Open Geospatial Consortium, Inc.

Date: 2010-09-09

Reference number of this document: OGC 10-079r3

Category: Public Engineering Report

Editor: Thomas Everding

OGC® OWS-7 Aviation Architecture Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

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

Preface

This public Engineering Report (ER) is a deliverable of the Open Geospatial Consortium (OGC) Interoperability Program Open Web Service (OWS) Testbed phase 7 (OWS-7).

The document describes the architecture that was implemented in the Aviation thread of OWS-7. The document provides an overview of the architecture and describes the implemented components. In addition it discusses "eventing" and notification techniques relevant for the aviation domain.

This work was supported by the European Commission through the GENESIS project (an Integrated Project, contract number 223996).

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Contents

Page

Figures

Page

Tables

Page

OGC® OWS-7 Aviation Architecture Engineering Report

1 Introduction

1.1 Scope

This OGC™ document describes the architecture used for the implementation of the OWS-7 Aviation scenarios. This includes an overview of the implemented components and workflows followed by a short description of each component.

This OGC™ document also contains a discussion about eventing and notification related techniques and problems that were faced during the OWS-7 testbed and are relevant to the aviation domain.

This OGC™ document does not describe the OWS-7 Aviation scenarios in large detail. Please refer to the according separate Engineering Report.

This OGC™ document does not describe the OWS-7 Aviation portrayal work in large detail. Please refer to the according separate Engineering Report.

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 Inc. 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.

1.2 Document contributor contact points

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

Name	Organization
Thomas Everding (editor)	Institute for Geoinformatics (IfGI), University of Münster, Germany
Johannes Echterhoff Ingo Simonis	International Geospatial Services Institute GmbH (iGSI)
Bruno Simmenauer	ALTICODE
Nadine Alameh	OGC
Debbie Wilson, Daniel Hardwick	Snowflake Software
Jan Vanacek	Frequentis
Alex Brooker	Envita
Robin Houtmeyers, Jeroen Dries	Luciad
Yuqi Bai	George Mason University
Ulrich Berthold	Comsoft
Leif Stainsby	Galdos
Sébastien Geindre	Meteo France
Thibault Dacla	Atmosphere

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
17.03.2010	0.0.0	TE	all	Initial draft
27.05.2010	0.1.0	TE	all	Release version for pending documents
31.05.2010	0.1.1	TE	7.3.2	Added ALTICODE contribution
09.06.2010	0.1.2	TE	7 and 8	Added contributions from Envita, Luciad, Snowflake and Frequentis, included final version of chapter 8 from iGSI
10.06.2010	0.1.3	TE	7	Added contributions from GMU, Meteo France, Comsoft and Galdos
11.06.2010	0.2.0	TE	all	Finalization
01.07.2010	0.2.0	TE	7.11	Added Atmosphere contribution

1.4 Future work

The following future work items for the aviation domain were identified in OWS-7:

- **Filtering of Aviation Events**
    - Well defined event channels yield the opportunity to get significant performance gains in an AIM Event Architecture. Instead of only doing content based filtering to get events of interest, channels can be defined that provide events that are commonly requested. For example, event channels could be defined that contain events from specific airports or for geographic regions. Best practices should be developed in the aviation domain for defining **aviation event channels** and the publication as well as subscription to these channels.
    - The process of subscribing for relevant aviation events needs to be simplified. This concerns both the representation of data that needs to be queried (e.g. the spatial extent of an event but also the full event data in dnotam Events) and the way filter statements are created. Various options to improve the situation are discussed in this report, among which are **domain specific filter functions** and **stored queries in services** (WFS but also Event Service)
    - The **accuracy of subscription filters** needs to be improved in order to further reduce the amount of uninteresting events matching a subscription. One interesting mechanism is to use **dynamic spatial filters**.
    - **Minimum filter capabilities** and **minimum content requirements** for dnotam events need to be evaluated and defined. This helps to create a specification for dnotam events that reduces the event size as much as possible but still enables sophisticated event filtering.
- **Subscription Management Strategies** – The lifecycle of subscriptions used in the testbed was rather simple. A subscription was created before the flight and explicitly terminated by the pilot and dispatcher at a given point in time. To facilitate the management of subscriptions, available functionality that was not used in the testbed should be tested (like automatic termination of subscriptions at given points in time). In addition, rules can be developed to automatically remove subscriptions for example when an airport is no longer within certain distance to the remaining flight path of an aircraft.
- **Feature Revision Mechanism** – GML features have standard properties, some of which are labels or a feature's identifier. AIXM time slices show that the provision of the correction number for the representation of certain data is a valuable mechanism. Dnotam events should also incorporate a mechanism to indicate possible revisions. This mechanism should also be considered as an extension to the standard GML feature properties.

- **Define Previous Event Semantics** – A dnotam event can point to a previous event. If the relationship between a given event and its previous event can have different semantics in the aviation domain, the dnotam event model should be extended to indicate the exact semantics to avoid the need to infer them.

- **Rules to Define Spatial Extent Computation for given Feature Type** – A feature often has a spatial extent. This extent is encoded in one or more of a feature's properties. It may be necessary to perform complex computations to get the features extent from these properties. Rules should be developed to encode the information necessary to compute the spatial extent for features of a given type in the definition of the according feature type. Such rules, if they were machine readable, would enable automatic computation of the spatial extent for any given feature by an Event Service. Problems that arise because of the heterogeneity of feature type definitions as well as the leeway given to data providers in populating the boundedBy properties of features could be solved with such a mechanism.

- **Clarify Semantics of Sequence Number Property** – Like time slices, a dnotam event may have sequence numbers. The purpose of the sequence number in an event needs to be clarified especially with respect to duplication of functionality provided by messaging middleware / standards that support reliable delivery of a sequence of events.

- **Publish-subscribe based extensions for Feature Portrayal Services** should be tested in order to allow the FPS to cache the feature data and automatically stay up-to-date to improve the portrayal speed even when using volatile data.

- **Investigation of common subscription mechanisms for different clients** should be investigated. Such mechanisms could guarantee that all notifications received by the dispatcher are also sent to the pilot automatically. Frequentis did some test in this direction already, another approach could be to make use of ad-hoc event channels as discussed in the Event Architecture ER (10-060) in section 6.2.2.8.

- **Client side caching** of aeronautical feature has to be investigated with a special focus on updating strategies. In case of an update (e.g. NOTAM received via the Event Service) the cached features could be updated or released from the cache in order to reload the current state from a WFS when accessed.

- **The scope of the data coverage of WFS instances should be made transparent.** In OWS-7 there was one instance serving U.S. data (Snowflake) and another serving European data (Comsoft). To avoid that clients need to know which from which instance the data has to be requested sufficient catalogues or aggregating WFSs that would delegate requests to those instances serving the data should be used.

- It is recommended that for **future release of WXXM, that feature types extended from observation feature types should extend from specialized observation feature types** classified by their result type as recommended in ISO/DIS 19156 Geographic information – *Observations and Measurements*. This

will enable generic tools such as GO Loader to parse and load the result properties.

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 Inc. 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.

OGC 03-105r1, *OpenGIS® OpenGIS Geography Markup Language (GML) Encoding Specification*

OGC 04-094, *OpenGIS® Web Feature Service Implementation Specification*

OGC 04-094, *OpenGIS® Filter Encoding Implementation Specification*

OGC 05-077r4, *OpenGIS® Symbology Encoding Implementation Specification*

OGC 05-078r4, *OpenGIS® Styled Layer Descriptor profile of the Web Map Service Implementation Specification*

OGC 06-042, *OpenGIS® Web Map Server Implementation Specification*

OGC 06-121r3, *OpenGIS® Web Services Common Standard*

OGC 07-006r1, *OpenGIS® Catalogue Services Specification*

OGC 07-036, *OpenGIS® Geography Markup Language (GML) Encoding Standard*

OGC 07-067r5, *OpenGIS® Web Coverage Service (WCS) Implementation Standard*

OGC 07-110r4, *OpenGIS® CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW*

OGC 08-132, *OpenGIS® Event Pattern Markup Language (EML)*

OGC 08-133, *OpenGIS® Sensor Event Service Interface Specification*

OGC 09-025, *OpenGIS® Web Feature Service Implementation Specification 2.0* as sent to the ISO Central Secretariat for issuing as Draft International Standard

OGC 09-026, *OpenGIS® Filter encoding 2.0*, as sent to the ISO Central Secretariat for issuing as Draft International Standard

OGC 09-032, *OpenGIS® OWS-6 SWE Event Architecture Engineering Report*

OGC 09-050r1, *OpenGIS® OWS-6-AIM Engineering Report*

OGC 10-060, *OpenGIS® OWS-7 Event Architecture Engineering Report*

OGC 10-127, *OpenGIS® OWS-7 Aviation Portrayal Engineering Report*

OGC 10-131, *OpenGIS® OWS-7 Aviation – AIXM Assessment Report*

OGC 10-132, *OpenGIS® OWS-7 Aviation – WXXM Assessment Engineering Report*

OASIS *OASIS/ebXML Registry Information Model v3.0*

OASIS *Web Services Base Notification 1.3*

OASIS *Web Services Brokered Notification 1.3*

OASIS *Web Services Reliable Messaging TC WS-Reliability 1.1*

OASIS *Web Services Topics 1.3*

ISO 19115 (all parts), *Geographic information – Metadata*

ISO 19119, *Geographic information - Services*

ISO 19139, *Geographic information - Metadata - XML schema implementation*

ISO 19143, *Geographic information - Filter encoding*

ISO 19156, *Geographic information – Observations and Measurements*

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*

3 Terms and definitions

Please refer to the OWS-6 Event Architecture Engineering Report (09-032) and the OWS-6 AIM Engineering Report (09-050r1).

4 Conventions

4.1 Abbreviated terms

AC	Aviation Client
AIM	Aeronautical information Management
AIXM	Aeronautical Information Exchange Model
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ATC	Air Traffic Control
ATM	Air Traffic Management
CDC	Change Data Capture
CEP	Complex Event Processing
COTS	Commercial Off-The-Shelf
CRS	Coordinate Reference System
CSW	Catalog Service – Web
ebRIM	Electronic Business Registry Information Model
ebXML	Electronic Business Extensible Markup Language
EFB	Electronic Flight Bag
EML	Event Pattern Markup Language
ER	Engineering Report
ESB	Enterprise Service Bus
ESP	Event Stream Processing
FE	Filter Encoding
FES	Filter Encoding Specification
FPS	Feature Portrayal Service
FTP	File Transfer Protocol
GML	Geography Markup Language
HTTP	HyperText Transport Protocol
ISO	International Standardization Organization

METAR	METeorological Aerodrome Report (may vary)
NOTAM	NOTice To AirMen
OGC	Open Geospatial Consortium
OS	Operating System
OWS	OGC Web Services
OWS-7	OWS testbed phase 7
PIREP	Pilot REPort
RDBMS	Relational Data Base Management System
SE	Symbology Encoding
SES	Sensor Event Service
SIGMET	SIGnificant METeorological phenomena
SLD	Styled Layer Descriptor
SQL	Structured Query Language
SWE	Sensor Web Enablement
SWIM	System Wide Information Management
UI	User Interface
UML	Unified Modeling Language
WCS	Web Coverage Service
WFS	Web Feature Service
WFS-T	WFS-Transactional
WMS	Web Map Service
WS-N	Web Services - Notification
WXXM	Weather Information Exchange Model
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

5 OWS-7 Aviation Architecture overview

The architecture and the involved components used in the OWS-7 Aviation thread can be separated in three tiers (see Figure 1):

- The Access Tier contains the various Web Feature Services serving the base data but also the 4D Weather Data Cube and other aviation infrastructure components like SWIM services.
- The Client Tier contains the client applications.
- The Business Process Tier lies between the other tiers. Components in this tier do not serve data but offer additional services for the clients on top of the Access Tier like discovery, portrayal, data handling and notification mechanisms.

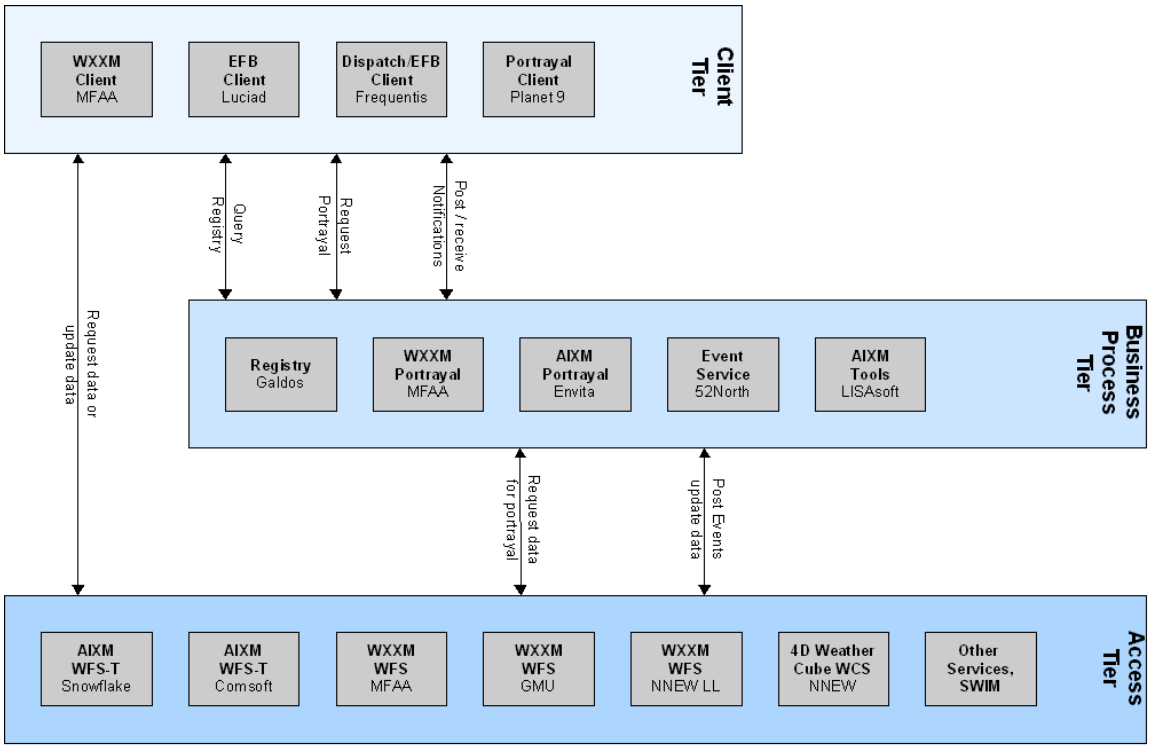

Figure 1 - Overview of the components

Descriptions of the components can be found in section 7.

6 Workflows

This section describes the most important workflows performed in the OWS-7 Aviation thread. The first describes the basic actions that are commonly executed in the demonstration scenarios. The second workflow shows the general interaction with the Portrayal Service.

6.1 Basic Workflow

Figure 2 gives an overview of the basic workflow steps. The first group shows how data is requested, the second how a subscription in made and the third how updates to the data are made.

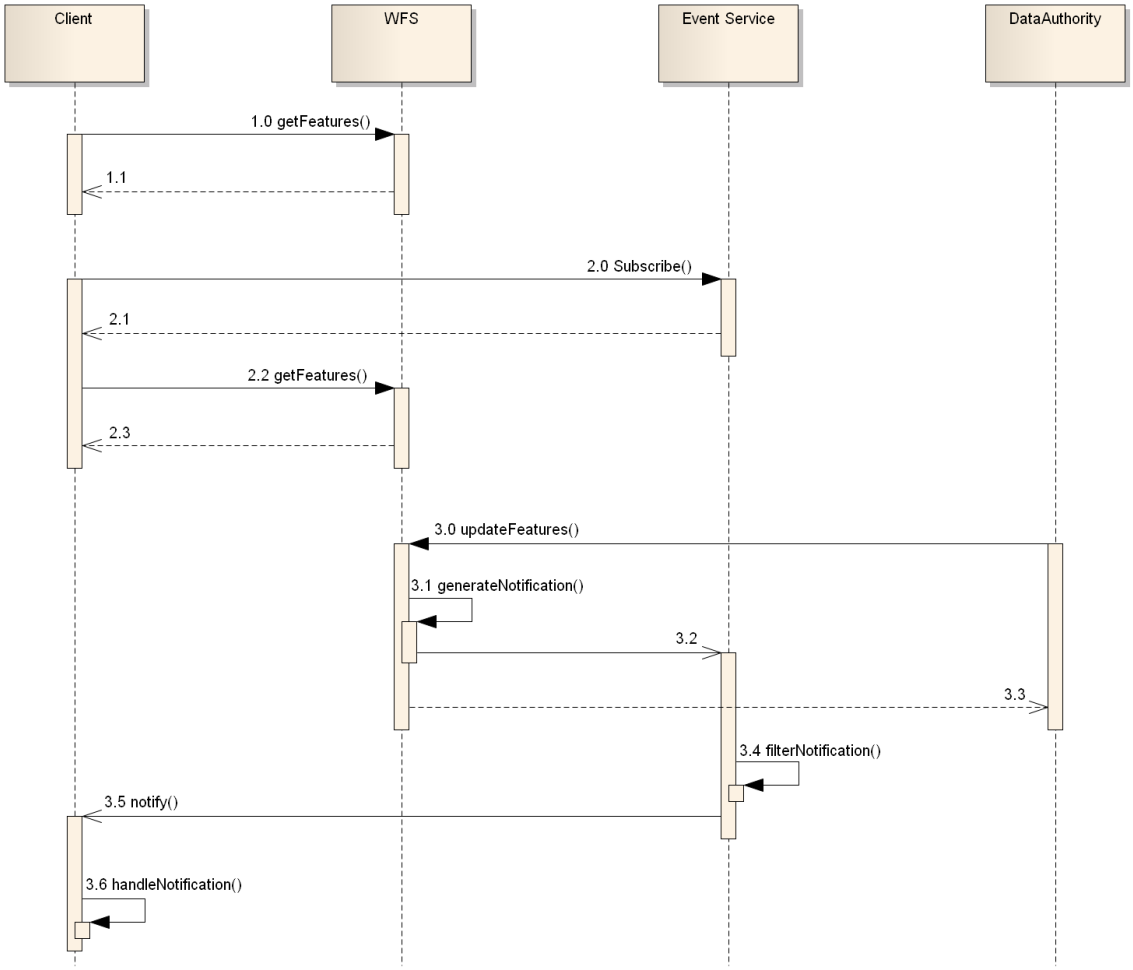

Figure 2 - Sequence of the basic workflow

1.0 and 1.1: In these steps data is requested from Web Feature Services. In the aviation scenarios this happens multiple times for instance when loading base data or requesting features with specific properties.

2.0 and 2.1: In this step a subscription for notifications is made at the Event Service. A subscription usually contains a filter statement identifying the notifications of interest. This may for instance be digital NOTAMs regarding a specific airport or weather information for a specific region.

2.2 and 2.3: These steps are similar to 1.0 and 1.1 as also data is requested from a WFS. Here it is performed in order to get the most recent state of the data. This should be done after each subscription. Otherwise it is possible to miss important information if it is published after the last data request but before the subscription for notifications.

3.0: In this step an update of the base data is posted to the WFS. This may for instance be a change in the AIXM baseline or a new weather forecast.

3.1: Besides adding the update to the underlying database the WFS generate a notification containing the new information.

3.2: The notification is forwarded to the Event Service for further publication.

3.3: From the perspective of the WFS the update is handled completely. Thus, it closes the communication with the authority entity that started the update process.

3.4: The Event Service filters the received notifications against the criteria given within the subscriptions made.

3.5: If a notification matches the subscription criterion of a client the notifications is forwarded.

3.6: The client handles the received notification e.g. displaying the new information.

6.2 Portrayal Workflow

The basic workflow describe above can be enriched by using Portrayal Services. These services allow requesting a rendered map that is generated on request using feature user defined features. Figure 3 gives an overview of the components used for the portrayal.

