided
                                </xs:documentation>
                            </xs:annotation>
                        </xs:enumeration>
```

```
                              <xs:enumeration value="separate">
                                  <xs:annotation>
                                      <xs:documentation>
                                          The service should identify related events (e.g.
for different
                                          features) and provide messages for
                                          all features within the update interval.
                                      </xs:documentation>
                                  </xs:annotation>
                              </xs:enumeration>
                          </xs:restriction>
                      </xs:simpleType>
                  </xs:element>
              </xs:sequence>
          </xs:complexType>
          <!-- CAPABILITIES -->
          <xs:element name="SubscriptionPolicyCapabilities"
type="essp:SubscriptionPolicyCapabilitiesType"/>
          <xs:complexType name="SubscriptionPolicyCapabilitiesType">
              <xs:sequence>
                  <xs:element name="SupportedPolicies">
                      <xs:complexType>
                          <xs:sequence>
                              <xs:element name="UpdateIntervalPolicy">
                                  <xs:complexType>
                                      <xs:sequence>
                                          <xs:element name="BatchingSupported"
type="xs:boolean"/>
                                          <xs:element
name="NonRelatedEventTreatmentSupported" type="xs:boolean"/>
                                      </xs:sequence>
                                  </xs:complexType>
                              </xs:element>
                          </xs:sequence>
                      </xs:complexType>
                  </xs:element>
              </xs:sequence>
          </xs:complexType>
          <!-- SYSTEM MESSAGES -->
          <xs:element name="NoNewMessages">