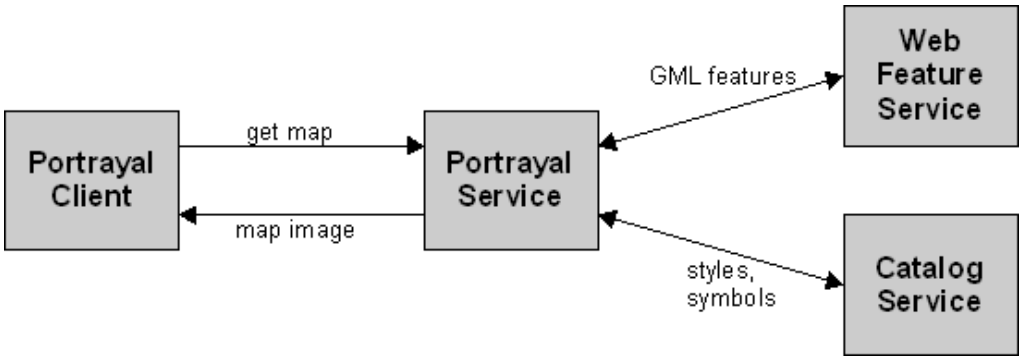

Figure 3 - Overview of the portrayal components

The workflow in detail is shown in Figure 4:

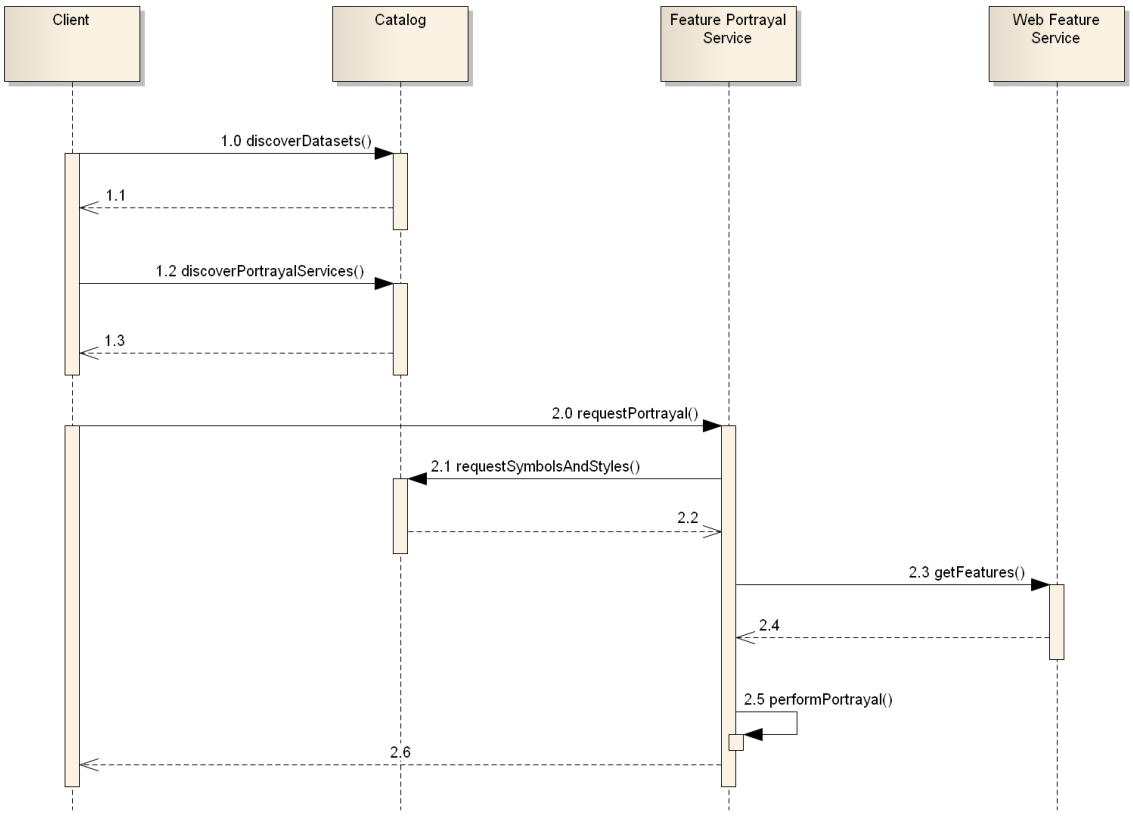

Figure 4 - Sequence of the portrayal workflow

1.0 and 1.1: The Client discovers datasets that shall be rendered by the portrayal service.

1.2 and 1.3: The Client discovers the available portrayal services.

2.0: The Client requests the portrayal at the selected portrayal service.

2.1 and 2.2: The Portrayal Service requests the necessary styles and symbols that shall be used at the catalog.

2.3 and 2.4: The Portrayal Service requests the features that shall be rendered from a Web Feature Service.

2.5: With all necessary information available, the Portrayal Service renders the map.

2.6: The map is returned to the Client.

Additional workflows and further information on portrayal and its use in the OWS-7 Aviation thread can be found in the separate OWS-7 Aviation Portrayal ER (OGC 10-127).

7 Component Descriptions

7.1 Snowflake AIXM WFS

Snowflake Software's GO Publisher commercial off-the shelf (COTS) product is comprised of flexible, scalable components capable of supporting the transformation and data exchange requirements of aeronautical and weather information systems (Figure 5).

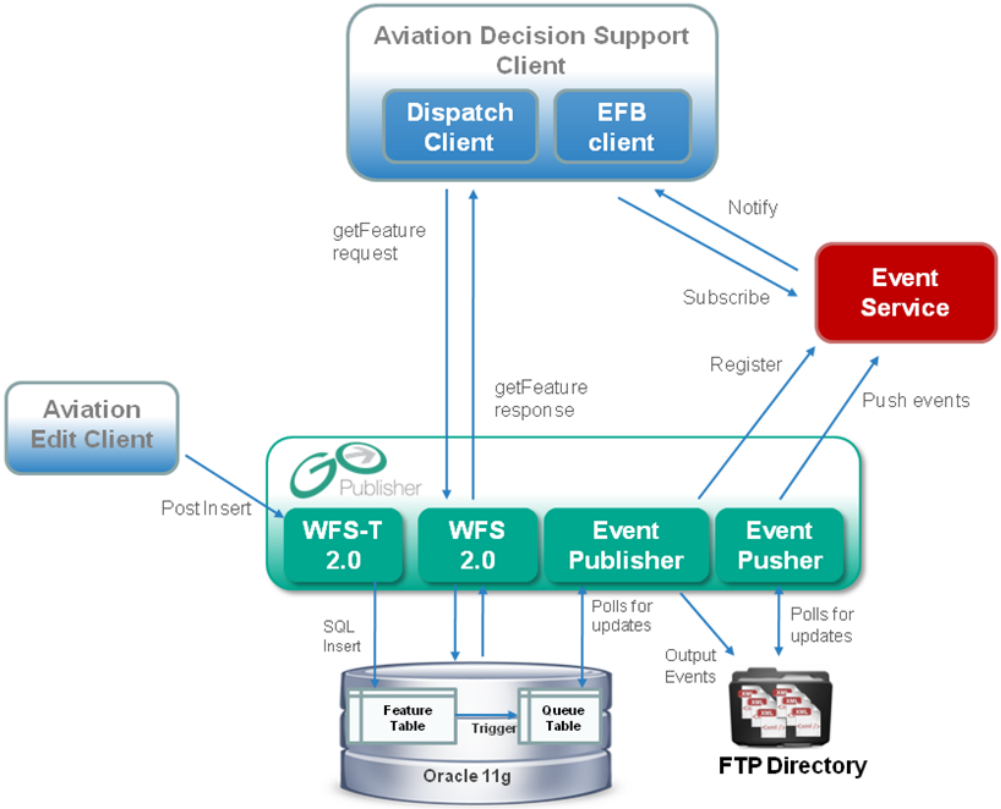

Figure 5 - Overview of the Snowflake aviation component architecture

Aeronautical and weather data are published using GO Publisher via four components:

1. **WFS 2.0**: this provides read-only access to AIXM 5.1 and WXXM 1.1 data by the aviation decision support clients: Dispatch Client and Electronic Flight Bag (EFB)

2. **WFS-T 2.0**: this allows aviation clients (create/edit) authorized to update the data store to post inserts of time slices to the Snowflake AIXM 5.1 data store

3. **Event Publisher**: creates events such as digital NOTAMs or WXXM events from the source database which are pushed to an event service

4. **Event Pusher**: is a java application that is registered at the Event Service via SOAP messages. It then runs as a background process polling the FTP directory for new NOTAMs which it then pushes to the Event Service within a SOAP container. The sent NOTAMs are then saved to an archive folder.

7.1.1 AIXM WFS-T 2.0

GO Publisher WFS-T 2.0 supports the following mandatory and optional operations / functions as specified by ISO/DIS 19142 Geographic information -Web Feature Service and ISO/DIS 19143 Geographic Information -Filter Encoding specifications (Table 1). Based on experience gained from OWS-6 and the Eurocontrol Digital Snowtam trials, our aim was to provide a comprehensive implementation of the WFS 2.0 specification and to test the new operations and functions to improve retrieval of AIXM features and properties. See the 10-131 OWS-7 Aviation - AIXM Assessment Report for further details.

Table 1 - Summary of operations and functions supported by GO Publisher WFS 2.0 within OWS-7

		WFS 2.0 Conformance Levels				
Feature		**Simple**	**Basic**	**Transactional**	**Locking**	**GO Publisher WFS 2.0**
Service Metadata	Get Capabilities	✓	✓	✓	✓	✓
	WSDL					✓
	Describe Feature Type	✓	✓	✓	✓	✓
Retrieval	Get Feature	✓	✓	✓	✓	✓
	Get Property Value		✓	✓	✓	✓
Stored Queries	Describe Stored Queries	✓	✓	✓	✓	✓
	List Stored Queries	✓	✓	✓	✓	✓
Manage Stored Queries	Create Stored Queries					
	Drop Stored Queries					
Transactions	Transaction			✓	✓	✓
	Lock Feature				✓	
	Get Feature With Lock				✓	
Filter Encoding (ISO 19143)	Minimal FE Spatial Filter (BBOX)		✓	✓	✓	✓
	Spatial Filter					✓
	Temporal Filter					✓
	Standard Filter					✓
	Version Navigation					
	Sorting					
	Stored Query					✓
	Ad hoc Query					✓

		WFS 2.0 Conformance Levels				
Feature		Simple	Basic	Transactional	Locking	GO Publisher WFS 2.0
Binding	HTTP GET					✓
	HTTP POST					✓
	SOAP					✓

7.1.1.1 GO Publisher WFS 2.0 Query Operations

Within OWS-7 two typical scenarios were developed to demonstrate how aviation clients such as dispatch clients and EFBs will access and use AIXM 5.1 and WXXM 1.1 data through a web feature service via the GetFeature request/response operation. Within both scenarios, both the dispatch client and EFB must retrieve data from the WFS during pre-flight dispatch and en-route workflows. These data requests range from:

- Simple Queries: where the user posts a request for all instances of specified feature types and optionally for a specific time and location
  Example: Dispatcher or Pilot – select all feature types representing the specified destination airport for the flight (e.g. EETN)
  (NOTE the AIXM temporality model requires that the selection includes a filter selecting the time slices for each feature in effect when a flight is scheduled to land at the destination airport)

- Complex Queries: where the user posts a query requesting only data that meet specific criteria.
  Example: Dispatcher identifying alternate destination airports for a flight – select all airports located within US airspace and within 200 nautical miles of route that has:

  - A runway that is at least 7,000 ft and has a hard surface
  - Passenger facilities
  - Re-fueling facilities
  - Must not have a scheduled closing time between 15:00 and 23:00

To support both simple and complex queries that will be required, GO Publisher WFS 2.0 supports a comprehensive set of ISO/DIS 19143 filter encoding query expressions (Table 2).

Table 2 - ISO/DIS 19143 Filter Encoding query expressions supported by GO Publisher WFS 2.0

Query Expression Category	Query Expression
Comparison Expressions	Equal to, Not equal to, Less than, Greater than, Less than or equal to, Greater than or equal to, Like, Is null and Between

Logical Expressions	And, Or, Not
Spatial Filter Expressions	Equals, Disjoin, Touches, Within, Overlaps, Crosses, Intersects, Contains, DWithin, Beyond, BBOX
Temporal Filter Expressions	After, Before, Begins, BegunBy, TContains, During, TEquals, TOverlaps, Meets, OverlappedBy, MetBy, EndedBy

The WFS 2.0 specification also has extended the mechanisms for retrieving data through the inclusion of an additional request operation: GetPropertyValue. The GetPropertyValue operation enables the retrieval and of specific property values only in the query response rather than the whole feature that contains the property values. Therefore, the GetPropertyValue request will be important in the scenario where a dispatcher or pilot only wants to be presented with a list of suitable diversion airports.

GO Publisher WFS has been configured to support both ad-hoc and stored queries.

- **Ad-hoc queries**: are specified on demand by the user and are not known by the service before they are executed

- **Stored queries**: are pre-configured queries that are stored within the WFS that can be executed by simply using the query's identifier and inputting only the query variable value

The inclusion of stored query within WFS 2.0 offer significant benefits for aviation clients retrieving AIXM data. The most significant benefits of stored queries will be the simplification of complex HTTP POST queries by parameterizing queries to be invoked via HTTP GET queries.

GO Publisher WFS 2.0 shall support the two mandatory stored query operators:

- **ListStoredQueries**: which returns a list of available stored queries offered by the WFS

- **DescribeStoredQueries**: which provides information describing the a stored query

- **CreateStoredQueries**: which enables users to create a stored query

The remaining stored query operations: and DropStoredQueries have not been implemented. However, stored queries can be dropped by the WFS administrator.

Stored query expressions enable complex filter encoding expressions to be encoded as more concise key-value pairs (KVP). To investigate how stored queries can be used to simplify the queries for retrieving AIXM 5.1 data, particularly to support generation of efficient queries for retrieving time slices, Snowflake configured four stored queries in addition to the mandatory GetFeatureByID stored query:

- **urn:ogc:def:query:OGC-WFS:GetFeatureById (Mandatory)**: the GetFeatureByID stored query enables the client to request features based on their gml:ID.

- **urn:snowflake:def:query:AirportHeliportTemporalBufferQuery**: This stored query accepts the parameters: "beginPosition","endPosition", "lineString" and "distance" to return AirportHeliports within a buffer distance of a linestring or point

- **urn:snowflake:def:query:OGC-WFS:AirportHeliportByDesignatorQuery**: this stored queries accepts the parameter “designator” to return all AirportHeliport features containing the requested designator (e.g. KJFK, KBOS)

- **urn:snowflake:def:query:gmlIdentifierTemporalQuery**: this stored query aims to simplify a long HTTP POST query requesting timeslices for a specified validTime range for specific AirportHeliports based on their gml:ID.

- **urn:snowflake:def:query:PropertyIsEqualToQuery**: this stored query will return AirportHeliport features based on their path length.

For a detailed summary of the applicability of stored queries to effectively retrieve AIXM data see 10-131 OWS-7 Aviation – AIXM Assessment Report.

7.1.2 AIXM WFS-T and Event Publisher

The creation of events re-used the Event Source Publication architecture developed within OWS-6. The Event Source Publication system is based on GO Publisher Agent which is a command-line bulk data publishing system. This system has been extended by the development of an event pusher, which posts the events created by GO Publisher Agent to the event service.

To enable publication of events based on the insertion of a time slice into the database, a Change Detection Module or Change Data Capture (CDC) module sits on the database side and uses APIs or triggers to identify these inserts.

As Oracle 11g was used as the AIXM 5.1 RDBMS, a CDC API was used. The implementation in this AIM thread uses database triggers to alert an agent that creates the Digital NOTAMs via a WFS request.

Triggers on the database tables populate a GO Publisher Agent queue database table. GO Publisher Agent reads the queue and constructs the NOTAM including a bounding box For NOTAM updates for non-geographic features GO Publisher Agent extracts the geometries from related features and constructs a bounding box.

Events are pushed to the Event Service via and Event Pusher.

The information flow diagram in Figure 6 shows the message exchange in the pre-flight and in-flight conditions and inter-operations between the Clients, WFS and the Event Service.

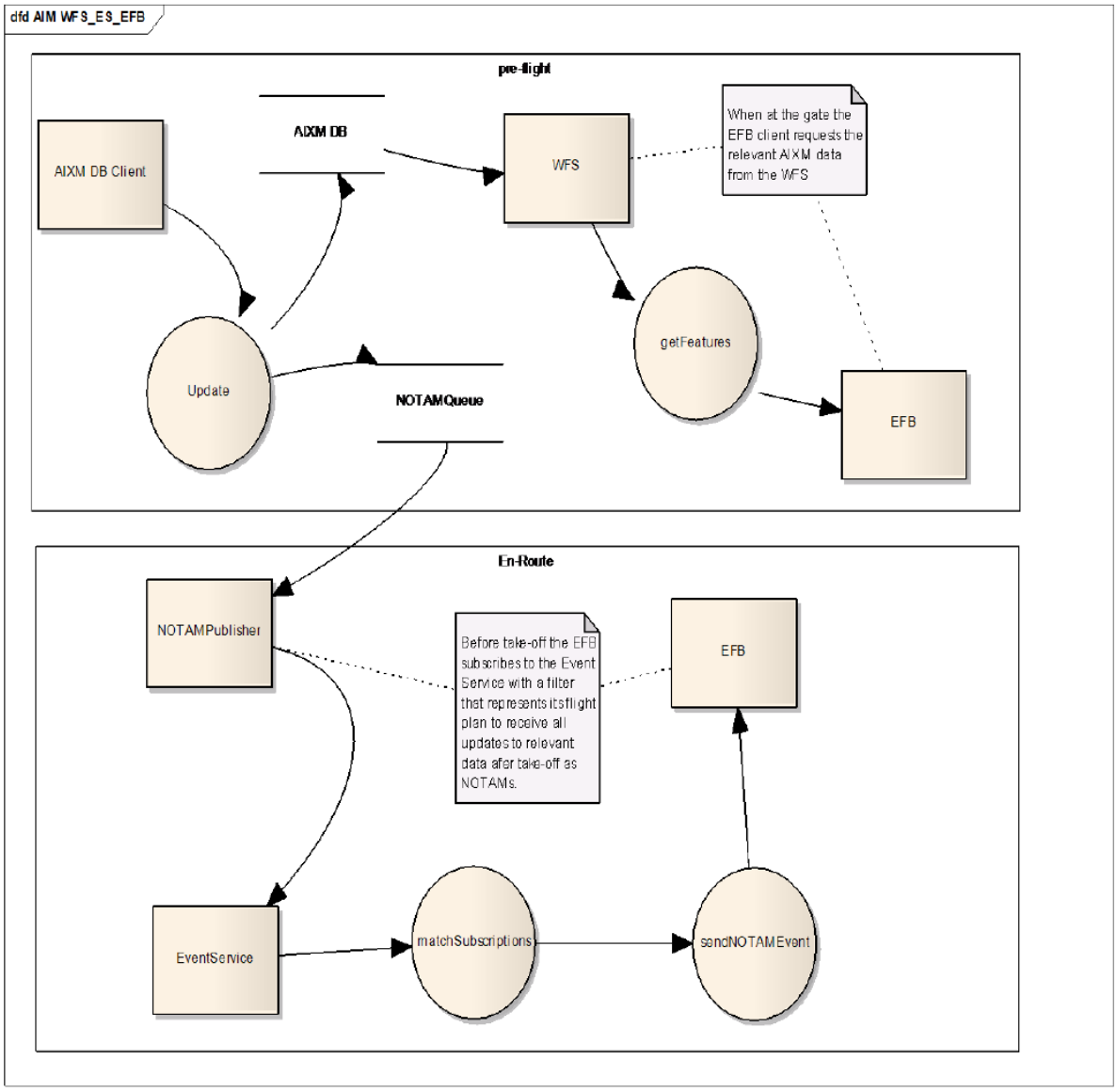

Figure 6 - Generation of events triggered by insertion of time slices

The updating of the AIXM database causes a trigger to create an update alert that is written to an Alert Queue. The alert queue is monitored by the NOTAM Agent and a NOTAM event is populated via a WFS request providing the feature ID of the changed AIXM feature and then sent to the NOTAM queue from where it is fetched by a Publisher service that pushes the NOTAM event to the Event Service for subscription matching and propagation to matching subscribers as is shown in Figure 7.

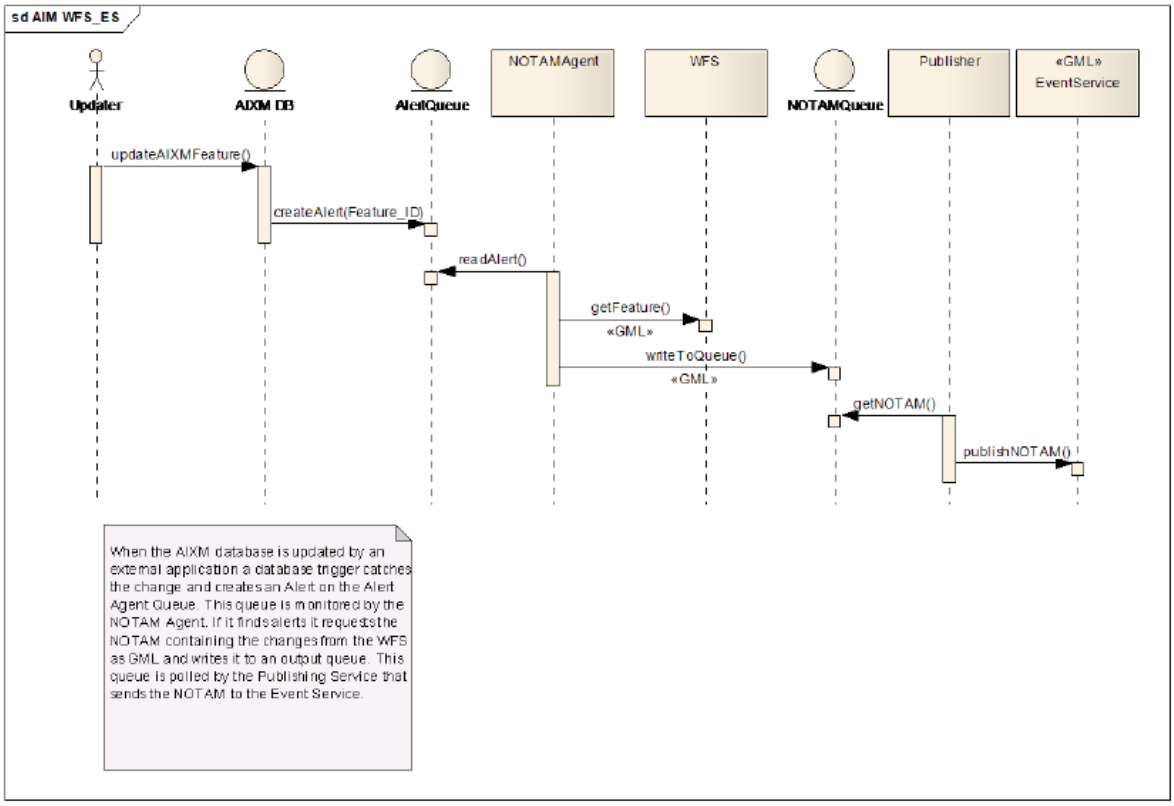

Figure 7 - Sequence Diagram summarizing the GO Publisher event source architecture

7.1.3 Inserting AIXM 5.1 timeslices via WFS-T 2.0

The GO Publisher WFS-T combines the loading and schema translation and capability of GO Loader alongside the translation and publication capabilities of GO Publisher WFS. It only supports the INSERT transaction operation for adding new features to the database. It has been assumed that as AIXM 5.1 data does not require the baseline data to be modified or edited. When changes to the state of an aeronautical feature occur or correction are made these are inserted as new features.

7.1.4 WXXM WFS 2.0

For the Snowflake GO Publisher WXXM WFS2.0 sample WXXM1.1 data was obtained from the Meteo France WFS1.1 which included icing and volcanic ash SIGMET features. GO Loader was used to load this WXXM 1.1 GML into the main data stored. GO Publisher was then configured and deployed to serve this data via a WFS 2.0 interface.

7.1.5 SWIM Components

The Snowflake Software SWIM architecture consists of these main components:

- Red Hat Enterprise Linux Server
- FUSE ESB 4.2

- GO Publisher WFS
- Oracle 11g Database

The server the used to host the technology is a 64bit Intel powered system running Red Hat Enterprise Linux version 5.2. Running on this server is FUSE ESB 4.2 which is used to deploy the GO Publisher WFS. The final piece in the architecture is an Oracle database that GO Publisher will query in order to provide data from where the response to the WFS requests originates.

7.1.6 Data Loading

Snowflake Software acquired aeronautical and meteorological data from a wide range of providers and services and re-used AIXM 5.0 data from OWS-6. These data were supplied as XML or harvested from existing WFS. As AIXM 5.1 and WXXM 1.1 are new specifications, few datasets were available conforming to these application schemas. Consequently, most data received were encoded according to a wide range of existing application schema.

Although this a significant challenge to overcome, Snowflake's commercial off-the-shelf (COTS) products: GO Loader and GO Publisher have generic schema parsing and transformation functionality that enabled these data to be transformed into AIXM 5.1 and WXXM 1.1 within days of receipt. This enabled AIXM 5.1 and WXXM WFS to be deployed and made available to support the development of the dispatch clients and Electronic Flight Bags (EFB) early in the test bed to maximize client productivity.

The ability to use generic schema translation technologies that did not require the need to write bespoke transformation scripts to support each of source datasets (which would have taken months to develop instead of days of configuration) enabled the test bed to have access to and use a comprehensive dataset necessary to support the proposed scenarios.

7.1.6.1 Aeronautical data

Aeronautical data made available to OWS-7 were supplied in a wide range of application schema (Table 3). To create an OWS-7 AIXM 5.1 data store, Snowflake transformed these disparate aeronautical datasets into AIXM 5.1 prior to loading into the Snowflake OWS-7 data store (Figure 8). GO Loader was used to automatically generate the AIXM 5.1 database schema from the AIXM 5.1 application schema and any datasets supplied in AIXM 5.1 were loaded directly into the OWS-7 AIXM 5.1 data store.

Datasets were initially loaded into an Oracle 11g staging database using GO Loader. GO Loader was used to automatically generate database schemas aligned to the source application schema. Once the database tables were created, the XML data were transformed and loaded into the database. This process was repeated for each non AIXM 5.1 dataset. Datasets that were

Once all of the data were loaded into the staging database, GO Publisher Desktop was connected to the database and used to define the schema translations required to transform the data into AIXM 5.1. Once configured, GO Publisher Desktop was used to publish AIXM 5.1 files from the staging database which were then loaded into the OWS-7 AIXM 5.1 data store using GO Loader (Figure 9).

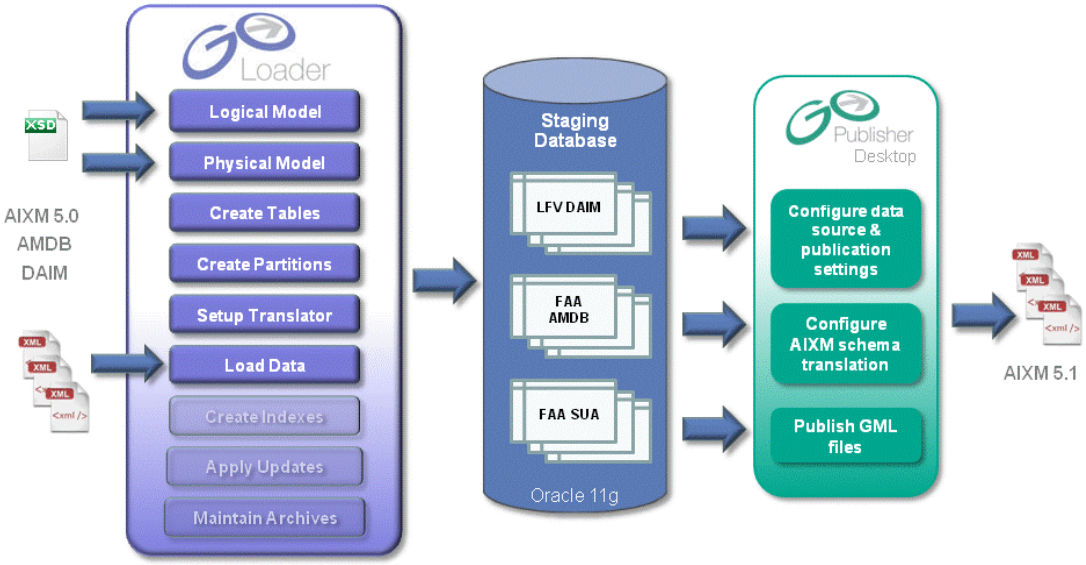

Figure 8 - Translating aviation data into AIXM 5.1: initial translation of non AIXM 5.1 data to publish into OWS-7 AIXM database

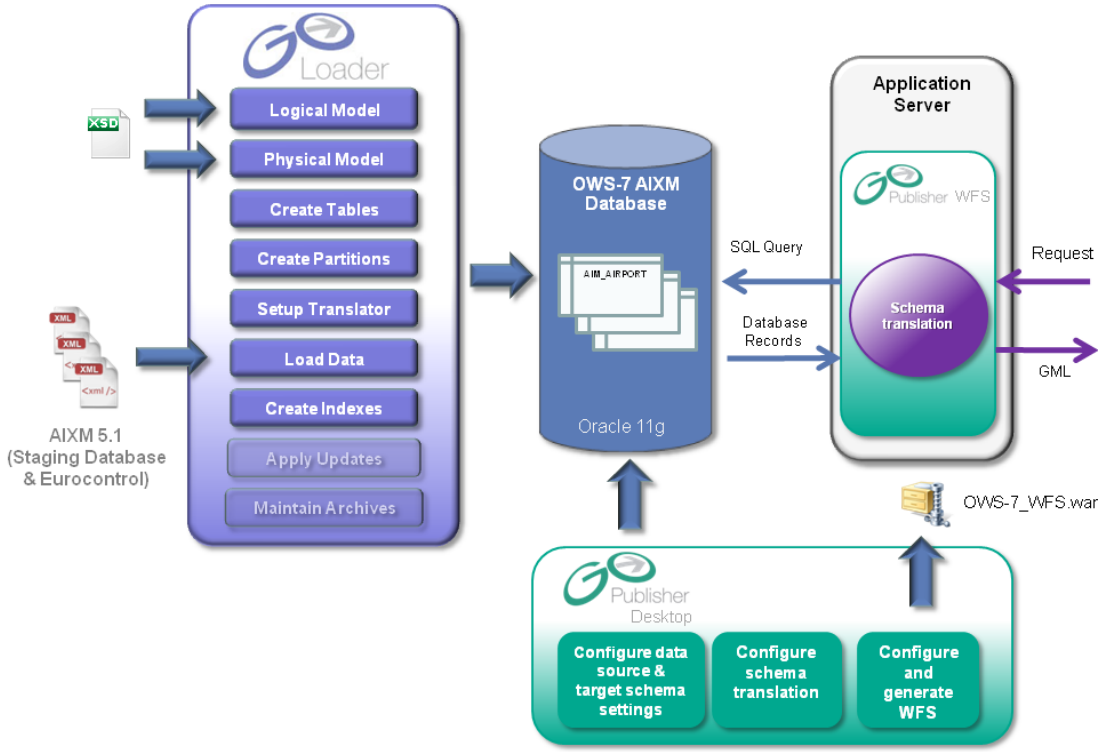

Figure 9 - Translating aviation data into AIXM 5.1: initial load of AIXM 5.1 data into OWS-7 AIXM database

Table 3 - Datasets received and transformed for publication via Snowflake WFS-T 2.0

Provider	Data Source	Application Schema	FeatureTypes extracted	Comments
Aeronautical Data				

Provider	Data Source	Application Schema	FeatureTypes extracted	Comments
FAA	AMDB WFS	AMDB GML 3.1.1.	• amdb:amdb_ runway • amdb:amdb_ taxiway • nasr:nasr_arp • nasr:nasr_ navigation_aid • amdb:amdb_ navaid • amdb:amdb_ control_tower • amdb:amdb_ obstacle • geography: states • geography: rivers	AMDB sample data while similar did not exactly match the AIXM5.1 structure. For example a single amdb_runway feature type was used to create both the AIXM Runway and RunwayElement features types.
	Special Use Airspace (SUA)	AIMS 5.0	• Airspace • GeoBorder • AirTraffic ControlService	FAA SUA AIXM5.0 data was manually modified to replace ArcbyCentrePoint, CirclebyCentrePoint and href geometries with geometries types supported by Oracle Spatial to enable the data to be loaded. Although we were provided with several SUA datasets we only loaded the datasets applicable to scenario 2 due to the overhead involved in manually transforming the data.
Euro-control	Snowtam Trial (TempDelta)	AIXM 5.1	• aixm:Airport Heliport • aixm:Runway • aixm:Taxiway	
	ESMS Scheduled Availability (Baseline)	AIXM 5.1	• aixm:Airport Heliport	

Provider	Data Source	Application Schema	FeatureTypes extracted	Comments
LFV	DAIM Baseline	DAIM GML 2.1	• DAIM_BASELINE.AM_AERODROMEREFERENCEPOINT • DAIM_BASELINE.AM_RUNWAYELEMENT • DAIM_BASELINE.AM_TAXIWAYELEMENT	DAIM sample data while similar did not exactly match the AIXM5.1 structure. For example a single Runway feature type was used to create both the Runway and RunwayElement features types.
OWS-6	OWS-6 AIXM Oracle Database (Baseline)	AIXM 5.0	• aixm:Route • aixm:Route Segment	OWS6 data was added to provide an example of how the proposed MTT information would be encoded.
EANS	Estonia	AIXM 5.1	All AIXM 5.1 features contained in data	Estonia data was provided from Comsoft for inclusion within our AIXM 5.1 data to ensure that the services had a consistent dataset to meet the requirements of the scenarios
Weather Data				
Meteo France	Meteo France WFS1.1	WXXM1.1	• avwx:SIGMET	Sample WXXM1.1 data was obtained from the Meteo France WFS1.1 which included icing and volcanic ash SIGMET features.

Issues with data loading

Issues were encountered loading datasets which contained complex geometry types: ArcByCentrePoint, CircleByCenterPoint and geometries encoded ByReference. For example, the Special Use Airspace data received from FAA contained features containing ArcbyCentrePoint, and CirclebyCentrePoint geometries. GO Loader was unable to load these geometries as these geometries are not included in the Simple Features for SQL specification and therefore not supported by Oracle Spatial or other mainstream databases. Consequently, manual transformation steps had to be performed as the data to convert the data into supported geometry types which were then loaded into the database.

7.1.6.2 WXXM Data Loading

To demonstrate that the web services have the flexibility to support both AIXM 5.1 and WXXM 1.1, Snowflake Software also provided a WXXM WFS 2.0. However, access to

weather data was much more limited. Therefore, Snowflake harvested WXXM 1.1 data from the Meteo France WFS 1.1 which included icing and volcanic ash SIGMET features. As the data was available in WXXM 1.1, GO Loader was used to automatically generate a WXXM database schema, translate and load the WXXM 1.1 data into the main OWS-7 Oracle 11g database.

Issues with data loading

GO Loader provides the functionality to automatically configure the schema translations required to translate the data into the database schema. However, the WXXM 1.1 schema does not substitute the anyType data type for om:result property for a specified data type (e.g. the different WXXM features. This means GO Loader cannot automatically define the database schema for these properties and discards the property (Figure 10). Manual intervention was needed to configure the load and schema generation of this property.

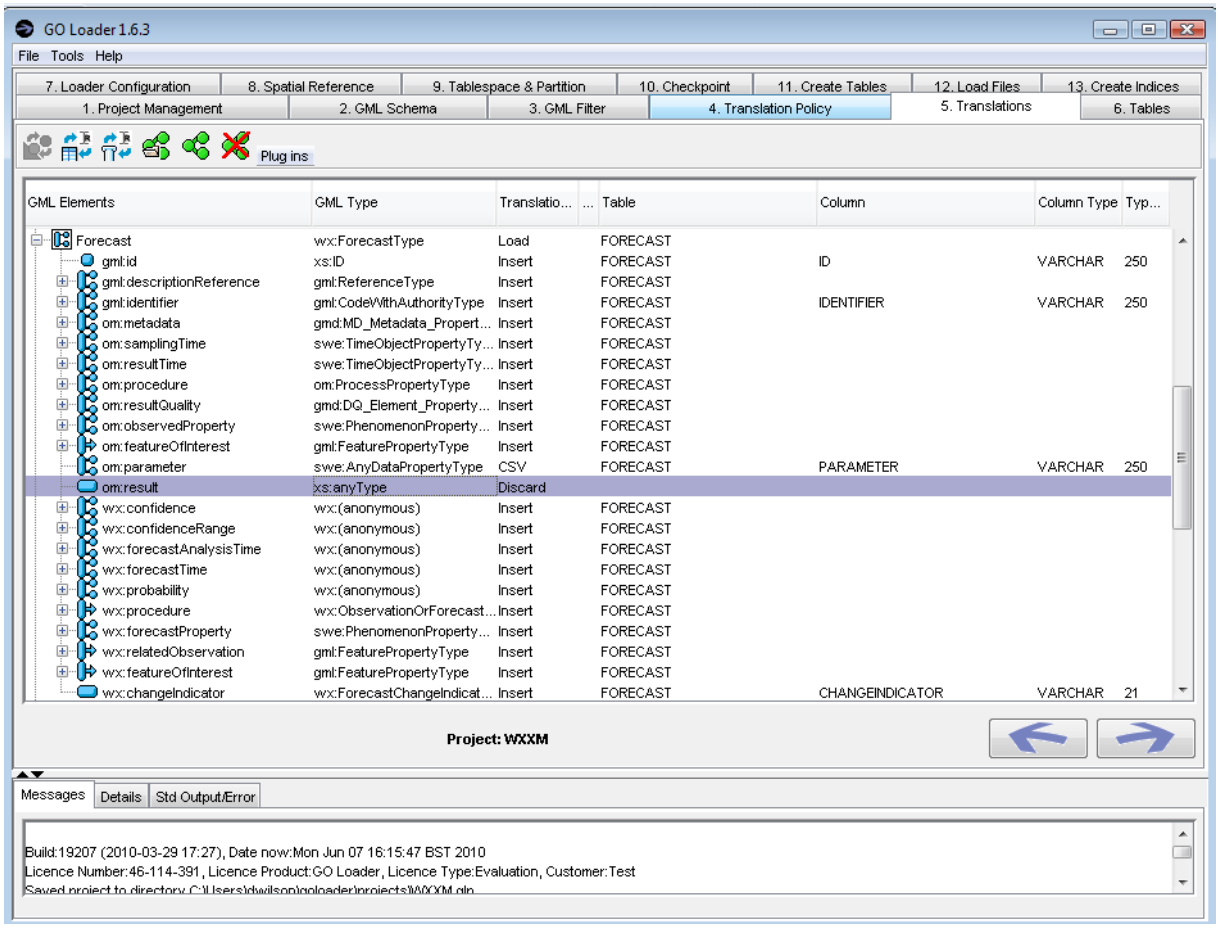

Figure 10 -Issues with om:result when loading WXXM 1.1 data

It is recommended that for future release of WXXM, that feature types extended from observation feature types should extend from specialized observation feature types classified by their result type as recommended in ISO/DIS 19156 Geographic information – *Observations and Measurements*. This will enable generic tools such as GO Loader to parse and load the result properties.

7.2 Comsoft AIXM WFS

7.2.1 Overview

One design principle of CADAS-AIM$_{DB}$ is the interoperability with other systems. As the database is the core of any integrated AIM solution an open interface that can be used independently from any platform and programming language is one of the key features. On the other side the interface has to fit the special use cases of aeronautical data without restricting the access to a limited set of pre-defined functions. For an optimal support of AIXM 5 CADAS-AIM$_{DB}$ provides the CAW-interface. However, after some modifications (see Section "Issues Encountered") also the WFS 2.0-T standard can be used for accessing AIXM 5 data.

7.2.2 WFS 2.0 conformance

Functions of the Simple WFS, the Basic WFS and the Transaction WFS conformance class are provided: Mainly two functions as described below are intended to work on the actual AIXM data:

1. The GetFeature function provides various possibilities to retrieve AIXM 5.1 features. Thereby one or more feature of the same or of different types can be selected. Selection is done by applying various filter criteria. For these filter criteria special attention has to be taken on the constraints implied by the AIXM specification. CADAS-AIM$_{DB}$ enforces these constraints when querying any AIXM data.

2. Modification of AIXM data is done by the Transaction function. For this function the special semantic of modifying AIXM data has to be considered: E.g. modification of data in AIXM 5 is done by inserting specialized Time Slice, called Correction. But on the level of database functionalities this is also an insert operation. CADAS-AIM$_{DB}$ thus restricts the operations that are available here depending on the current use case.

Automatic data locking is used for this operation, thus the client is not required to lock any data before modifying. For this reason also, no locking functions are provided by CADAS-AIM$_{DB}$.

7.2.3 Selecting AIXM 5 data with the GetFeature function

The GetFeature function is the central place to retrieve AIXM 5 compliant data in the form of Time Slices. Retrieving only certain properties of a feature is available. Thereby the correctness of the result against the AIXM schema is ensured. This means that all properties are placed in correct Time Slice data structures and that any object relations are fully expressed.