              <xs:complexType>
                  <xs:attribute name="currentTime" type="xs:dateTime"/>
              </xs:complexType>
          </xs:element>
</xs:schema>
```

FilterWithProjectionClause Schema

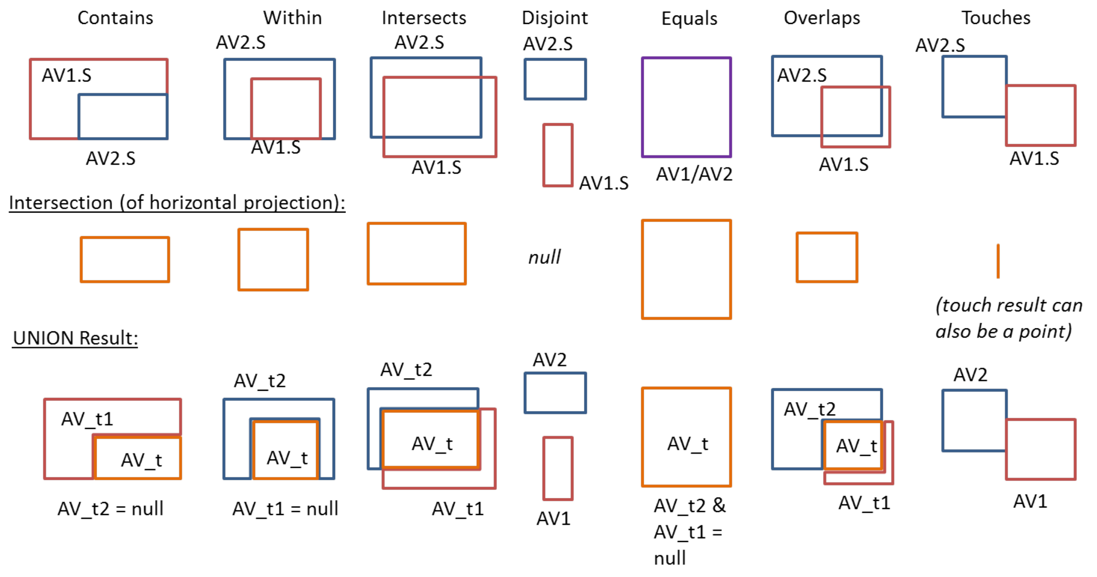
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fes="http://www.opengis.net/fes/2.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:es-pc="http://www.opengis.net/es-pc/0.0"
targetNamespace="http://www.opengis.net/es-pc/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:import namespace="http://www.opengis.net/fes/2.0"
schemaLocation="http://schemas.opengis.net/filter/2.0/filter.xsd"/>
    <!--FilterType EXTENSION -->
    <xs:element name="ExtendedProjectionClauseFilter" type="es-
pc:ExtendedProjectionClauseFilterType">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ExtendedProjectionClauseFilterType">
        <xs:complexContent>
            <xs:extension base="fes:FilterType">
                <xs:sequence>
                    <xs:element ref="fes:AbstractProjectionClause" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- New Element with Filter and ProjectionClause -->
    <xs:element name="FilterWithProjectionClause" type="es-
pc:FilterWithProjectionClauseType">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="FilterWithProjectionClauseType">
        <xs:sequence>
            <xs:element ref="fes:AbstractProjectionClause" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element ref="fes:Filter" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Annex C - Airspace Geometry Computation

Geometric Operation / Aggregation Type: **UNION**
Surface$_1$ UNION Surface$_2$

```
// AV1 is part of result set – the first AirspaceVolume is part of this set by default afaics
If(AV1.S.DISJOINT(AV2.S) OR AV1.S.TOUCHES(AV2.S))
  include AV2 in result set
Else
  remove AV1 from result set // as it intersects with AV2 and thus needs to be split up
  AV_t,AV_t1,AV_t2; // variables to capture temporary AirspaceVolumes
  AV_t.setHorizProj(AV1.S.intersection(AV2.S))