Selecting the data is done by specifying one or more filter criteria according to the AIXM constraints. All these filter criteria a expressed in the query syntax as defined by ISO 19143. All available filters can be differentiated into the following classes:

- Basic selection rules are provided by the AIXM Filter Criteria that restrict the queries to one or more AIXM feature types and that provides simple expressions to search for specific Time Slice types. This filter also provides the type Snapshot that creates an aggregation of multiple Time Slices representing a specific feature state at a point in time or during a time period (see the AIXM specification for an exact definition of Snapshot Time Slices).

- The Temporal Filter Criteria address the core of all AIXM data which is the inherent temporality of all kinds of information. These criteria provide a bi-temporal selection using either single time stamps or time periods. The bi-temporality is expressed in a Functional Time ("when are the information valid") and a Technical Time ("since when do we know about that information ").

- The Property Filter Criteria provide a simple selection on the values by arbitrary feature properties. All AIXM data types together with the respective comparison operators are supported. Property filter can be applied on the properties on the Time Slice level as well as on the Object level.

- A specialized version of the Property Filter Criteria exist for geospatial information in form, the so called Geospatial Filter Criteria. With these criteria all AIXM 5 features can be searched that have a location assigned (in the form of a point or boundary). They provide the capabilities of the proprietary geospatial database extension as standardized WFS queries to all clients and users. Geospatial comparison operators depending on the actual data type as defined in ISO 19143 are supported.

- Join Filter Criteria are used to traverse the linking between multiple AIXM 5.1 features. Thereby only references that are defined in the AIXM specification can be used. This is ensured by specifying the association that creates the link as defined in the AIXM. Only stating the feature types that should be joined is not sufficient for AIXM data and is thus not supported. When using a Join Filter Criteria always all affected features are returned by the GetFeature function.

7.2.4 Data modification with the Transaction function

The Transaction method provides all operations for manipulating the contents of CADAS-AIM$_{DB}$. Data manipulation is always done by means of AIXM data structures.

For inserting data into the database the Insert operation is provided. This operation is the standard way to insert new AIXM 5 Time Slices. With this operation Time Slices for both event types (permanent and temporary delta) can be inserted. Also corrections and cancellations can be issued. By inserting respective Time Slices, the commissioning and decommissioning of AIXM 5 features is handled too.

As AIXM 5 defines the Time Slice types Baseline and Snapshot as aggregations of multiple permanent or temporary deltas these Time Slices types cannot be inserted directly. Instead the respective deltas have to be inserted.

7.2.5 User management

All functions of the WFS interface are integrated into the user management of CADAS-AIM$_{DB}$. A fine grained configuration of user rights on the data contents and on the functions is provided:

- Defining user rights on the AIXM data: AIXM data can be classified by special tags and attributes to customize the user rights. Thus it is possible to define the rights on features sets or on single features regardless of any AIXM constraints. Particularly, it is not required to have the same rights for all features of one feature type. This assignment of user rights depending on special classification allows the use of CADAS-AIM$_{DB}$ for data of multiple organizations and or countries.

- Functional user rights: Function rights define the operation a user is allowed to use. This is independent from the data they are executed on. Functional rights are defined for each operation that is provided by the WFS interface and that operate on the actual user data. Functions that are used to retrieve meta data about the interface itself are restricted by any user rights.

The actual permission to execute a specific function or operation is evaluated by the combination of the function that is executed and the data which are affected. This allows the definition of access rights in a use-case oriented way.

On the other side an abstraction from the single users into user roles is implemented. This defines a grouping of the user corresponding to their scope of work. According to the use-case oriented view of the user management, the actual user rights are always defined on these roles. User and user roles are associated by a n to m relation, which means that each user can participate in multiple roles and that each role can have one or more users.

7.3 Meteo France WXXM WFS

7.3.1 Context

Meteo France participates to the OWS-7 with two others companies, Alticode and Atmosphere Service. The objective was to implement a full stack of interoperability from the EFB client to the WFS server through the FPS portrayal.

Meteo France WFS server should be able to serve volcanic ash SIGMET in WXXM 1.1 format which could be considered as a complex feature. Volcanic Ash SIGMETs provided by the WFS Meteo France will be ”drawn” by the Alticode FPS and handled by EFB type client, designed by Atmosphere.

7.3.2 Scope

The Meteo France WFS …

- serves WXXM 1.1 SIGMETs based on GML 3.2

- supports requests using a subset of the OGC Filter Encoding on WXXM 1.1

7.3.3 Software Architecture

Meteo France does not have its own implementation of WFS server. So they had to build an architecture on available components.

Geoserver[1] has been chosen as reference implementation of WFS 1.1. Geoserver serves simple features, so to be able to serve WXXM 1.1, we wrapped it with Cocoon[2] component which makes the translation from simple features to complex features.

Schematically, the applicative structure of the component can be seen as a simple layered one, as depicted in Figure 11:

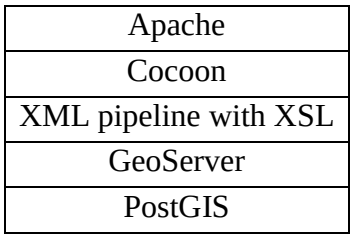

Figure 11 - Structure of the Meteo France architecture

The translation between simple features served by Geoserver to complex features as WXXM is done by XML transformation with XSL build upon the Cocoon framework. Also the filter encoding statements are transformed using the Cocoon framework.

Geoserver and PostGIS[3] are used to store volcanic ash SIGMETs and to serve them as simple features. These two components are deployed in Tomcat container, the HTTP protocol is handled by Apache.

7.3.4 Limitations

Directly linked to the constraints of the architecture, the limitations are many. The WFS of Meteo France is based on a WFS 1.1 associated with GML 3.1. It uses data based on the WXXM 1.1 format which is based on GML 3.2.

One limitation is that the Filter Encoding queries based on XPath expressions that cannot be translated in a generic way to adapt the pattern of WXXM 1.1. Thus, only a small set were taken into account.

Other limitations are reported in the OWS-7 Aviation WXXM Assessment Engineering Report (OGC 10-132).

---

[1] http://geoserver.org/

[2] http://cocoon.apache.org/

[3] http://postgis.refractions.net/

7.4 GMU Severe Weather Detection WFS

7.4.1 Background

Center for Spatial Information Science and Systems (CSISS) at George Mason University has been collaborated with scientists at NASA Goddard Space Flight Center (GSFC), University of Wisconsin, and National Severe Storms Laboratory in developing techniques and systems for the detection and tracking of satellite image features associated with extreme physical events for Sensor Web targeting observing through a NASA-funded project.

One of the research results is a prototype system which is capable of processing NOAA GOES data or NASA MODIS data, driving the underlying detection model, and generating satellite image features that are associated with extreme physical events. The front end of this system is a WFS 1.1.

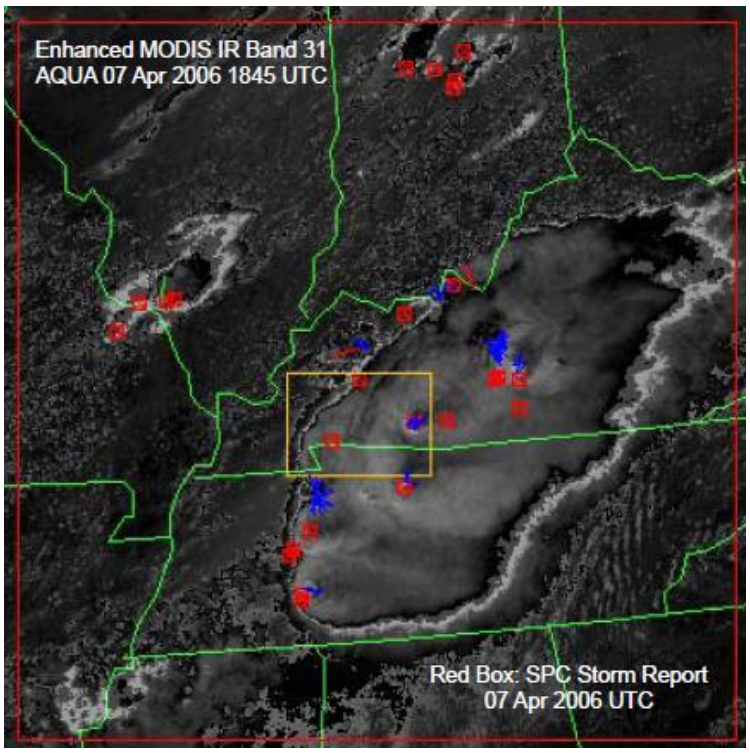

Figure 12 - MODIS Image Detection Result

As shown in Figure 12, the generated satellite image features (couplets) are shown in lines in red and green. The red box shows Storm Prediction Center storm report within 20 minutes of the image time. As we can see from this image, the generated features match-up with the store reports nicely.

In support of OWS-7 aviation thread, CSISS staffs have further wrapped this service to be compliant with WFS 2.0, and transformed the generated satellite image features into WXXM 1.1 elements.

7.4.2 Data flow

The data flow within this system begins from NOAA GOES data or NASA MODIS data, followed by detection model output, the legacy WFS output, and finally the new WFS 2.0 outputs. The generated satellite image features were encoded as sensor observation elements and now are converted into WXXM 1.1.

7.4.3 WFS 2.0 / WXXM 1.1

This service wraps the model outputs into three types of service outputs: peak point feature, sink point feature and thermal couplet feature. These features are encoded as "wx:Observation" elements, which are further wrapped by "wx:featureMember" elements. Each feature has its unique geo-location in latitude/longitude pair, and its temporal signature. This service supports GetCapabilities, DescribeFeatureType, and GetFeature operations.

7.4.4 Interaction with the OWS-7 clients

This service's output has been successfully retrieved and visualized in the Aviation client developed by Luciad. The GOES test image is shown in Figure 13, and the generated satellite image features (peak point, sink point, couplet), which are encoded as WXXM 1.1 elements, are visualized in the Luciad's Aviation Client, as shown in Figure 14, Figure 15 and Figure 16.

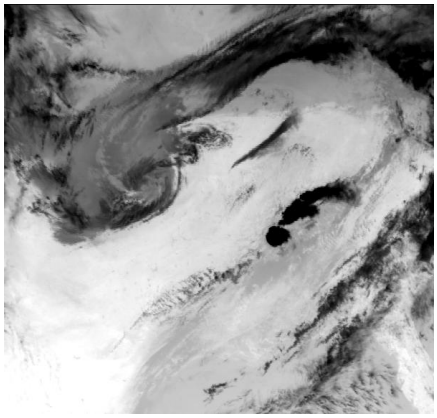

Figure 13 - GOES Test Image

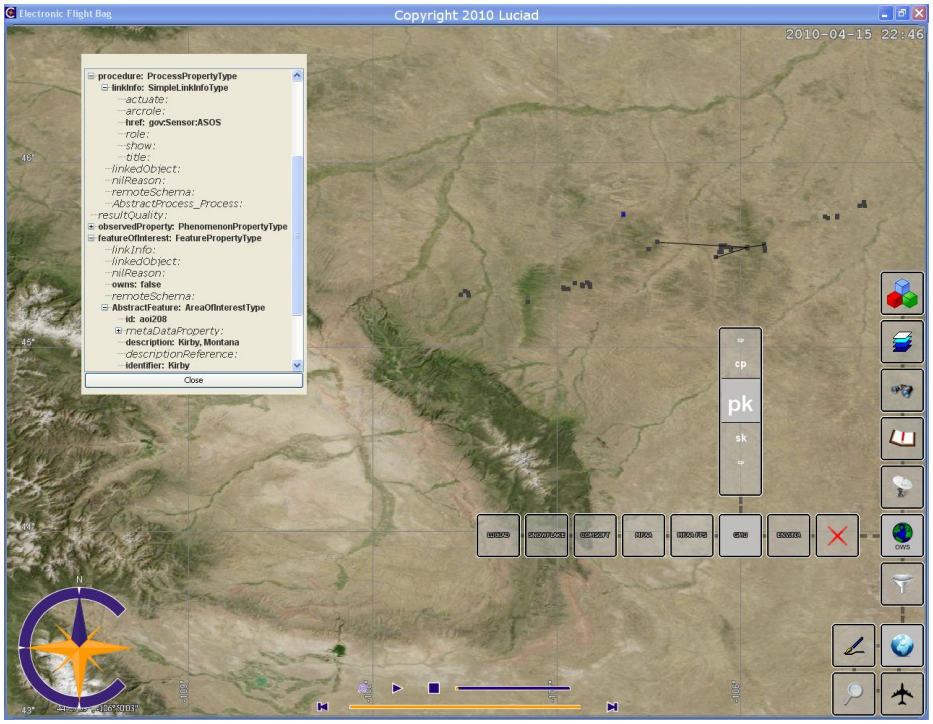

Figure 14 - WFS Outputs shown in the Luciad's Aviation Client (1)

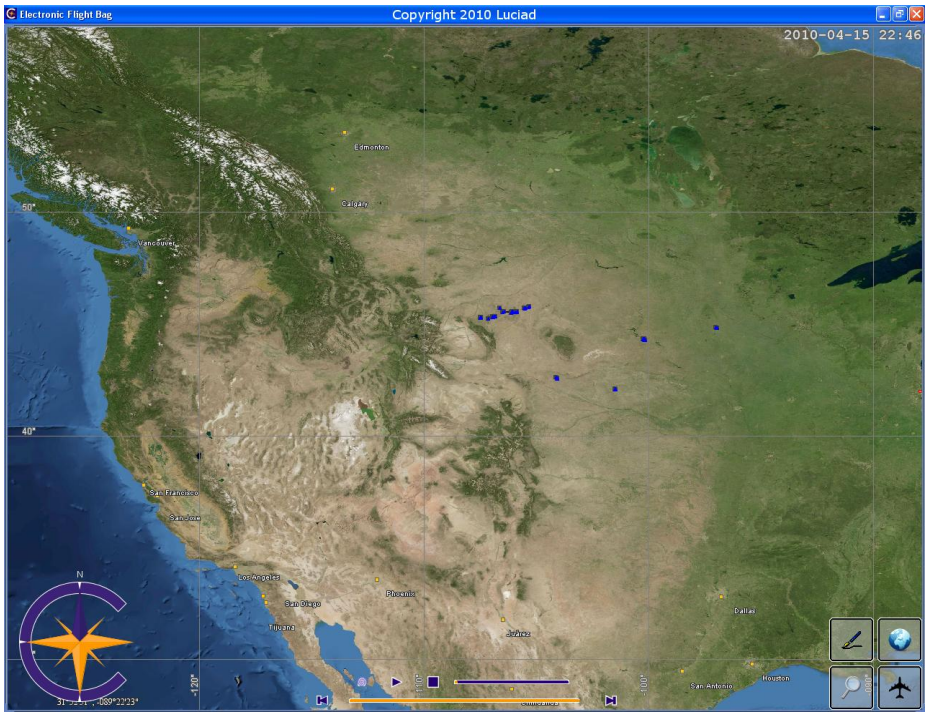

Figure 15 - WFS Outputs shown in the Luciad's Aviation Client (2)

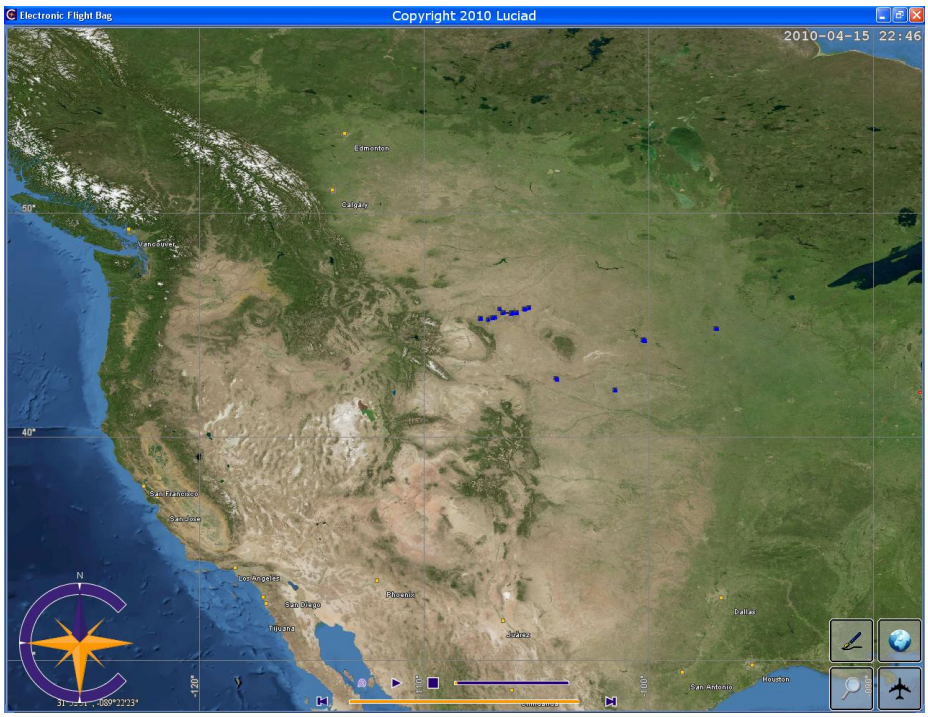

Figure 16 - WFS Outputs shown in the Luciad's Aviation Client (3)

7.4.5 Access

- Endpoint
  This service's endpoint is http://data.laits.gmu.edu:8081/ows7/wfs

- Demonstration page
  You may access the demonstration page at
  http://data.laits.gmu.edu:8081/ows7/index.html

7.5 Envitia AIXM / WXXM FPS

The general computational architecture used as the basis for Envitia FPS experimentation in OWS-7 is shown in Figure 17. This shows all of the key components of the sub-system including the registry, WFS, WCS and Feature Portrayal Service. The style editor clients created the Styled Layer Descriptors (SLD) which are then uploaded into the registry using the web browser interface. The FPS clients make a WMS request with an additional SLD parameter referencing an SLD file contained in the registry.

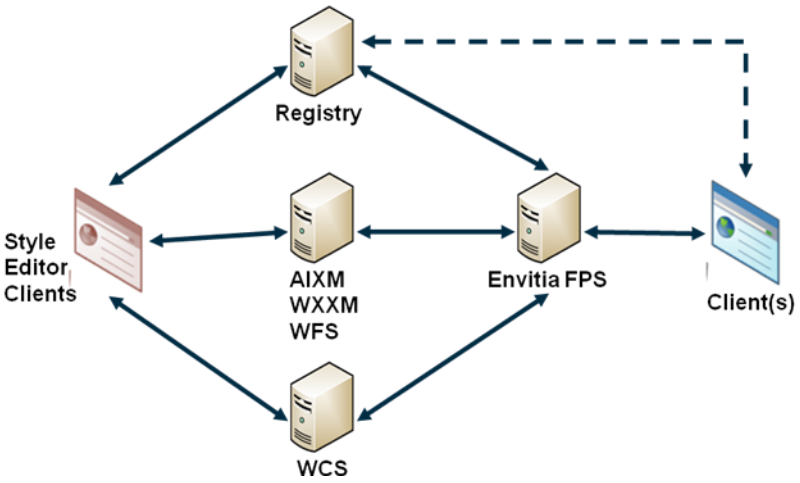

Figure 17 - Feature Portrayal Service Architecture

Specific portrayal rules were created for AIXM and WXXM features and were then stored in the registry. Hence styling rules and symbology could be directly referenced via a registry query URL, e.g.:

> *http://registry.galdosinc.com/ows7/query?request=GetRepositoryItem&id=urn:envitia:com:sld:fmawxxm*

These URLs were then supplied as the "sld" argument in the GetMap request made from the clients to the FPS, e.g.:

> *http://217.33.30.11/envitia.wms.ows7.aixm2/wms.aspx?REQUEST=GetMap&SERVICE=WMS&VERSION=1.3.0&FORMAT=image/png&BGCOLOR=0xFFFFFF&TRANSPARENT=TRUE&CRS=EPSG:4326&BBOX=-93.2,44.8,-93.1,44.9&WIDTH=968&HEIGHT=843&LAYERS=Layers.9ADDBBB9&**sld=http://registry.galdosinc.com/ows7/query?request=GetRepositoryItem&id=urn:envitia:com:sld:fmawxxm***

The FPS was configurable between component and integrated modes but in OWS-7 was acting in an integrated capacity, rather than as a component implementation (integrated and component FPS implementations are described here: OGC 05-078r4). The layers made available via the FPS were configured to match the WFS and WCS services available within the test bed. Feature data was also stored in a cache at the FPS to aid performance, avoiding a repeated WFS/WCS query for every FPS request and so reducing latency and reducing load on the WFS. In some cases (where data is volatile, the FPS was set to query the WFS every time. An alternative is to include a publish-subscribe client in the FPS so that once cached any changes in the cached WFS can be detected and the FPS kept immediately up to date. This should be the subject of experimentation in future testbeds.

7.6 Alticode WXXM FPS

7.6.1 Context

Alticode participates to the OWS-7 in the frame of a consortium led by MeteoFrance, and also comprising Atmosphere Services. The Feature Portrayal Server implemented by Alticode was thus primarily intended to make the link between the WFS provided by MeteoFrance, serving Volcanic Ash SIGMETs, and the EFB type client, designed by Atmosphere.

The FPS development was initiated “from scratch”, and has been based exclusively on the OGC standards documents (more specifically on WMS, SLD and SE specifications, whose respective references are 06-042, 05-078r4 and 05-077r4).

In spite of being targeted in priority to render polygons describing volcanic ash clouds, provided in WXXM encoded SIGMETs, no specific assumption was made on the nature of data processed when designing the FPS. Within OWS-7, Alticode's FPS should thus be able to serve both WXXM and AIXM types of data.

7.6.2 Software Architecture

The Alticode FPS' structure mirrors rather closely that of OGC standards: objects modeling Filter Encoding, Symbology Encoding and Styled Layer Description have been implemented in separate modules. Schematically, the applicative structure of the component can be seen as a simple layered one, as depicted on the diagram below (Figure 18):

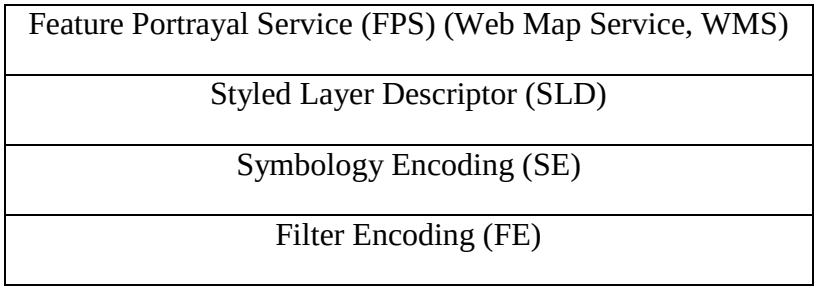

Figure 18 - Layer structure of the Alticode FPS

The FPS component is written in Python programming language[4]. The associated web site (http://www.alticode.net/OWS7/) has been built using the django framework[5], and is served by Apache.

[4] http://www.python.org/

[5] http://www.djangoproject.com/

7.6.3 Limitations

It hasn't been possible to implement a fully fledged FPS within the short OWS-7 time frame. Hence, the component is affected by a number of limitations, which are further detailed in the following subsections.

7.6.4 Scope

The scope of possibilities offered by the OGC standards referenced above was thought to be incompatible with the development schedule and has thus been voluntarily restricted. The only SLD/SE version accepted is 1.1.0 and the main functional limitations are listed below:

- SLD: layers and styles are assumed to be defined by the clients, respectively via UserLayer and UserStyle elements (NamedLayer and NamedStyle elements are silently ignored);

- SE: only two of the five possible symbolizers are currently implemented (at the time of document writing), namely PolygonSymbolizer and PointSymbolizer;

- FE: filtering capabilities based on spatial operators, object identifiers and functions are not yet implemented (at the time of document writing)

7.6.5 Efficiency

From the client's perspective, the FPS efficiency is mostly characterized by the latency (delay before the requested map is effectively rendered), which should be as short as possible. The decision to try and implement a fully flexible “component server” (as opposed to integrated ones, see SLD specification) would not have been made if short response times had been considered as a priority. In other words, the chosen architecture (no cache, no a priori knowledge of any WFS) is known to be incompatible with optimized response times.

7.6.6 Robustness

The FPS robustness can be challenged in (at least) two ways:

- Through the large variety of options made possible by the standards (an extensive test campaign was not compatible with the project's tight schedule);

- By submitting it to heavy loads (high rate of client requests). The component developed for OWS-7 was meant to be a prototype, not an industrial product. Since it was not a targeted objective, the capacity to sustain heavy loads has not been taken into account in the component development – for which it can become a dimensioning factor.

7.7 Galdos INdicio™ Registry

INdicio™ is a Web Registry that implements the Open Geospatial Consortium (OGC) Web Registry Service specification (also known as the Catalogue Service for the Web). Unlike conventional geographic catalogues, INdicio™ is highly configurable and can be readily deployed to manage a wide variety of objects.

7.7.1 Key features of INdicio™

- Open standards-based
- Easy installation and configuration
- Easy management and monitoring capabilities
- Secure and flexible interface
- Supports multilingual application domains
- Support for large transactions

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

- Services taxonomy (source: ISO 19119 “Geographic information – Services”)
- 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)
- Feature codes (source: DGIWG FDD)
- Property categories based on Dublin Core (source: DCMI metadata terms)

INdicio™ is inherently extensible and can be adapted to multiple application requirements by using OGC CSW layered on top of ebRIM. Galdos Systems Inc. is constantly developing new extension packages, such as the CRS Extension Package, as well as custom extension packages on request.

The Flexible and secure query interface:

- Supports spatial queries using GML 3.1.1
- Supports XACML specification
- Supports role-based access control
- Full text search of XML resources

- Extensible harvest
- Customizable life cycle management
- Output transformations facilitate rapid web services integration

7.7.2 Technical summary

- Implementation of OGC Cat 2.0.2, CSW-ebRIM 1.0.1
- OGC Filter Expressions (v1.1.0)
- PostgreSQL 8.4, Oracle 11g, Oracle Express Edition (XE) 10g
- Java JDK 1.6
- Windows 2003
- Red Hat Enterprise Linux 4.0
- Apache Tomcat 6x Web Application Server

7.7.3 Supported specifications

- OGC CSW-ebRIM 1.0.1
- OGC Catalogue 2.0.2 / CSW
- OASIS ebRIM 3.0
- OGC Filter 1.1.0
- GML 3.1.1
- OWS Common 1.0

7.8 52°North Event Service

The Event Service used in the OWS-7 Aviation scenarios is based on the 52 North implementation of the OGC Sensor Event Service discussion paper that was also used as a basis for the OWS-6 AIM Event Service.

7.8.1 Service Description

As the OWS-7 Aviation Event Service is based on the OGC Sensor Event Service (SES) specification it uses the same communication protocol and relies on the same standards. All communication with the Event Service is done via SOAP messages transmitted using HTTP.

On top of that the Event Service implements operations defined by the OASIS Web Services Notification (WS-N) specifications. The mandatory operations are:

- Notify: This is the basic method to send notifications (e.g. digital NOTAMs) from a notification producer to the Event Service and from the Event Service to a notification consumer (e.g. the clients). In contrast to the other methods no request-response communication is used here, all notifications are pushed instead.

- Subscribe: Using this operation a notification consumer can express its interest in notifications. Such a subscription may include a filter statement to specify the sub set of notifications of interest.

- GetCurrentMessage: This method allows requesting the last message that was posted to a specific topic.

- RegisterPublisher: This method is used to register a notification producer at the Event Service. This method is not mandatory to be used with the OWS-7 Aviation Event Service, however it is necessary if the DescribeSensor operation (see below) shall return complete and useful information.

In addition the following optional operations are implemented:

- Renew: This method allows renewing a subscription in a sense that its termination time is changed to a later one.

- Unsubscribe: This method cancels a specific subscription.

In addition to these methods defined by OASIS WS-N the Event Service supports some operations that are common to OGC and OGC SWE services:

- GetCapabilities: The GetCapabilities operation is common to the OGC services and returning the service metadata and capabilities like the supported operation and filter languages.

- DescribeSensor: This method is common to OGC SWE services. It is used to provide more detailed information about sensors that are used by the service. It is inherited form the SES specification. In case of the common event service this operation can be used to access information about the registered notification producers (see RegisterPublisher) which may be sensors. With respect to the attempt to build an Event Service specification for all areas of the OGC started in the OWS-6 and -7 eventing activities this method will most likely be renamed to a more general name.

Further important standards used by the Event Service cover the encoding of the filter statements used in the subscriptions. The OWS-7 Aviation Event Service supports two languages, however only the first one is used in the demonstrated scenarios. The languages are:

- The OGC Filter Encoding Specification (FES) 2.0: This filter language was originally designed for the use in requests to OGC Web Feature Services (WFS). Due to its generic design it can be applied on various placed including filtering of events. The language defines the encoding of the most important comparison, spatial and temporal operators as well as logical links and functional expressions.

- The OGC Event Pattern Markup Language (EML): The EML is a language to encode event patterns for Complex Event Processing (CEP) and Event Stream Processing (ESP). Using such event patterns it is possible to define rules how new information shall be derived from a stream of events instead of just filtering it. The EML is currently available as a discussion paper and not used in the OWS-7 Aviation scenarios.

7.8.2 Changes to the Service

Compared to its source the Sensor Event Service (SES) the Event Service for Aviation is changed in some places. On the one hand the sensor specific operations like DescribeSensor are no longer supported. Possible ways of providing metadata about the available events that are applicable not only in the SWE domain are discussed in the OWS-7 Event Architecture ER (10-060).

Also the supported input formats are extended. The pure SES does only support notifications encoded as Observations & Measurements or as EML Events which is an experimental format for the encoding of complex events. For the Aviation version of the Event Service the support for the Aeronautical Information Exchange Model (AIXM) and the Weather Information Exchange Model (WXXM) were added.

Compared to the version provided for OWS-6 there are two major changes. One is the previous mentioned support for WXXM which was not available in OWS-6. Thus, now it is possible to serve notifications on aeronautical features and weather information from a single service. The other change is related to the subscription handling inside the Event Service. The lifecycle management for the subscriptions allows setting a lifetime for each subscription. If the time runs out of a subscription it is automatically removed. To prevent subscriptions running out that are needed longer than initially it is possible to extend the validity period via the Renew operation. It is also possible to manually remove an existing subscription via the Unsubscribe operation.

All these features described above are demonstrated in the OWS-7 Aviation scenarios including the use of multiple event encodings, multiple concurrent subscriptions, subscription lifetime handling and use of the Unsubscribe operation.

7.9 Frequentis Dispatch Aviation Client and EFB Client

This section describes the technical solution for the Dispatch Aviation client and EFB (Electronic Flight Bag) client developed by Frequentis within the OWS-7 testbed.

7.9.1 Component functions

The OWS-7-Aviation Client (AC) component provides pre-flight and on-flight assistance functions for flight dispatchers and pilots. The pilot can use this component for gathering and delivery most up-to date information concerning his/her flight.

The implementation scope for such a component is generally wide, but this implementation focused mainly on the parts that interface Web Services providing AIXM, WXXM and graphical data. This implementation did not focus on areas like: Flight Plan Management (organizing and assigning flight plans), Aircraft Data (for fuelling calculations etc.), User Management (pilots and dispatchers as users of OWS-7-AC) or Security (which needs to be dealt-with cross-thread the OWS-7 testbed).

The OWS-7-AC component focuses on implementing User Interface (UI) functions enabling the user to search aeronautical and weather data concerning the flight. Both pilot and dispatcher can use OWS7-AC functions to find data, which are provided by WFS (Web Feature Services) for AIXM and WXXM data. The graphical component connects to FPS server(s) to present data graphically. Both - the dispatcher and the pilot can subscribe to the Event Service for data updates relevant to the flight or to areas, which the user is specifically interested in. The Event-Consumer module of the OWS-7-AC component receives data from the Event Service based on the subscriptions and presents them to the users (both pilot and dispatcher).

Within the OWS-7 testbed several functions (e.g.: AIXM WFS) have been implemented by more than one contributors. The OWS-7-AC dealt with connecting to multiple data providers (offering either different scope or different format versions of data).

When implementing the OWS-7-AC, many interesting questions appeared. The following three conceptual topics are indicating the need for further analysis and discussion:

1. Communication between Pilot and Dispatcher
   The Frequentis OWS-7-AC component implementation has not dealt with Pilot and Dispatcher as separate clients to WFS or Event service. But the Flight entity itself (where dispatcher and the pilot contributes on pre-flight and on-flight data assistance) acts here as the client to the WFS and Event Service. Any data subscriptions created by the dispatcher (on a specific Flight session) cause Event data retrieval by the pilot (on the same Flight session) and the other way around. This concept assures that both parts have the same information.

2. Caching data on client
   When an event (data change) arrives at the client, it is displayed at the first place (graphically or textually). But also any local (cached) data stored on client side and affected by this event should be either updated or released from the local client cache (and consequently next time forcing the data (in time slice of the incoming event) to be reloaded from the WFS).

3. Scope of data coverage by Services
   Different data providers cover different scope of data (e.g.: In this project

Snowflake covers U.S. part, Comsoft covers parts of Europe). The OWS-7-AC component implementation should ideally make the data providers transparent for its users. The current implementation needs the user to select the data provider first (as their data coverage is not known generally). A central WFS, which would have the coverage information could route requests to specific WFS servers and responses back to the clients.

7.9.2 Component architecture

In the client-server architecture the OWS-7-AC component is the client to WFS, FPS and Event Service. The core of the OWS-7-AC component is a Web Server. Users connect to the OWS-7-AC Web Server using a Web Browser and access data from the WFS, FPS and Event Service.

The Event-Consumer component of the OWS-7-AC Web Server listens for incoming events. These are sent by Event Service to the OWS-7-AC Web Server based on subscriptions made by the user.

7.9.3 Technologies used

The Web Server is running on RedHat Linux OS in Apache Tomcat Web Server container. The application framework has been implemented in Spring Framework. The key software components interfacing indirectly the WFS components are the RestClient and SoapClient, both implemented using existing Spring components of the framework. This proved the Spring framework to be very suitable for the client (and the server interface) parts.

The client technology (web browser interface) stands on AJAX components communicating asynchronously with the web server. The map component for graphical presentation of data was built on top of OpenLayers libraries providing API interface to WMS (Web Map Server) clients.

7.10 Luciad EFB Client

7.10.1 Luciad introduction

Luciad is a software product company providing standards-based software components for high performance visualization in the ATC/ATM, Defense, and Security areas.

Luciad closely follows open standards such as the geospatial standards defined by the OGC. Luciad has applied these standards in its flagship product LuciadMap which is used in numerous projects for customers including Eurocontrol, FAA, NATO and large system integrators.

Last year, Luciad was also actively involved in the OGC OWS-6 initiative, in which an Electronic Flight Bag (EFB) client was developed within the AIM thread. This client was successfully demonstrated during the different TIEs and during the AIXM Users Conference organized by sponsor organizations Eurocontrol and FAA.

7.10.2 Software Architecture

For the OWS-7 Aviation thread, Luciad contributed an Aviation client application, with focus on Electronic Flight Bag (EFB) functionality but also equipped with some aspects of a Flight Dispatch application.

A schema of the application's architecture is shown in Figure 19.

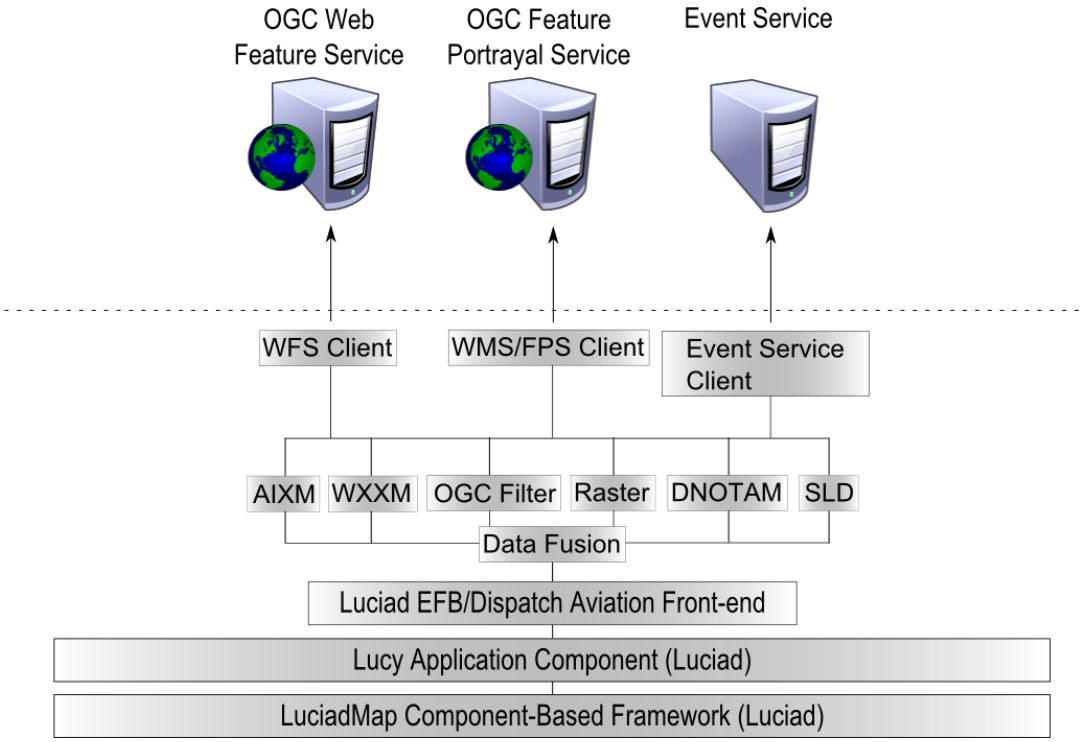

Figure 19 - Luciad EFB and Dispatch Aviation Client

Client characteristics:

- Integrated WFS client, supporting versions 1.0.0, 1.1.0, and 2.0. The client supports the WFS basic and transactional profiles.
    - Support for OGC Filter Encoding 1.0, 1.1 and 2.0.
    - A comprehensive user interface to create relevant filters according to the scenario is provided in the client. For example, to filter on objects falling within a given distance from the flight plan, and complying with temporal and other non-spatial constraints.
- Integrated WMS client, supporting versions 1.1.0, 1.1.1 and 1.3.0. The client supports the WMS basic, queryable, and SLD profiles.
- Integrated WCS client, supporting version 1.0.0.
- Integrated client to communicate with an Event Notification service

  - Subscription of a consumer with support for complex filtering.
  - Unsubscription of a consumer.
  - Listening to events by a subscribed consumer.

- Wide range of data format support, including
  - AIXM 5.0 & 5.1 for aeronautical data
  - WXXM 1.1.1 for weather data
  - Additional aeronautical and weather data formats like AIXM 3.3/4.0/4.5, ARINC 424, DAFIF and GRIB.
  - Raster format support (GeoTIFF, TIFF, JPEG, JPEG 2000, PNG, GIF, ECW, MrSID, CADRG/ADRG/USRP, DTED, USGS DEM …) for satellite imagery and elevation background data.
  - Vector format support (KML 2.2, ESRI Shape, MapInfo MIF/MAP, GML 2/3.1.1/3.2.1, SVG …) for vector-based background data.
  - 3D format support (OBJ, OpenFlight …) for data such as 3D terrain and 3D buildings.
- Digital NOTAM and weather event support for aeronautical/weather information updates
- Support for METARs, TAFs, SIGMETs and PIREPs for weather observations
- Visualization of aeronautical and weather data according to ICAO guidelines
- Support for visualization in 2D and 3D for all data sources
- Capability of rendering feature data with an OGC Styled Layer Descriptor (SLD) both locally and remotely through a Feature Portrayal Service (see below)

7.10.3 Usage of a Feature Portrayal Service

The client allows connecting with a Feature Portrayal Service to remotely render feature data using an OGC Styled Layer Descriptor (SLD). Because the client has direct support for SLD and also has an interface to WFS, it also supports local rendering of feature data – thus mimicking the behavior of a Feature Portrayal Service locally at the client.

7.10.4 Technical Notes

All development is done in Java, using the Java Development Kit (JDK) 1.5. The client application is developed on top of Luciad's core products LuciadMap and Lucy. LuciadMap is a suite of software components for application development with focus on

high performance visualization and situational awareness. Lucy is a high-level application component for rapid application development, providing all LuciadMap components in one configurable, extendable application.

The software runs on any operating system for which a Java Virtual Machine 1.5 or higher exists. For the 3D visualization, a graphics card with support for OpenGL 1.2 or higher is required.