  AV_t.setAltitude(UNION,AV1,AV2) // result is one or two AirspaceVolumes
  include AV_t in result set
  AV_t1.setHorizProj(AV1.S.difference(AV_t.S))
  AV_t2.setHorizProj(AV2.S.difference(AV_t.S))
  if (AV_t1.S != null) // then AV1.S is not „within" AV2.S
    AV_t1.setAltitude(AV1.altitude)
    include AV_t1 in result set
  if (AV_t2.S != null) // then AV1.S does not „contain" AV2.S
      AV_t2.setAltitude(AV2.altitude)
      include AV_t2 in result set
    // if AV1.S.EQUALS(AV2.S) then just AV_t is included in result set
```

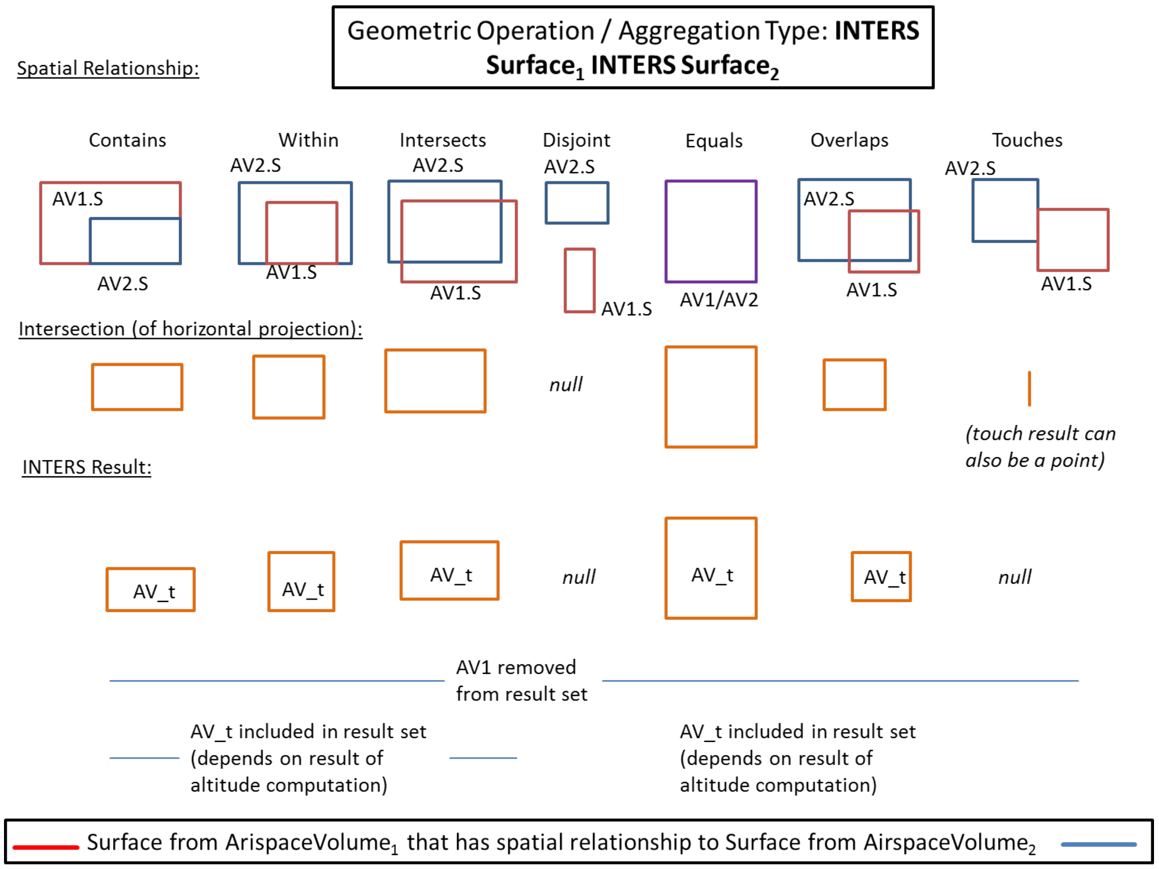
Geometric Operation / Aggregation Type: **UNION**
Surface$_1$ UNION Surface$_2$



🟥 Surface from ArispaceVolume$_1$ that has spatial relationship to Surface from AirspaceVolume$_2$ 🟦

Geometric Operation / Aggregation Type: **INTERS**
Surface$_1$ INTERS Surface$_2$

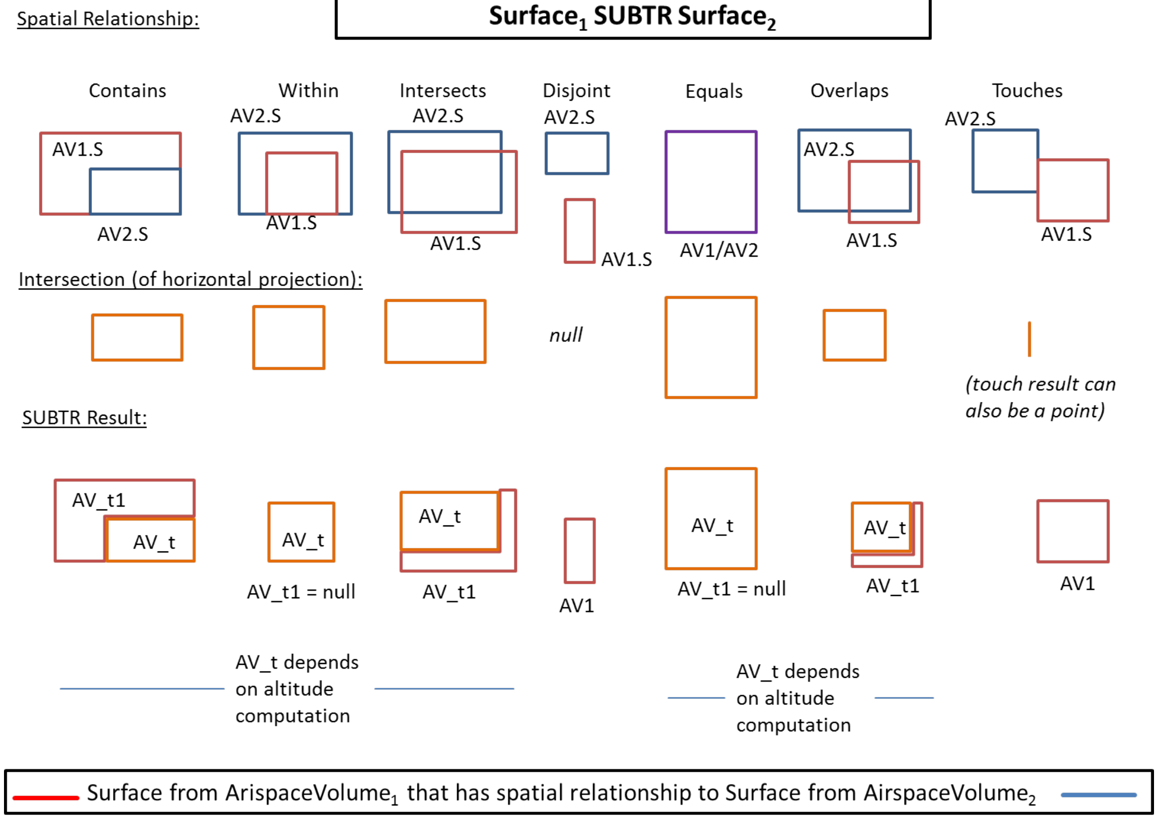
```
// AV1 is part of result set – the first AirspaceVolume is part of this set by default afaics
If(AV1.S.DISJOINT(AV2.S) OR AV1.S.TOUCHES(AV2.S))
  remove AV1 from result set // would result in empty result set (TBC)
Else
  remove AV1 from result set // only intersection is of interest, computed in following steps
  AV_t; // variable to capture temporary AirspaceVolumes
  AV_t.setHorizProj(AV1.S.intersection(AV2.S))
  AV_t.setAltitude(INTERS,AV1,AV2) // results in zero or one AirspaceVolume
  If (AV_t != null)
    include AV_t in result set
```


Geometric Operation / Aggregation Type: INTERS
Surface1 INTERS Surface2
Spatial Relationship:
Contains
Within
Intersects
Disjoint
Equals
Overlaps
Touches
AV1.S
AV2.S
AV1/AV2
Intersection (of horizontal projection):
null
(touch result can also be a point)
INTERS Result:
AV_t
null
AV1 removed from result set
AV_t included in result set (depends on result of altitude computation)
AV_t included in result set (depends on result of altitude computation)
Surface from ArispaceVolume1 that has spatial relationship to Surface from AirspaceVolume2

Geometric Operation / Aggregation Type: **SUBTR**
Surface$_1$ SUBTR Surface$_2$

```