7.11 Atmosphere WXXM Handheld Client

7.11.1 Context

Atmosphere participated to the OWS-7 Testbed in the frame of a consortium led by MeteoFrance, along with Alticode. The main purpose of what was called the MFAA consortium was to implement an end to end service from MeteoFrance WXXM WFS to the Atmosphere handheld client through the portrayal service (FPS) implemented by Alticode. In that context, the features meant to be provided, portrayed and eventually displayed by the handheld client were Volcanic Ash features.

As much as possible, the handheld client was supposed to be able to connect with other services implement by the OWS-7 participants. The handheld client architecture has been developed based on the open OpenLayers library, initially designed to interact with WMSs and to display maps, layers and features in a user-friendly way. To fit in the end to end chain mentioned above, modifications have been operated on OpenLayers for the client to interact with FPSs.

OGC standards documents used are WMS and SLD specifications (06-042, 05-078r4).

7.11.2 Software Architecture

The handheld client, as being a first try for implementing OGC standards, has been developed has a Web Application. It is based on the following libraries:

- OpenLayers
- Ext JS (now Sencha)
- GeoExt

OpenLayers is a set of java script libraries that provide all the structure for data retrieving and queries manipulations. OpenLayers is OGC compliant. However, in order to interact with FPSs, additional modules had to be developed. These additional modules are based on WMS specification and the WMS.js module integrated in OpenLayers.

ExtJS and GeoExt are a set of libraries also written in java script. Their role is mostly to present the handheld client in an as user-friendly way as possible. All the graphical design of the application is rendered thanks to both these libraries. The entire project has been developed in java script.

The client is available at the following URL:
http://galbn1.atmosphere.aero

7.11.3 Limitations

Atmosphere, as a newcomer and due to the short OWS-7 time frame and the steep learning curve, decided to develop the handheld client as a web application. Hence, the component is affected by a number of limitations: Due to its nature, the handheld client was unable to interact with the Event Service without having to develop an intermediate proxy (between the client and the Event Service).

The main objective (that is, showing an implementation of an interoperable service between the MeteoFrance WMS, the Alticode FPS and the Atmosphere client) has been completed, yet all the possibilities of the client have not been explored (e.g. connection to an Event Service or implementation of SOAP). With the acquired knowledge of the OGC standards, a "rich client" as an entity independent of any browser could be envisaged for coming Testbeds.

As described in the WXXM ER, the MFAA has encountered difficulties to quickly understand the standard through the given examples of the WXXM use. This is partly due to the fact that examples are very basic (which is good for the first steps of understanding) and that there is a lack of more complex examples that would allow to fully understand most of the possibilities given by WXXM. A set of more complex examples would be of great help in coming Testbeds, for newcomers and for previously participating members such as MFAA.

1.1.4 Scope

The Handheld client has a connection to the following OGC participants:

- Alticode FPS
- Envitia FPS/WMS

It is retrieving Data from AIXM and WXXM servers.

Possibilities to filter on the following parameters are available:

- Valid time (begin and end)
- Flight level (bottom and top)
- Feature type (SIGMETs : Volcanic Ash, Icing and Turbulence)

7.11.4 Perspective of evolution

Atmosphere as part of MFAA, for its first participation, developed a light client mostly to show that the standard was quite easy to understand and use, even in just a few weeks. However, understanding the WXXM standard took time, and thus the client has not been

developed as far as it could have been (see the limitations section). Further participation could allow MFAA to go deeper into the standard and implement more of its possibilities.

A possible option could be to integrate the FPS service and the client, for bandwidth optimization for instance (exchange XML/GML instead of pictures).

An integrated SLD creator would also be an interesting possibility, to enhance potential customization of the data for the end user.

8 Aviation Event Architecture

This section of the aviation architecture describes the dynamic aspects in the aviation scenario based on the general concept of events and publish-subscribe interaction models. The aviation scenario makes use of the general publish-subscribe model, where notifications are sent to receivers in case that previously defined filter criteria are matched by new or updated data. During the preparation phase as well as during the entire duration of the flight, flight dispatchers and pilots can subscribe to a web services based notification service to receive specified changes, in particular additions, to aeronautical and weather databases at the time they are made. Subscriptions are modified or cancelled during the flight as the aircraft proceeds on its route.

The challenges arise from the integration and processing of aeronautical and weather information models and encodings, namely AIXM (version 5.1) and WXXM (version 1.1.1) and the realization of the scenarios compliant to the general OGC Event Architecture (see OGC 10-060).

8.1 AIM Specifics: Event Encodings

8.1.1 AIXM

The AIXM extension for AIXM version 5.1 contains the model and XML encoding for aviation events. The base model is described in more detail in OGC 10-131 OWS-7 Aviation – AIXM Assessment Report.

8.1.1.1 AIXM 5.1 basic message / dnotam event

In this section several aspects are discussed related to the encoding of Aviation events.

The AIXM 5.1 model that was tested in OWS-7 defines two types that can be used to provide information on feature changes: the AIXMBasicMessage and the Event. The former is defined in the core AIXM 5.1 package while the latter was provided as part of the xnotam draft extension package.

Both the AIXMBasicMessage and the Event message derive from the same base type, the AbstractAIXMMessageType. In addition, both are modeled as explicit features, so they can be identified. However, an Event adds some more specific properties that are relevant from an Event Architecture point of view. These will be discussed later on. In the testbed only the Event type was used. Thus the following discussion concentrates on this data type.

Common (though optional) properties of AIXM messages are a sequence number as well as message metadata. In addition, names as well as an identifier, description and boundary may be provided. These properties are now discussed in more detail.

8.1.1.1.1 Sequence Number

Conversations during the testbed showed that the sequenceNumber property in an AIXM message was designed to support traceability of messages.

An example was provided in which a series of Events is issued. If the receiver of these events only gets Events n.1 and n.3, then the gap in the message sequence (Event n.2) can be detected.

This use of the sequenceNumber property is appropriate to identify gaps in the sequence of all Events that are generated by an event source. This can for example be a WFS. Being able to support this exact functionality was not relevant in the use cases tested in OWS-6 and OWS-7, however, as most clients were interested in specific events that fulfilled their subscription's filter criteria.

If the sequenceNumber is only intended to be used for identifying gaps in a transmission of an Event sequence between two system entities, then it seems to be more appropriate to leverage functionality of the transport mechanism or the protocol used to convey the events to achieve the same goal. For example, WS-ReliableMessaging is designed to reliably transmit a sequence of messages from one system entity to another. The technology plugs in to the SOAP protocol and uses automatically added sequence numbers and an acknowledgement mechanism to ensure that all messages are delivered. Various quality levels are covered by the specification, for example to avoid duplicate delivery. A future revision of AIXM may consider deprecating or removing the sequenceNumber in AIXM messages or clarifying the intended semantics.

8.1.1.1.2 Message Metadata

Each AIXM message may provide metadata according to ISO 19115 and encoded following ISO 19139.

As defined in the OWS-6 (SWE) Event Architecture ER (OGC 09-032), any feature can be considered as an event as long as it supports the provision of the time when the happening represented by that feature took place.

As each AIXM message has one or more members and as a dnotam Event provides a selection of time slice information for these features, such an Event can be considered to itself be an indicative event. As it indicates that (potentially multiple) other events have happened, we could also call it a summary event.

If the dnotam Event type itself is considered to be an event, it has to provide an event time. The most appropriate property to provide the required semantics seems to be the date that is buried deep in the message's metadata structure (Event/messageMetadata/identificationInfo/citation/date). This date can – using the appropriate code – give the time when the resource, in this case the dnotam Event, was brought into existence. A different time, the issue time, may be provided as well (adding another value to the same date property). The creation and issue time in most cases will be the same.

Another property of a message's metadata, the dateStamp, should be treated with care. It does not necessarily represent the event time of the dnotam Event. It rather provides the date that the metadata itself was created – which may be different in case that the metadata of an Event was added to it later on.

These considerations are important from the Event Architecture point of view. In practice, the content of a dnotam Event is of primary interest. There, the time slices of features are the most important information. The event time of a time slice depends upon the type of time slice at hand.

8.1.1.1.3 Identifier and Names

A dnotam Event has properties that can be used to provide a name to the event, identify and describe it. The following observations can be made.

An Event may provide values for specific textual name and description properties. As the dnotam Event is derived from the aixm:AbstractAIXMFeatureBaseType it inherits the gml:description and gml:name properties from that data type.

There are subtle encoding differences. The dnotam properties consist of a string value with size limitations. They are nillable and can therefore contain an optional nil reason description. The gml:description is not nillable but may be empty and also contain a nil reason. A gml:name has an optional codeSpace and is not nillable as well.

A revision of the AIXM model and encoding may remove or clarify this duplication of information.

A dnotam Event may also have identity. Providing identity for an Event is useful for example when an Event that was sent in error needs to be cancelled and therefore referenced. The gml:identifier property of an Event should be used to contain the identifier value.

In the testbed the use of the gml:id attribute in an Event was also discussed. GML itself only requires this attribute to be unique in one XML instance and AIXM does not seem to enforce a stricter requirement.

Using the gml:id to provide an Event's identity came up because the WFS requires that the value of this property is unique for all the features it governs. Because this requirement does not apply to all services and clients in an Event Architecture, it is only valid for a WFS. It is therefore not safe to assume that the gml:id can be used as identifier for Event features in all cases. Especially when dnotam Events are transmitted not only between two but across a network of entities, the gml:identifier is a better place to put an Event's identifier value. Agreement upon this mechanism is primarily important to support clients in querying – not referencing – Events via their known identifier value.

8.1.1.1.4 Spatio-Temporal Extent / Boundary

In OWS-6 the boundary of a feature was used to perform spatial filtering. The approach, although not particularly precise, was used in OWS-7 as well. Sections 8.1.1.3 and 8.4

discuss the issues encountered and ideas developed for performing spatial filtering in more detail. This section shortly outlines the problem of computing the boundary for any given feature instance.

The boundary property of aviation features defines the spatio or spatio-temporal extent of this feature. This is an optional property of any GML feature. It is up to the entity that creates the feature to populate the property with an appropriate value. In OWS-7, it was difficult for service providers to do so as the rules to get or compute the spatial extent for any aviation feature were not readily available. The definition of such rules as well as a specification how to encode them in a machine readable way would be a useful enhancement that benefits the whole OGC community. Services could leverage such descriptions – for example contained in feature type descriptions – and compute the boundary of a given feature instance on the fly without requiring intervention by a service administrator.

8.1.1.1.5 Previous Event

Whenever a dnotam Event is updated or replaced, a new Event is published. This new Event will point to the old one in its previousEvent property. The purpose of this property therefore is to support a form of Event history.

Consider the following example: an airspace is announced to be closed between 12:00 and 15:00 o'clock. At 14:00 o'clock the decision is made to re-open it. In that case, a new "correction" event is sent, which replaces the previous one.

The previousEvent property is useful in supporting above use case. If the relationship between a given Event and its previous Event may have different semantics in the aviation domain, the dnotam Event model should be extended to indicate the exact semantics to avoid the need to infer the semantics. Table 1 in the OWS-6 SWE Event Architecture ER (OGC 09-032) lists some possible relationships between events (cause, initiation, perpetuation (or facilitation), hindrance (or blocking), termination, supersedes); Table 2 in that report lists an additional one. CodeLists can be defined to be used in identifying such relationships. These CodeLists may also be defined in and reused from a more general Event Model (see OGC 10-060, section 12.2.4).

8.1.1.1.6 Event Type

A dnotam Event has a property (Event/properties/type) to indicate the type of changes that the event contains information about. The codes used in this property are not comparable to those used in the interpretation property of a time slice, but they are related. This is important to understand the dnotam Event model.

The following list explains the code values allowed in an Event's type property:

- TEMPORARY: for temporary situations (example: a snow contamination) – a client can expect to have TEMPDELTA time slices that will update some existing baseline information

- TRANSIENT: a client can expect to have new, ad-hoc features with a limited lifetime

- PERMANENT: a client can expect to have PERMDELTA time slices that will update some existing baseline information

8.1.1.2 Correction of Event Information

In domains where estimates or forecasts are published to inform clients about the future state of a feature (e.g. a runway or airspace), it is necessary to have a mechanism to revoke or change such messages. Section 8.1.1.1.5 contains a good example where this is useful. In order to unambiguously identify the information that is being updated, it has to be identifiable. This can happen by stating the identifier of for example a previously sent event. How to identify a dnotam Event has been discussed in section 8.1.1.1.3. This can also happen by referencing the previously sent event. The dnotam Event supports this through the previousEvent property, discussed in section 8.1.1.1.5.

Time slices as defined in the AIXM model have a correctionNumber property. This is used to unambiguously identify any correction that was made to a given time slice. The time slices of a feature can be identified through the combination of the sequenceNumber and the optional correctionNumber.

When a dnotam Event needs to be updated, it can currently only happen by pointing to it from the new Event. Different update semantics also cannot be communicated (see section 8.1.1.1.5). Furthermore, it is only possible to point to one specific Event that is updated, not a complete list of event revisions.

It would be useful to add a correction or revision number to a dnotam Event. An Event can thus have one fixed identifier which – in combination with the revision number – can be used to uniquely identify one specific dnotam Event. Moreover, one can easily identify all revisions of a given Event because they will all have the same identifier value – only the revision number is different.

The OGC should consider this mechanism for any GML feature as it is applicable for other domains as well (e.g. the Sensor Web domain). This could be achieved by adding a revision number property to the standard GML object properties.

8.1.1.3 Reverse Associations Extension for Event Filtering

This section explains how the reverse association extension can help with filtering of events. It also shortly discusses approaches to better model these associations.

What is the purpose of a reverse association? First of all, such an association is an extension that can be added to a given AIXM feature in its extension property. It makes an association navigable in both directions if it is usually only navigable in one direction.

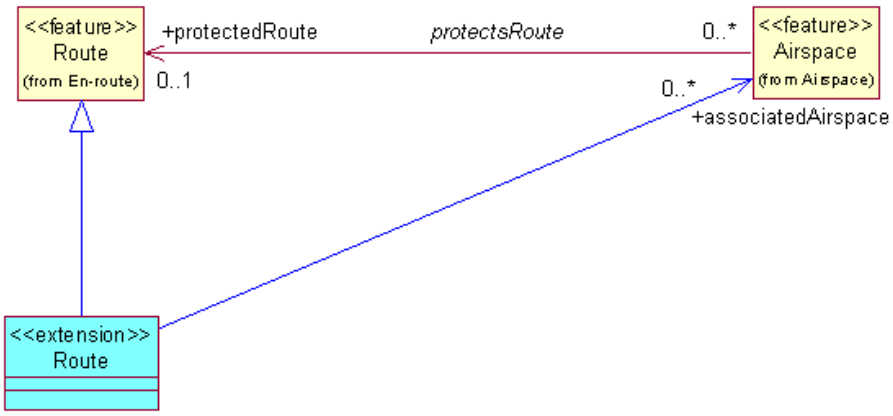

Figure 20 - Model of a Reverse Association from Route to Airspace

The figure above (Figure 20) shows the directed protectsRoute association. It goes from the Airspace feature type (being the association's source) to the Route feature type (being the target). The reverse association extension (modeled in blue) allows a Route to point to the Airspace(s) that it is protected by if that information is necessary. Associations to any class can be added in this way. Figure 21 shows the XML schema of the reverse association extension that was available in OWS-7.

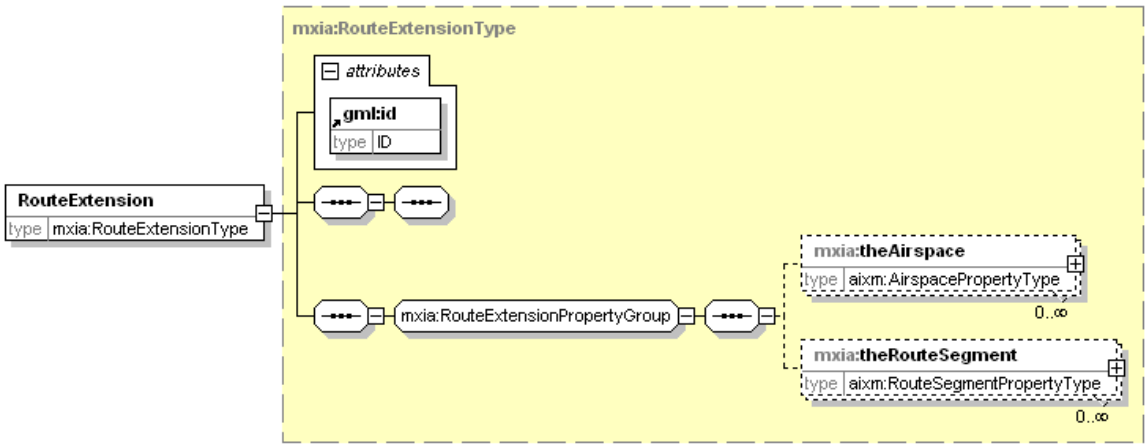

Figure 21 - Schema for a Route's Reverse Associations Extension

As shown in Figure 20 a Route may point to several Airspace elements. In addition, it may point to the RouteSegment elements that constitute a Route (this is not shown in Figure 20). The following listing shows how reverse associations can be integrated in a Route instance, using the extension schema element shown in Figure 21.

```
<dnotam:Event xmlns:dnotam="http://www.aixm.aero/schema/5.1/dnotam" …
gml:id="uniqueid">
   <dnotam:properties>
      <dnotam:EventProperties gml:id="ID_010">
         <dnotam:name>DONLON_EVENT</dnotam:name>
         <dnotam:description>Example of Digital NOTAM
Event</dnotam:description>
         <dnotam:type>TEMPORARY</dnotam:type>
       </dnotam:EventProperties>
   </dnotam:properties>
  <dnotam:hasMember>
    <aixm:Route gml:id="rtsg211">
      <gml:identifier
codeSpace="http://www.aixm.aero/schema/5.1/example">a14a8751-5428-46bc-
a2d1-32ef84d37b5c</gml:identifier>
      <aixm:timeSlice>
        <aixm:RouteTimeSlice gml:id="rts211">
          <gml:validTime>
            <gml:TimePeriod gml:id="vtSTAR224077">
              <gml:beginPosition>2010-07-
20T00:00:00</gml:beginPosition>
              <gml:endPosition>2010-07-20T12:00:00</gml:endPosition>
            </gml:TimePeriod>
          </gml:validTime>
          <aixm:interpretation>TEMPDELTA</aixm:interpretation>
          <aixm:sequenceNumber>3</aixm:sequenceNumber>
          <aixm:featureLifetime>
            <gml:TimePeriod gml:id="ltSTAR224077">
              <gml:beginPosition>2009-01-
01T00:00:00.000</gml:beginPosition>
              <gml:endPosition indeterminatePosition="unknown"/>
            </gml:TimePeriod>
          </aixm:featureLifetime>
          <aixm:flightRule>IFR</aixm:flightRule>
          <!--====================-->
          <!-- === REVERSE ASSOC. ===-->
          <!--====================-->
          <aixm:extension>
            <mxia:RouteExtension gml:id="REV_ASSOC">
              <!-=== Links to associated RouteSegments ===-->
              <mxia:theAirspace
xlink:href="http://www.aixm.aero/schema/5.1/example#xpointer(//aixm:Air
space[gml:identifier='cc9c7cc4-e000-4741-854d-b7d93973e099'])"/>
              <mxia:theRouteSegment
xlink:href="http://www.aixm.aero/schema/5.1/example#xpointer(//aixm:Rou
teSegment[gml:identifier='bc430a08-bb5d-48dd-8ef0-85ff50dcfb9d'])"/>
              <!-- ... omitted for brevity ... -->
            </mxia:RouteExtension>
          </aixm:extension>
          <!--====================-->
        </aixm:RouteTimeSlice>
      </aixm:timeSlice>
    </aixm:Route>
  </dnotam:hasMember>
</dnotam:Event>
```

Figure 22 - DNOTAM Event with route and reverse associations

The listing (Figure 22) shows a dnotam Event that provides temporary information about a Route feature. The extension is provided in a RouteTimeSlice.

8.1.1.3.1 Filtering using reverse associations

The way reverse associations help event filtering is quite simple: a filter statement can leverage the additional information provided through the features that are made available.

Consider the example Event in Figure 22. There, a RouteTimeSlice points to an Airspace the Route is protected by. Clients that are interested in all events that concern features protected by that Airspace can create an according filter statement. When searching for events that have an association (not necessarily directly in the Event itself but its child elements) to an Airspace with given identifier, they will get the Route update information provided through the Event in Figure 22.

Spatial filtering of events can also be facilitated. If a given aeronautical feature does not have a spatial extent itself, then the according geometry needs to be identified by other means. A Route, for example, does not directly provide its geometry. The geometry can be computed from the RouteSegments that are part of the Route. A different and probably less exact geometry would be provided by the Airspaces that protect the Route. As said before the Route feature model currently does not provide associations to point to the Airspace(s) or its RouteSegments. These can now be added via reverse associations. Event filter statements can include a function to compute the geometry – either via the RouteSegment(s) or the Airspace(s). This function would need to be defined. The geometry resulting from the function can then be used in spatial filter operations performed upon events.

A problem with this approach is that the reverse associations may not be available in dnotam Events. An Event Service should thus indicate support of these reverse associations, which is discussed shortly in the following section. Some modeling aspects also play a role. This is further discussed in section 8.1.1.3.3.

8.1.1.3.2 Policy considerations

The reverse association mechanism is an extension. As such, it is optional. For clients to be able to leverage the functionality provided by this extension, they need to know if it is supported by a service.

This can be achieved in different ways. First of all, a service can state in its capabilities or via policies that each of the information items it creates contains all possible reverse associations. This, however, can result in significant overhead, especially if many reverse associations are added to the AIXM model.

A service may also allow clients to request the addition of certain reverse associations. This can happen by including according request policies. The service would either already contain the necessary associations in its database or compute them on-the-fly (which can be a significant performance issue). A response to the client may or may not contain the reverse associations – this can also be controlled via policies.

Apparently, the realization and utilization of the reverse association extension can become complex. This is even more the case if a service does not have the means to compute the reverse associations for a given feature itself. This is currently the case for an Event Service. Here, the service only operates on the information given in incoming events. In order to execute filter statements from subscriptions that use reverse associations, the events send to or generated by the Event Service have to contain them. An Event Service thus needs to be able to control the behavior of the event sources that publish data to the service.

Two ways have been identified to solve this issue:

1. The Event Service requires all aviation events sent to it to contain all known reverse associations.

2. The service defines policies that tell the publishers (event sources) which reverse associations are required. This is a more fine grained approach.

Both approaches rely upon the publishers being able to provide the required information. Using reverse associations this way can get complex, especially if not only direct associations are considered but also second-, third- and n-th level reverse associations.

To summarize, policy and capability statements can be used to define behavior that is offered and requested by and from a service with respect to the realization of the reverse association extension. Further tests are needed to better understand the complexity in applying this extension in practical use cases.

8.1.1.3.3 Modeling aspects

This section reviews the way that the reverse association extension was modeled and realized in OWS-7.

The schema implementation of the extension is exemplarily shown in Figure 21. In general, the approach currently taken is to create an AIXM feature type specific extension which explicitly lists the available reverse associations. In Figure 21 this is the *RouteExtension* element which has a group containing two elements, *theAirspace* and *theRouteSegment*.

This approach is inflexible and causes overhead. On the one hand, one extension element needs to be defined per AIXM feature type to contain the reverse associations allowed for that type. On the other hand, listing of the allowed reverse associations in such an extension only allows for these extensions to be added in the content model of a given AIXM feature. New features added in a new version of the core AIXM model or other extensions thereof cannot be incorporated in this way. Even reverse associations to further features of the existing model, if such associations were not incorporated for any reason, cannot be added once the list was finalized.

A better approach would be to create one abstract reverse association extension type that can be used in all AIXM features. This could then be used in any feature type that is added to the AIXM model via future extensions or in new versions of the core model.

One derivation of the abstract reverse association extension type per AIXM feature type would be required – see Figure 23.

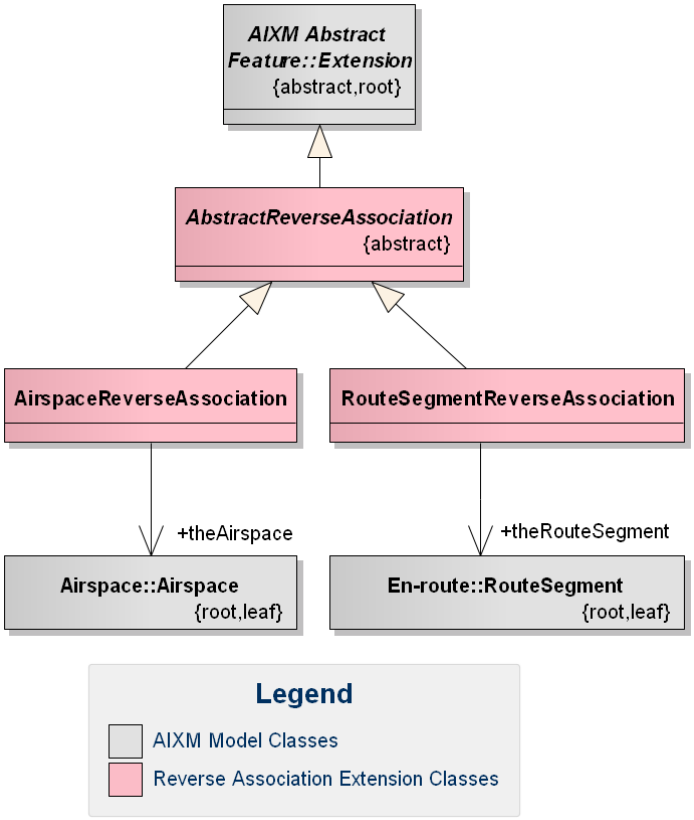

Figure 23 - Modeling reverse associations – common extension type and explicit specializations

Each specific reverse association type can then be used to point back to an instance of the according feature type. In Figure 24 the *AirspaceReverseAssociation* is used by both a Route and a given AIXMFeature to point back to the Airspace that has direct dependencies on these features (the Route and the AIXMFeature).

This approach requires one specialization of the abstract reverse association type per AIXM feature type for which reverse associations are needed. Any AIXM feature – rather: its time slices – can include a number of these special reverse association types as needed.

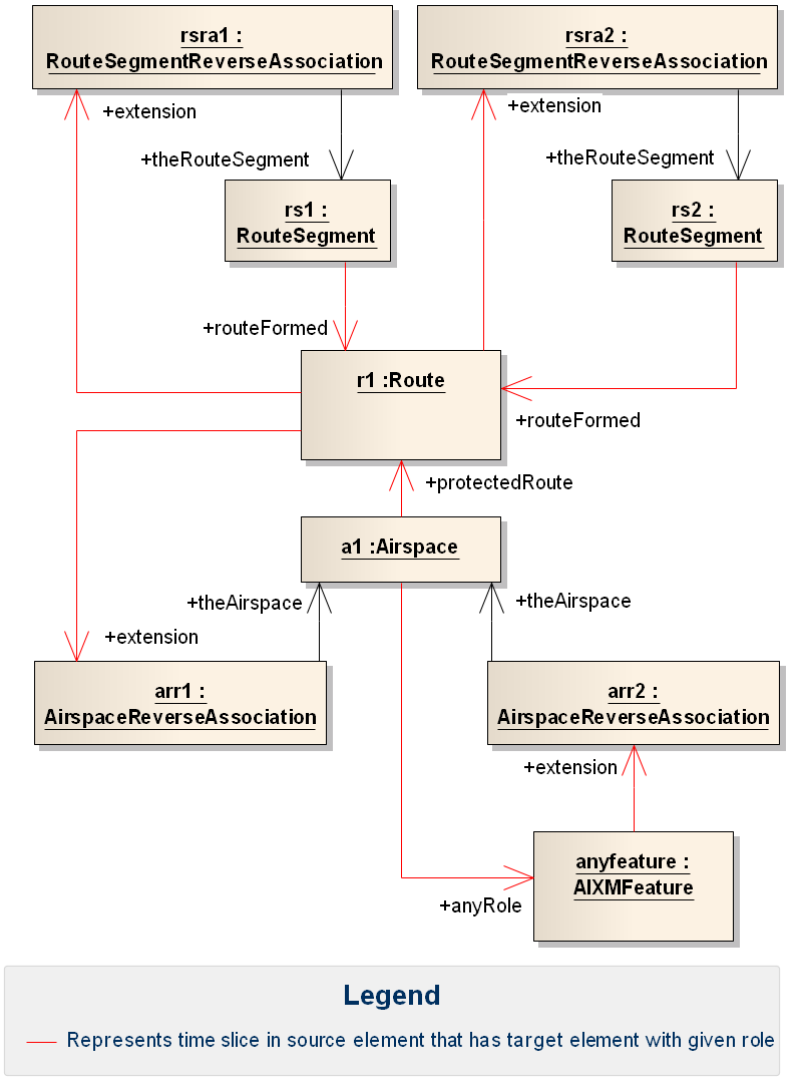

Figure 24 - Using the different reverse association model – UML object diagram

In addition to the modeling approach just described, it is also possible to use a more soft-typed approach (see Figure 25). There, only one reverse association type is needed that points to a given AIXM feature and which also defines the type of that feature.

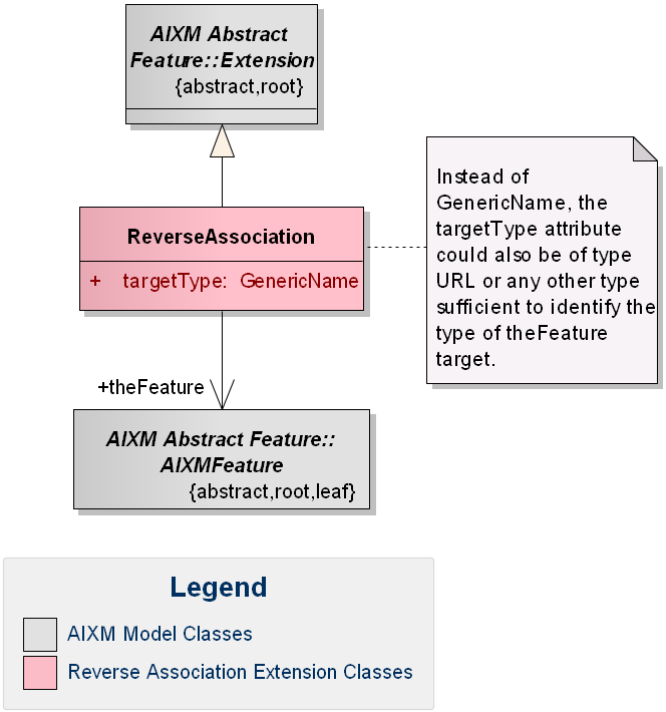

Figure 25 - Modeling reverse associations – soft-typed approach

This would only require well-defined rules for populating the *targetType* property value. A simple approach would be to use the namespace of a given AIXM application schema model and the name of a feature defined by that model. For the Route example, this would be *http://www.aixm.aero/schema/5.1* and Route. In this case the two values can also be concatenated, resulting in the URL *http://www.aixm.aero/schema/5.1/Route* to be used as identifier of *theFeature*'s target type.

Note: The gml:remoteSchema attribute that is part of a GML property given by reference (i.e. using xlink:href and xpointer) might seem to be appropriate for the purpose of recognizing the type of the referenced feature. However, according to the GML 3.2.1 schema the gml:remoteSchema attribute is deprecated. In addition, the gml:remoteSchema element's purpose seems to be (according to GML 3.1.1 – no information was found in GML 3.2.1) to point to the schema that constrains the description of the remote resource referenced by the xlink. Thus, clients would still need to resolve the schema and then investigate it to determine which type of information an xlink is possibly pointing to. In addition, if the xlink only points to the whole schema then – for example in case of the AIXM feature schema – still many feature types may be defined in it, forcing the client to grab (at least the beginning of) the referenced property and investigate its root element.

8.1.1.4 Handling of event properties given inline and byReference

In the XML Schema implementation of the hasMember property of a dnotam Event, an AIXM feature always needs to be provided inline but can also be given byReference using xlinks – see Figure 26.

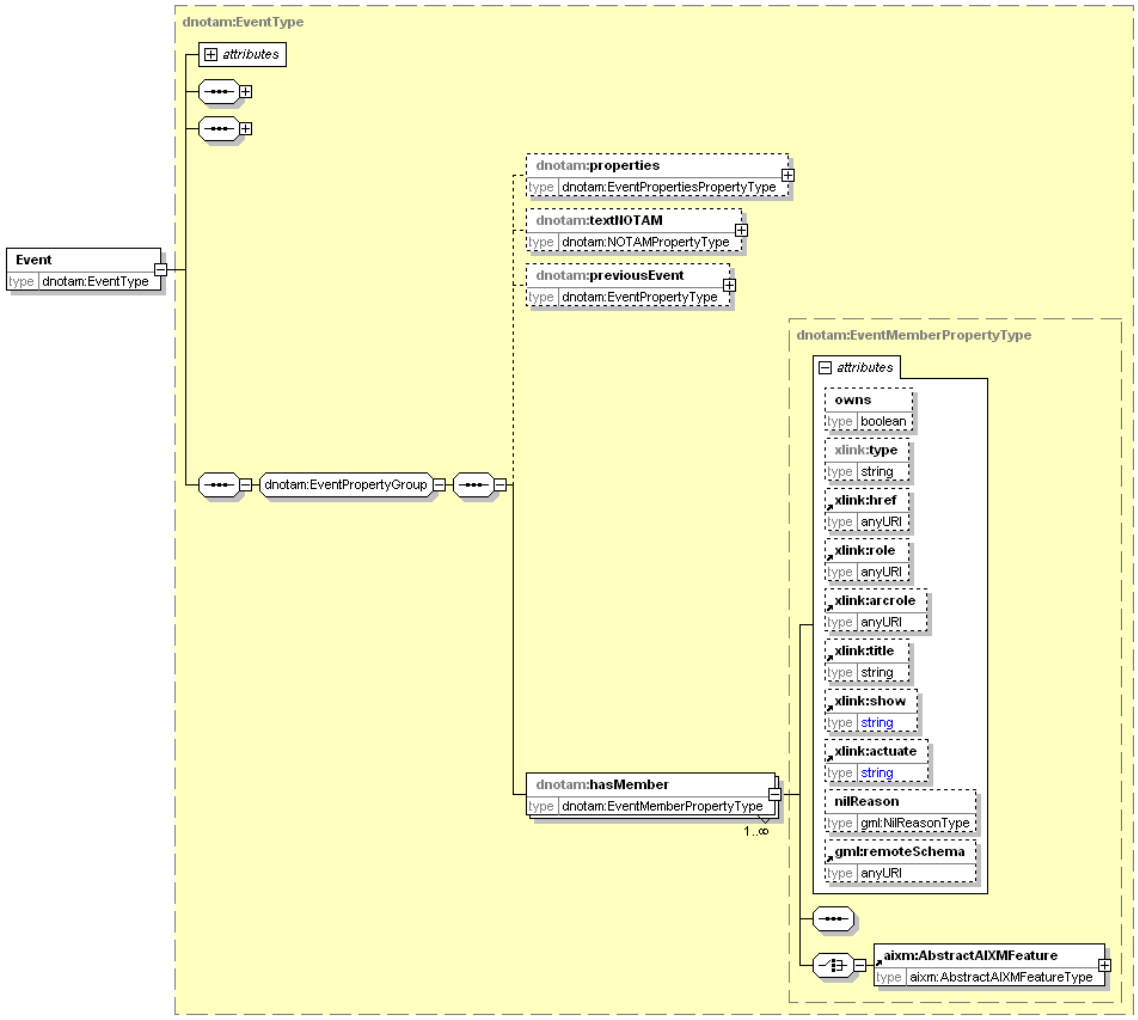

Figure 26 - DNOTAM Event model – hasMember property given inline and byReference (optional)

This encoding deviates slightly from the GML 3.2 encoding rules (see section E.2.4.11 in OGC 07-036). However, one could also say that this is a valid restriction / profile of the encoding rules.

Important for an Event Service – or any other client consuming AIXM encoded data – is that the rules defined in GML 3.2.1 clause 7.2.3.4 are not changed:

"If both a link and content are present in an instance of a property element, then the object found by traversing the xlink:href link shall be the normative value of the property. The object included as content shall be used by the data recipient only if the remote instance cannot be resolved; this may be considered to be a "cached" version of the object." (OGC 07-036)

8.1.2 WXXM

In OWS-7, also weather events were tested in the AIM Event Architecture. Like aeronautical events encoded in AIXM, WXXM encoded weather events are distinct representations of possibly continuous phenomena. We can also say that these

phenomena have been discretized. Weather events can therefore be encoded and sent to an Event Service where they can be filtered according to subscriptions' filter criteria.

Weather events were published to the Event Service in pure WXXM encoding, that is without an additional container element like the dnotam Event used for aeronautical events. Some issues were identified with ambiguous elements encountered in the WXXM 1.1 encoding, for example the duplication of the featureOfInterest property in the Forecast feature type. This resulted in confusion both from a service provider and client point of view, which did not know where to exactly put and filter weather information. These issues were solved during the testbed and the necessary changes incorporated in WXXM version 1.1.1 – for more information see OWS-7 Aviation – WXXM Assessment ER (OGC 10-132).

8.2 Aviation Event Channels

So far, subscriptions tested in the OWS-6 AIM and OWS-7 Aviation threads have been made against the whole set of events available at an Event Service. With an increasing amount of different events published at a service and increasing complexity of the filter statement needed to get only those events that are really of interest to a consumer, advanced publication concepts should be tested and put in place. The work performed in the OWS-7 eventing cross thread started to address this problem with the definition and conceptual modeling of so called event channels – see section 6.3 in OGC 10-060. This work exemplarily created event metadata, also on event channels, for the aviation domain. This was primarily used for the development and testing of initial event service discovery work. For more information, see the OWS-7 Event Architecture Engineering Report (OGC 10-060). This work should be pursued in future work for the aviation domain, as it promises considerable performance benefits for the Aviation Event Architecture.

8.3 AIM Events and Subscriptions

This chapter provides an overview of the subscriptions and notifications that were used in the two scenarios.

Table 4 shows at which steps in the demonstration scenarios a system entity subscribes for certain information. The subscriptions target is given by event type (SIGMET, METAR, NOTAM etc.) as well as spatial filter (e.g. given by airport code).

Table 4 - Subscriptions in the Aviation Scenarios

Scenario	Step	Time	Subscriber	Description (Subscription for …)	SubRef
1	3	08:30	dispatcher	NAT MTT messages	1.1
1	11	10:15	dispatcher	Aeronautical updates for EETN and EFHK with effect after 18:00 UTC	1.2
1	13	10:25	dispatcher	METARs for EETN or EFHK	1.3

Scenario	Step	Time	Subscriber	Description (Subscription for …)	SubRef
1	14	10:25	dispatcher	SIGMETs for an area including EETN and EFHK that have effect after 18:00 UTC	1.4
1	20	10:40	dispatcher	NOTAMs on Kbbb or KBOS having affect during 15:00 UTC until ETA	1.5
1	22	10:50	dispatcher	METARs for Kbbb or KBOS	1.6
1	24	11:55	dispatcher	SIGMETs within 200nm of the flight path during the flight period (15:00 to ETA).	1.7
1	29	13:55	dispatcher	METARs for KJFK	1.8
1	30	14:00	pilot	pilot subscribes to same services as dispatcher (see SubRef 1,2,3,4,5,6,7,8)	p1
1	37	14:52	dispatcher, pilot	PIREPs for EETN, EFHK and EFTU	1.9
1	42	15:00	pilot	METARs or PIREPs on EETN or EFHK, one for each airport	1.10
1	43	15:05	Pilot	METARS on Kbbb, CYUL and CYQX, one for each airport	1.11
1	44	15:05	Pilot	NOTAMs on navaids and runways on Kbbb, CYUL and CYQX, one for each airport	1.12
2	6	08.38	Dispatcher	SIGMETs affecting area inside of the bounding box defined by the points (55°N, 110°W) and (90°N, 180°W).	
2	11	08.45	Dispatcher	METARs and PIREPs for PAFA, PANC and CYYC	
2	12	08.55	Dispatcher	AIREPs for area between 114W to 150W and 50°N to 90° N	
2	15	09.15	Dispatcher	NOTAMs within 100nm of flight path with effect at any time until 20:00 UTC	
2	20	09.35	Dispatcher	NOTAMs for Devil's Lake SUA during the next 12 hours	
2	35	12.40	Pilot	METARs for PAFA, PANC and CYYC	
2	36	12.50	Pilot	NOTAMs for PAFA, PANC and CYYC	
2	52	16.00	Pilot	NOTAMs for KDEN and KSFO	
2	53	16.05	Pilot	METARs for KDEN and KSFO	

At the end of the scenario two, that is the end of the flight, all subscriptions are cancelled by the pilot. For scenario one, the pilot and dispatcher terminate individual subscriptions. This is shown in Table 5.