// AV1 is part of result set – the first AirspaceVolume is part of this set by default afaics
If(AV1.S.DISJOINT(AV2.S) OR AV1.S.TOUCHES(AV2.S))
  // nothing to do, keep AV1 in result set
Else
  remove AV1 from result set // as it intersects with AV2 and thus needs to be split up
  AV_t,AV_t1; // variables to capture temporary AirspaceVolumes
  AV_t.setHorizProj(AV1.S.intersection(AV2.S) // intersection of horizontalProjection
  AV_t.setAltitude(SUBTR,AV1,AV2) // result is zero, one or two AirspaceVolumes
  if (AV_t != null)
    include AV_t in result set
  AV_t1.setHorizProj(AV1.S.difference(AV2.S))
  if (AV_t1.S == null)
    // AV1.S.EQUALS(AV2.S) OR AV1.S.WITHIN(AV2.S)
  else
    AV_t1.setAltitude(AV1.altitude)
    //resulting AirspaceVolume has new shape but altitude from AV1
    include AV_t1 in result set
```

Geometric Operation / Aggregation Type: **SUBTR**
Surface$_1$ SUBTR Surface$_2$



Spatial Relationship:
Contains
Within
Intersects
Disjoint
Equals
Overlaps
Touches
AV1.S
AV2.S
AV1/AV2
Intersection (of horizontal projection):
null
(touch result can also be a point)
SUBTR Result:
AV_t1
AV_t
AV_t1 = null
AV1
AV_t depends on altitude computation
Surface from ArispaceVolume1 that has spatial relationship to Surface from AirspaceVolume2

Annex D – Spatial Relation Process Defintions

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd" service="WPS"
version="1.0.0" lang="en-US">
    <!--Contains-->
    <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
        <ows:Identifier>Contains</ows:Identifier>
        <ows:Title>Contains Test</ows:Title>
        <ows:Abstract>
      Performs a contains test for two given features. Checks whether the first feature
      contains the second feature.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="contains"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
        </wps:DataInputs>
        <wps:ProcessOutputs>
            <wps:Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Contains test result.</ows:Title>
                <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature contains
          the second input feature.
        </ows:Abstract>
                <wps:LiteralOutput>
                    <ows:DataType ows:reference="xs:boolean"/>
                </wps:LiteralOutput>
```

```
              </wps:Output>
          </wps:ProcessOutputs>
      </wps:ProcessDescription>
      <!--Covers-->
      <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
          <ows:Identifier>Covers</ows:Identifier>
          <ows:Title>Covers</ows:Title>
          <ows:Abstract>
      Performs a covers test for the two given features. Checks whether the first
feature
      covers the second feature.
    </ows:Abstract>
          <ows:Metadata xlink:title="spatial"/>
          <ows:Metadata xlink:title="covers"/>
          <ows:Metadata xlink:title="AIXM"/>
          <wps:DataInputs>
              <wps:Input minOccurs="1" maxOccurs="1">
                  <ows:Identifier>Feature1</ows:Identifier>
                  <ows:Title>wps:Input Feature 1</ows:Title>
                  <ows:Abstract>The first input feature.</ows:Abstract>
                  <wps:ComplexData>
                      <wps:Default>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>
                              <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                          </wps:Format>
                      </wps:Default>
                      <wps:Supported>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>

      <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                          </wps:Format>
                      </wps:Supported>
                  </wps:ComplexData>
              </wps:Input>
              <wps:Input minOccurs="1" maxOccurs="1">
                  <ows:Identifier>Feature2</ows:Identifier>
                  <ows:Title>wps:Input Feature 2</ows:Title>
                  <ows:Abstract>The second input feature.</ows:Abstract>
                  <wps:ComplexData>
                      <wps:Default>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>
                              <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                          </wps:Format>
                      </wps:Default>
                      <wps:Supported>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>

      <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                          </wps:Format>
                      </wps:Supported>
                  </wps:ComplexData>
              </wps:Input>
          </wps:DataInputs>
          <wps:ProcessOutputs>
              <wps:Output>
                  <ows:Identifier>Result</ows:Identifier>
                  <ows:Title>Covers test result.</ows:Title>
                  <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature covers
          the second input feature.
        </ows:Abstract>
                  <wps:LiteralOutput>
                      <ows:DataType ows:reference="xs:boolean"/>
                  </wps:LiteralOutput>
              </wps:Output>
          </wps:ProcessOutputs>
      </wps:ProcessDescription>
      <!--Crosses-->
      <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
```

```
        <ows:Identifier>Crosses</ows:Identifier>
        <ows:Title>Crosses</ows:Title>
        <ows:Abstract>
      Performs a crosses test for the two given features. Checks whether the first 
feature 
      crosses the second feature.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="crosses"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
        </wps:DataInputs>
        <wps:ProcessOutputs>
            <wps:Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Crosses test result.</ows:Title>
                <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input 
feature crosses
          the second input feature.
        </ows:Abstract>
                <wps:LiteralOutput>
                    <ows:DataType ows:reference="xs:boolean"/>
                </wps:LiteralOutput>
            </wps:Output>
        </wps:ProcessOutputs>
    </wps:ProcessDescription>
    <!--Disjoint-->
    <wps:ProcessDescription processVersion="1" storeSupported="false" 
statusSupported="false">
        <ows:Identifier>Disjoint</ows:Identifier>
        <ows:Title>Disjoint</ows:Title>
        <ows:Abstract>
      Performs a disjoint test for the two given features. Checks whether the two shapes 
are 
      disjoint.
    </ows:Abstract>
```

```
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="disjoint"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
        </wps:DataInputs>
        <wps:ProcessOutputs>
            <wps:Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Disjoint test result.</ows:Title>
                <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature is disjoint from
          the second input feature.