A much more detailed subscription management strategy is possible. For example, different owners of subscriptions may be in place, and rules may exist when a subscription may be terminated. Also, a subscription may be terminated automatically at a specific termination time, which would avoid idle subscriptions. This can be tested in the future.

Table 5 - Unsubscriptions in the Aviation Scenarios

Scenario	Step	Time	Un-Subscriber	Description (no more …)	SubRef
1	51	17:50	Pilot	METARs for Kbbb	1.11
1	52	17:55	pilot	NOTAMs for Kbbb	1.12
1	54	18:55	Pilot	METARs for CYUL	1.11
1	55	19:00	pilot	NOTAMs for CYUL	1.12
1	59	21:30	pilot	METARs for CYQX	1.11
1	60	21:32	pilot	NOTAMs for CYQX	1.12
1	74	23:50	pilot	Pilot cancels all open subscriptions	p1, 1.10
1	75	23:55	dispatcher	all open subscriptions	1.1 - 9
2	54	17:00	Pilot	All subscriptions	

Table 6 shows which events are published and which subscriptions would target them. Note the different encoding of the messages. According to their content WXXM or AIXM types are used to convey the information.

Table 6 - Notifications in the Aviation Scenarios

Scenario	Step	Time	Description	Subscription	Encoding
1	26	13:30	MTT message	1.1	AIXM
1	27	13:45	METAR: Bad weather in KBOS	1.6	WXXM
1	36	14:50	Non-convective SIGMET, 3000 square miles over Baltic sea	1.4, 1.7, p1	WXXM
1	39	14.56	PIREP on Tallin	9	WXXM
1	53	18:05	METARS for CYUL, CYQX, EETN and EFHK	1.11, 1.10	WXXM
1	57	21:00	METAR for EETN	1.10	WXXM
1	58	21:05	PIREP for EETN	1.9, 1.10	WXXM
1	58	21:20	PIREP for EETN	1.9, 1.10	WXXM

Scenario	Step	Time	Description	Subscription	Encoding
2	13	09.03	METARs for PAFA, PANC and CYYC	Step 11	WXXM
2	16	09.20	AIREP	Step 12	WXXM
2	21	09.40	AIREP	Step 12	WXXM
2	23	10.03	METARS for PAFA, PANC and CYYC	Step 11	WXXM
2	25	10.08	AIREP	Step 12	WXXM
2	27	11.03	METARS for PAFA, PANC and CYYC	Step 11	WXXM
2	28	12.03	METARS for PAFA, PANC and CYYC	Step 11	WXXM
2	31	12.20	NOTAM for Devil's Lake SUA	Step 20	AIXM
2	38	13.03	METARS for PAFA, PANC and CYYC	Step 11, 35	WXXM
2	39	14.00	Volcanic ash SIGMET for Alaska IFR	Step 6	WXXM
2	40	14.10	METARS for PAFA, PANC and CYYC	Step 11, 35	WXXM
2	41	15.20	NOTAMs closing PAFA and PANC due to VA	Step 15, 36	AIXM

8.4 Spatial Filtering of Aeronautical and Weather Events

8.4.1 Introduction

There is a general issue with filtering events based on geometry information in the simple case and based on geometry information in combination with other properties in the complex case. This results mainly from the fact that many aeronautical features have specific geometries or even no geometry at all. An airspace is a collection of geometry components, an airport has a reference point but also a "boundary" polygon, etc. Having this plethora of options, it appears very difficult at that stage to define a unified handling of the locations of features.

Currently, spatial querying is done using the Q line of text NOTAM messages. To achieve a major improvement, digital NOTAMs need to support integrated filtering, i.e. filtering on spatial properties in combination with other thematic properties, such as type of aircraft or specific flight characteristics. To allow at least some basic spatial filtering, the workaround used in OWS-6 was applied in OWS-7 as well. Here, bounding boxes had been created by the service provider explicitly and included in the NOTAM. The bounding box was then used for spatial filtering. In the future, filtering on any combination of properties, including thematic and spatial aspects should be supported and tested.

In the following sections we describe the issues that we face when performing spatial filtering of aeronautical events with the current AIXM model.

8.4.2 Filtering using Bounding Boxes

During the testbed, boundary information of aeronautical features was used to perform spatial filtering of events. Issues and possible solutions using this approach are discussed in the OWS-7 Event Architecture Engineering Report (OGC 10-060) section 9.1.3 (because of the relevance to multiple domains).

> Note: the dnotam:Event is an example for a container event mentioned in OGC 10-060 section 9.1.3 – usually services populate the boundedBy properties of the members of a dnotam:Event and clients subscribe for them.

The main issues when filtering upon spatial properties of aeronautical features via the bounding box computed from these properties are:

- inaccuracy of the bounding box geometry may result in irrelevant data not being filtered out
- no OGC wide mechanism exists to govern the population of bounding box information in features

Approaches to solve these issues are discussed in OGC 10-060.

8.4.3 Filtering using Generic Event Properties

The OGC Filter Encoding Specification (FES) enables clients to create highly detailed filter statements. Any feature property can be identified via XPath like expressions and its value subsequently be used in filter operations (comparisons, spatial filters etc).

The approach described in the previous section – using bounding boxes for spatial filtering of features – uses a small part of the overall functionality that FES provides. If more specific filtering needs to be performed in order to further narrow down the amount of events transmitted to consumers, the full functionality of FES should be made available to and used by clients that subscribe for events.

Clients can create highly detailed filters to for example perform more accurate spatial filtering of events (using those event properties that provide the event's spatial extent in high detail).

This may also include some domain specific functions to compute geometries on the fly from simple feature properties. For example, a Route is formed by several *RouteSegments*, each having a defined start and end point. The points can be expressed in various ways which makes it fairly difficult to create a simple filter expression that incorporates the Route's geometry encoded as a line (because of the many options for encoding the position of a *RouteSegment*'s start and end point). A function could be defined in the aeronautical domain that automatically computes such a geometry for a given Route. Note that the current model requires reverse associations to point from a

Route to its *RouteSegments* – see section 8.1.1.3. Services that provided this function would help clients in creating filter statements a lot. We therefore recommend investigating the definition of domain specific functions in future testbeds to simplify the creation of filter statements for clients.

That it is the clients' responsibility to create complex filter statements is good from a service's point of view because the service does not need to incorporate special logic to support filtering (like being able to compute the boundedBy property of features on-the-fly to support spatial filtering upon this property). Then again, the more complex the filter statements get the more computing resources are needed to execute them.

Additional filter complexity may be caused for example through the need to compute a compound geometry (like the line string of a Route) from multiple feature properties (like a Route's RouteSegments – rather: the start and endpoint of these segments). Clients also need to know exactly which properties are relevant for filtering. Although it is a client's responsibility to know what it is doing, too many options and a rather complex model can be too difficult for new clients to understand in a reasonable amount of time. To support such clients to perform simple tasks, common queries that are used in the aeronautical domain could be hardwired aeronautical services. WFSs support stored queries. This concept can enable the required functionality. The concept should also be considered for other services, like Event Services. This functionality therefore is an interesting candidate to be developed further in the future.

8.4.4 Filtering of AIXM Events in a Standalone Service

The AIXM model poses an interesting problem for filtering events in a standalone service. To understand the problem, we first need to understand the relevant basics of the AIXM model.

The AIXM Temporality Model encodes AIXM feature properties via time slices. The union of all time slices of a given aeronautical feature represents the state of that feature at a given time. Baseline information as well as temporary and permanent changes can be represented. A client can determine the value of a specific feature property by evaluating the baseline value and taking into account any subsequent change to it.

The temporality model is appropriate to account for the dynamic of an aeronautical feature. However, the way that updates to a given feature are currently published is problematic for an Event Service. The problem is that a dnotam Event only provides the updated feature information. It usually does not contain information about the other feature properties.

Why is this a specific problem for the Event Service? Because at the moment it does not have the means to determine the values of feature properties that are not reported in the events that it receives. Thus, subscription filters that query upon such properties cannot be executed. For example a subscription may be interested in all events for the airspaces in a given region. In other words, the subscription defines a spatial filter to identify these airspaces. The filter can for example look for all airspaces that the region *contains*. As discussed before, the AIXM encoding of an Airspace feature uses time slices to encode

the changes of feature properties. The geometry of an Airspace is defined via one or more geometry components. Whenever an update to one of the Airspace's properties is performed, a dnotam Event is generated which contains that update. A client – like the EFB of a pilot – that receives such an update is supposed to have all relevant information about the updated feature. In other words, the client at least has the latest snapshot of that feature. An EFB can for example have gotten this information from the flight dispatcher before liftoff. The feature is identified via the identifier property which therefore needs to be populated in a dnotam Event. An Event Service, on the other hand, is supposed to filter incoming events based upon the information contained in that event. If a dnotam Event for example notifies about a change of an Airspace's activation, it usually does not contain any other property values of the Airspace – also not on the geometry of the Airspace. How then shall the Event Service apply the spatial filter upon this new event? As discussed in sections 8.4.1 and 8.4.2, so far the boundedBy property of aeronautical features that are members of an event were populated and used for spatial filtering. If clients shall be enabled to filter upon any property of a feature that is contained in an event, the Event Service has to be able to get the value of that property.

A solution for this problem needs to be found if detailed filtering of aeronautical events shall be enabled in an Event Service. An AIXM feature could contain or better point to the relevant snapshot of the feature that applies to the time when the updated information is valid. Other solutions may also exist. For example, a dnotam Event or its members (the AIXM features themselves) may contain endpoint information of the WFS that governs the feature and thus has the missing information. These approaches indicate the need for special treatment of AIXM encoded data by an Event Service. A profiling or extension of the general Event Service functionality may be needed to better support filtering of aeronautical events.

8.4.5 Filtering of Events Based on Aircraft Position – Dynamic Spatial Filters

An idea that came up during the testbed was to use dynamic instead of static spatial filters. The spatial filters that were tested so far in subscriptions to aeronautical events used a static buffer geometry computed around the flight path of an aircraft. All events that were within this buffer (other relationships are also possible) matched the subscription and were sent to the subscription's consumer. This also means that all events that are received for regions that the aircraft has long passed match the subscription. To avoid delivery of these events, Event Processing functionality can be applied. There, it would be possible to recalculate the buffer around the flight path from time to time based upon the updated aircraft position. The subscription would include the according processing instructions so that an event service can automatically adjust the buffer around the flight path as the aircraft progresses. Events that are not located within the updated buffer will no longer be delivered - Figure 27 illustrates the concept.

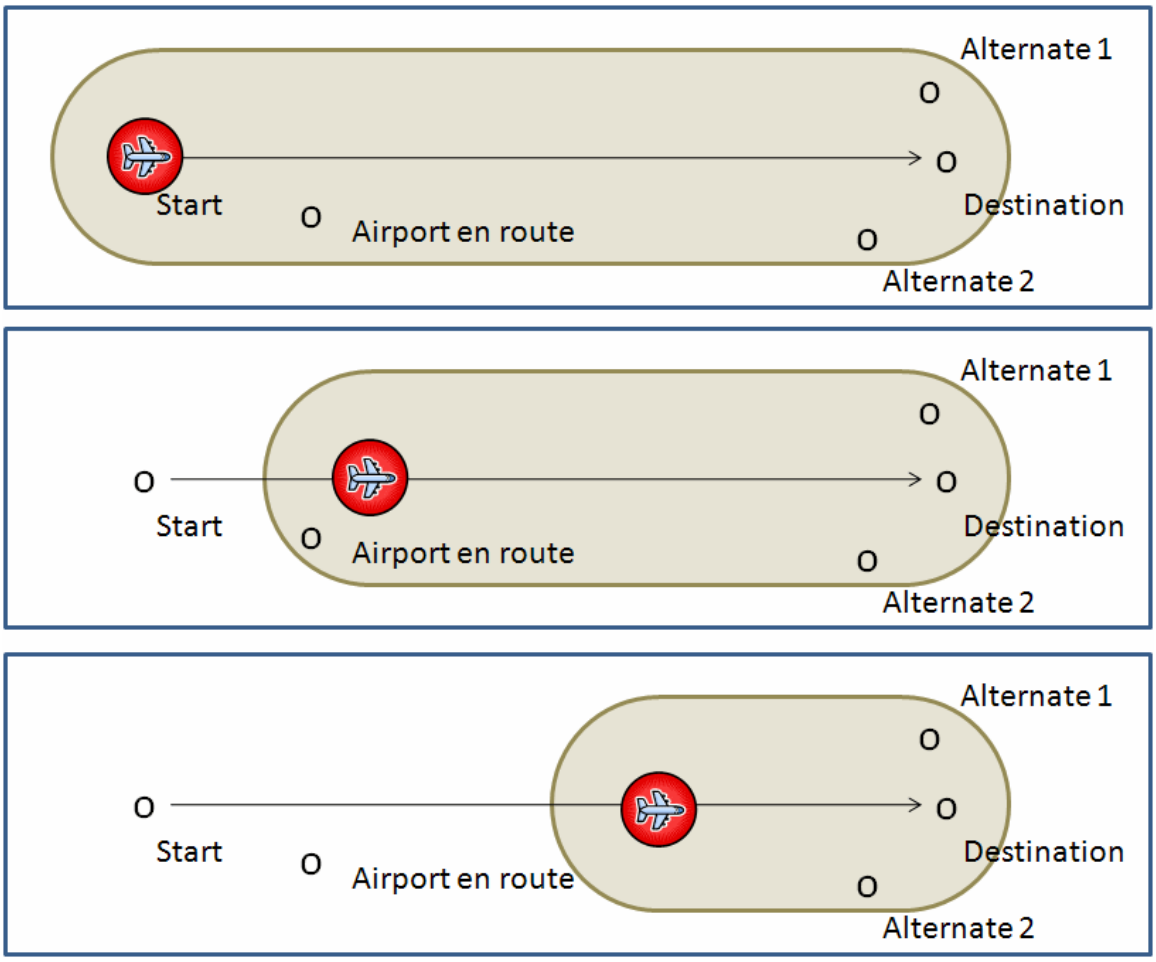

Figure 27 - Buffer around flight path, used for spatial filtering - dynamic updates to avoid past events

Spatial filtering of events using dynamic filter properties is described in more detail in section 9.3.2 of the OWS-7 Event Architecture Engineering Report (OGC 10-060).

8.4.6 Summary

In this chapter various aspects of performing (spatial) filtering on aviation events were discussed.

Various issues exist that make filtering of these events problematic. Using the boundaries of AIXM features for spatial filtering can be inaccurate (in case that the boundary is encoded as a bounding box) while explicitly filtering upon spatial properties of AIXM feature instances can become complex and may not even be possible due to the way AIXM features are encoded in events.

Several options have been discussed to improve the filtering functionality. A common approach to provide the boundary of a given feature could be developed. Also, domain specific functions and stored queries may simplify the creation of filter statements for aeronautical events. Finally, a way to get access to the complete information of an aeronautical feature from a dnotam Event itself may be feasible.

Finally, an interesting mechanism to further improve the filtering of aeronautical events using Event Processing functionality was discussed shortly. It could improve the filtering of aeronautical events so that events that are located in regions that an aircraft has already passed are no longer reported to it.

Future work should investigate how to simplify the process of filtering aviation events while at the same time supporting the functionality to fulfill required use cases. Furthermore, mechanisms to improve the filtering functionality by reducing the amount of uninteresting events that are matched in subscriptions should be investigated.

9 Description of OGC Web Services 7.0

In July 2010, work in the OWS-7 test bed was completed. The test bed had three main interoperability threads.

Sensor Fusion Enablement (SFE)

The SFE Thread builds on the OGC Sensor Web Enablement (SWE) framework of standards that has achieved a degree of maturity through previous OWS interoperability initiatives and deployments worldwide. SFE will focus on integrating the SWE interfaces and encodings with workflow and web processing services to perform sensor fusion. SFE will continue the development of Secure SWE architecture, as well as the interoperability of SWE and Common CBRN Sensor Interface (CCSI).

Emphasis for SFE during this phase of the OWS test bed will be the following:

- Motion Video Fusion. Geo-location of motion video for display and processing. Change detection of motion video using Web Processing Service with rules.
- Dynamic Sensor Tracking and Notification. Track sensors and notify users based on a geographic Area of Interest (AOI). The sensor and the user may be moving in space and time.
- CCSI-SWE Best Practice. Building on OWS-6, develop an ER to be considered by the OGC Technical Committee as a Best Practice.

Feature / Decision Fusion (FDF)

The FDF Thread builds on OWS-6 GPW and DSS work, in particular to advance the state of information cataloguing, Web Processing Services (WPS) profiles, and the Integrated Client. The following task areas have been identified for Feature and Decision Fusion:

- Schema Automation. Transformation from UML to profiles of GML and KML, including generation of constraints.
- Data Discovery and Organization. Use of thematic categories in multi-media data discovery, including augmented metadata for quality of source, fitness for use, and uncertainty of the data values, propagated through usage and workflow.

- Feature and Statistical Analysis. Workflow profiles for feature fusion, including statistical analysis, vector and topological processing. WFS, WMS and WMTS will be used to support statistical mapping.
- Geosynchronization. Web services and client components to support synchronization of geospatial data and updates across a hierarchical Spatial Data Infrastructure (SDI).
- Data and Analysis Sharing. The use of OWS Context for collecting links to web services and analysis results.
- Integrated Client. A field-ready client application to support and display sensor information, cataloguing metadata, notification alerts, feature and statistical analysis, and OWS Context. This client will include an embedded WFS server for geosynchronization support.

Aviation Thread

This thread seeks to further develop and demonstrate the use of the Aeronautical Information Exchange Model (AIXM) and the Weather Information Exchange Model (WXXM) in an OGC Web Services environment.

The US Federal Aviation Administration (FAA) and EUROCONTROL have developed AIXM as a global standard for the representation and exchange of aeronautical information. AIXM uses the OGC Geography Markup Language (GML) tailored to the specific requirements for the representation of aeronautical objects, including the temporality feature that allows for time dependent changes affecting AIXM features. FAA and EUROCONTROL are using AIXM as an integral part of their efforts to modernize their aeronautical information procedures and to transition to a net-centric, global aeronautical management capability. More specifically, AIXM is being used in the net-centric System Wide Information Management (SWIM)-related components of the US NextGen and European Union (EU)'s SESAR programs. Indeed, it is expected that the results of the Aviation Thread of OWS-7 will be contributed to both programs with a focus on recommended OGC specifications that can be applied in the definition and implementation of both SWIM environments.

WXXM is also jointly developed by FAA and EUROCONTROL, as a proposed standard for the exchange of aeronautical weather information in the context of a

net-centric and global interoperable Air Transport System (ATS). WXXM also uses GML tailored to the specific requirements of aeronautical meteorology and is based on the OGC Observation and Measurement Model. WXXM development is harmonized and coordinated with the World Meteorological Organization (WMO), the organization traditionally responsible for standards in meteorology.

The OWS-7 Aviation Thread will investigate and demonstrate the applicability of AIXM and WXXM along with relevant OGC specifications and web services to applications and tools that support Airline Operations Centers and Flight Dispatch applications. Such applications provide information for representing a Common Operating Picture; supporting flight planning (including General Aviation) and preparation (MET and AIM); calculating weight and balance; estimating fuel requirements; in-flight emergency response; etc. The primary focus in OWS-7 is on ground usage of the information, although provision of information packages to the crew, on the ground and in the air, is also of interest.

To support the above goal, the OWS-7 Aviation Thread will cover the following tasks:

- Evaluation and advancement of AIXM, in the areas of using and testing new AIXM 5.1 features, developing components/tools for generating, validating, converting, and parsing AIXM, addressing feature metadata requirements and performance constraints, and supporting the portrayal of AIXM information.
- Evaluation and advancement of WXXM, focusing on the incorporation and demonstration new weather concepts such as the 4-D Weather Data Cube, including the impact of such concepts on the Event Notification Architecture, the support for probabilistic events, and the definition and usage of the time model in WXXM.
- Advancement of the Event Notification Architecture developed and exercised in OWS-6 to support multiple sources of events and multiple types of events and data changes (AIXM, WXXM), and to investigate different delivery protocols (push/pull), registration and subscription lifecycle management approaches, and domain/schema-specific matching between events and subscriptions.
- Integration of AIXM/WXXM in the FAA SWIM environment, focusing on investigating approaches for leveraging SWIM Interface Management, Messaging

and Security capabilities, as well as enabling Aviation clients to access SWIM services in addition to OGC services.