        </ows:Abstract>
                <wps:LiteralOutput>
                    <ows:DataType ows:reference="xs:boolean"/>
                </wps:LiteralOutput>
            </wps:Output>
        </wps:ProcessOutputs>
    </wps:ProcessDescription>
    <!--Equals-->
    <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
        <ows:Identifier>Disjoint</ows:Identifier>
        <ows:Title>Disjoint</ows:Title>
        <ows:Abstract>
      Performs an equals test for the two given features. Checks whether the two shapes
are
      equal.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="equals"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
```

```
                    <ows:Title>wps:Input Feature 1</ows:Title>
                    <ows:Abstract>The first input feature.</ows:Abstract>
                    <wps:ComplexData>
                        <wps:Default>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>
                                <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                            </wps:Format>
                        </wps:Default>
                        <wps:Supported>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                            </wps:Format>
                        </wps:Supported>
                    </wps:ComplexData>
                </wps:Input>
                <wps:Input minOccurs="1" maxOccurs="1">
                    <ows:Identifier>Feature2</ows:Identifier>
                    <ows:Title>wps:Input Feature 2</ows:Title>
                    <ows:Abstract>The second input feature.</ows:Abstract>
                    <wps:ComplexData>
                        <wps:Default>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>
                                <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                            </wps:Format>
                        </wps:Default>
                        <wps:Supported>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                            </wps:Format>
                        </wps:Supported>
                    </wps:ComplexData>
                </wps:Input>
            </wps:DataInputs>
            <wps:ProcessOutputs>
                <wps:Output>
                    <ows:Identifier>Result</ows:Identifier>
                    <ows:Title>Equals test result.</ows:Title>
                    <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature is equal to
          the second input feature.
        </ows:Abstract>
                    <wps:LiteralOutput>
                        <ows:DataType ows:reference="xs:boolean"/>
                    </wps:LiteralOutput>
                </wps:Output>
            </wps:ProcessOutputs>
        </wps:ProcessDescription>
        <!--Intersects-->
        <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
            <ows:Identifier>Intersects</ows:Identifier>
            <ows:Title>Intersects</ows:Title>
            <ows:Abstract>
      Performs an intersection test for the two given features. Checks whether the two
shapes intersect.
    </ows:Abstract>
            <ows:Metadata xlink:title="spatial"/>
            <ows:Metadata xlink:title="intersects"/>
            <ows:Metadata xlink:title="AIXM"/>
            <wps:DataInputs>
                <wps:Input minOccurs="1" maxOccurs="1">
                    <ows:Identifier>Feature1</ows:Identifier>
                    <ows:Title>wps:Input Feature 1</ows:Title>
                    <ows:Abstract>The first input feature.</ows:Abstract>
                    <wps:ComplexData>
                        <wps:Default>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>
                                <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                            </wps:Format>
                        </wps:Default>
                        <wps:Supported>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                            </wps:Format>
                        </wps:Supported>
                    </wps:ComplexData>
                </wps:Input>
                <wps:Input minOccurs="1" maxOccurs="1">
                    <ows:Identifier>Feature2</ows:Identifier>
                    <ows:Title>wps:Input Feature 2</ows:Title>
                    <ows:Abstract>The second input feature.</ows:Abstract>
                    <wps:ComplexData>
                        <wps:Default>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>
                                <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                            </wps:Format>
                        </wps:Default>
                        <wps:Supported>
                            <wps:Format>
                                <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                            </wps:Format>
                        </wps:Supported>
                    </wps:ComplexData>
                </wps:Input>
            </wps:DataInputs>
            <wps:ProcessOutputs>
                <wps:Output>
                    <ows:Identifier>Result</ows:Identifier>
                    <ows:Title>Intersects test result.</ows:Title>
                    <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature intersects
          the second input feature.
        </ows:Abstract>
                    <wps:LiteralOutput>
                        <ows:DataType ows:reference="xs:boolean"/>
                    </wps:LiteralOutput>
                </wps:Output>
            </wps:ProcessOutputs>
    </wps:ProcessDescription>
    <!--Overlaps-->
    <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
        <ows:Identifier>Overlaps</ows:Identifier>
        <ows:Title>Overlaps</ows:Title>
        <ows:Abstract>
      Performs an overlap test for the two given features. Checks whether the first
feature overlaps
      with the second feature.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="overlaps"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
```

```
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
        </wps:DataInputs>
        <wps:ProcessOutputs>
            <wps:Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Overlaps test result.</ows:Title>
                <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature overlaps
          the second input feature.
        </ows:Abstract>
                <wps:LiteralOutput>
                    <ows:DataType ows:reference="xs:boolean"/>
                </wps:LiteralOutput>
            </wps:Output>
        </wps:ProcessOutputs>
    </wps:ProcessDescription>
    <!--Touches-->
    <wps:ProcessDescription processVersion="1.0.0" storeSupported="false"
statusSupported="false">
        <ows:Identifier>AIXMTouches</ows:Identifier>
        <ows:Title>AIXM features Touches method</ows:Title>
        <ows:Abstract>
      Performs a touches test for the two given features. Checks whether the first
feature overlaps
      touches the second feature.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="touches"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
        </wps:DataInputs>
        <wps:ProcessOutputs>
            <wps:Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>Touches test result.</ows:Title>
                <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature touches
          the second input feature.
        </ows:Abstract>
                <wps:LiteralOutput>
                    <ows:DataType ows:reference="xs:boolean"/>
                </wps:LiteralOutput>
            </wps:Output>
        </wps:ProcessOutputs>
    </wps:ProcessDescription>
    <!--Within-->
    <wps:ProcessDescription processVersion="1" storeSupported="false"
statusSupported="false">
        <ows:Identifier>Within</ows:Identifier>
        <ows:Title>Within</ows:Title>
        <ows:Abstract>
      Performs a within test for the two given features. Checks whether the first
feature is within
      the second feature.
    </ows:Abstract>
        <ows:Metadata xlink:title="spatial"/>
        <ows:Metadata xlink:title="equals"/>
        <ows:Metadata xlink:title="AIXM"/>
        <wps:DataInputs>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature1</ows:Identifier>
                <ows:Title>wps:Input Feature 1</ows:Title>
                <ows:Abstract>The first input feature.</ows:Abstract>
                <wps:ComplexData>
                    <wps:Default>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>
                            <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                        </wps:Format>
                    </wps:Default>
                    <wps:Supported>
                        <wps:Format>
                            <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                        </wps:Format>
                    </wps:Supported>
                </wps:ComplexData>
            </wps:Input>
            <wps:Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>Feature2</ows:Identifier>
                <ows:Title>wps:Input Feature 2</ows:Title>
                <ows:Abstract>The second input feature.</ows:Abstract>
```

```
                  <wps:ComplexData>
                      <wps:Default>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>
                              <wps:Schema>http://www.aixm.aero/schema/5.1</wps:Schema>
                          </wps:Format>
                      </wps:Default>
                      <wps:Supported>
                          <wps:Format>
                              <wps:MimeType>text/xml</wps:MimeType>

    <wps:Schema>http://www.aixm.aero/gallery/content/public/schema/5.1/AIXM_Features.xsd
</wps:Schema>
                          </wps:Format>
                      </wps:Supported>
                  </wps:ComplexData>
              </wps:Input>
          </wps:DataInputs>
          <wps:ProcessOutputs>
              <wps:Output>
                  <ows:Identifier>Result</ows:Identifier>
                  <ows:Title>Overlaps test result.</ows:Title>
                  <ows:Abstract>
          Returns a literal boolean which determines whether or not the first input
feature is within
          the second input feature.
         </ows:Abstract>
                  <wps:LiteralOutput>
                      <ows:DataType ows:reference="xs:boolean"/>
                  </wps:LiteralOutput>
              </wps:Output>
          </wps:ProcessOutputs>
    </wps:ProcessDescription>
</wps:ProcessDescriptions>
```