Testbed-12 Asynchronous Messaging for Aviation

Table of Contents

1. Introduction . . . 7
  1.1. Scope . . . 7
  1.2. Document contributor contact points . . . 7
  1.3. Future Work . . . 8
    1.3.1. PubSub-enablement of Data Provision Services . . . 8
    1.3.2. Profile Definition for Geospatial Queries . . . 8
    1.3.3. WFS Queries as Subscription Filters . . . 8
    1.3.4. Integration of FAA SWIM Data Producers . . . 8
    1.3.5. Network of Messaging Brokers . . . 8
    1.3.6. Moving Object Subscription . . . 9
    1.3.7. Integration of Message Broker Software . . . 9
    1.3.8. Definition of an ontology for relevant concepts . . . 9
  1.4. Foreword . . . 9
2. References . . . 10
3. Terms and definitions . . . 11
  3.1. SESAR SWIM-TI and OGC PubSub Taxonomy . . . 11
    3.1.1. General communication characteristics . . . 11
    3.1.2. Message exchange patterns . . . 12
    3.1.3. Communication roles . . . 13
    3.1.4. OGC PubSub 1.0 . . . 14
4. Conventions . . . 15
  4.1. Abbreviated terms . . . 15
5. Overview . . . 17
6. Related Technological Standards . . . 18
  6.1. Advanced Message Queuing Protocol 1.0 . . . 18
  6.2. OASIS Web Services Notification . . . 19
  6.3. Java Message Service . . . 19
7. Current State of Asynchronous Messaging . . . 21
  7.1. Federal Aviation Administration . . . 21
    7.1.1. SWIM Architecture . . . 21
    7.1.2. SWIM in Production . . . 22
  7.2. Previous OGC Work . . . 23
    7.2.1. Event Service . . . 23
    7.2.2. Web Event Processing Service . . . 23
    7.2.3. OGC PubSub 1.0 . . . 24
8. Requirements . . . 27
  8.1. Problem Statements . . . 27
9. Solutions . . . 28

9.1. Targeted Solutions 28
9.1.1. Service Architecture 28
9.1.2. Data Publication 28
9.1.3. Data Dissemination 29
9.2. Recommendations 29
9.2.1. OGC PubSub for Asynchronous Messaging 29
9.2.2. AMQP 1.0 Data Dissemination 29
10. Architecture 30
10.1. General Testbed Architecture 30
10.2. OGC Publish/Subscribe Architecture 31
10.3. Workflows and Use Cases 32
10.3.1. Standalone Publisher 32
10.4. Harris Data Exchange Integration 33
10.5. Subscribe Patterns 34
10.5.1. OGC PubSub 1.0 34
10.5.2. Support for Geospatial Queries 37
10.6. AMQP 1.0 Profile for OGC PubSub 1.0 38
10.6.1. Definitions 39
10.6.2. Exemplary Workflow 39
10.7. FAA-specific AMQP 1.0 Profile for OGC PubSub 1.0 43
10.7.1. Integration Workflow 43
10.7.2. Definitions 44
10.7.3. DEX Taxonomy 46
10.8. Patterns for Publishing Data 46
10.8.1. Data Providers 46
10.8.2. Harris Data Exchange 47
10.9. Delivery Methods 47
10.9.1. AMQP 1.0 48
10.9.2. Harris Data Exchange 49
10.9.3. Web Services Notification 49
11. Implementations 50
11.1. Services 50
11.1.1. 52°North subverse 50
11.1.2. Harris Data Exchange 50
11.2. Clients 52
11.2.1. 52°North aviationFX 52
11.3. Message Broker Software 53
11.4. AMQP 1.0 Software Solutions 53
11.4.1. Apache Qpid 54
11.4.2. Microsoft Azure AMQP.Net Lite 56
12. Conclusions 58

12.1. Accomplishments . . . 58
Appendix A: Asynchronous Messaging Server Capabilities Document . . . 59
Appendix B: AIXM DNOTAM Document . . . 64
Appendix C: FIXM Flight Document . . . 69
Appendix D: Revision History . . . 72
Appendix E: Bibliography . . . 73

Publication Date: 2017-04-25

Approval Date: 2016-09-13

Posted Date: 2016-08-29

Reference number of this document: OGC 16-017

Reference URL for this document: http://www.opengis.net/doc/PER/t12-E003

Category: Public Engineering Report

Editor: Matthes Rieke, Aleksandar Balaban

Title: **Testbed-12 Asynchronous Messaging for Aviation**

OGC Engineering Report

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

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

Abstract

The Asynchronous Messaging for Aviation Engineering Report (ER) focuses on the design of an architecture to create an Publish/Subscribe (PubSub) messaging layer between different Aviation components such as clients, data provider instances and Data Brokers. In order to achieve interoperability among these components, the OGC PubSub 1.0 standard forms the basis of this architecture. The design of this architecture will cover methods for subscribing for specific subsets of data (e.g. Flight Information Exchange Model (FIXM) Flights intersecting a given Airspace), managing such subscriptions as well as publishing data to the Asynchronous Messaging Server. Different delivery methods such as Advanced Message Queuing Protocol (AMQP) 1.0, Java Message Service (JMS) and OASIS WS-Notification are considered. In particular, their harmonization with OGC PubSub 1.0 is evaluated.

This report focuses on the interface design required to define an interoperable approach for Aviation using this OGC PubSub 1.0. Specific service level integrations (i.e., Federal Aviation Administration (FAA) System-Wide Information Management (SWIM) and Single European Sky ATM Research Programme (SESAR) SWIM) have been investigated but an implementation has not been fulfilled.

Business Value

Air Traffic Management (ATM) data as well as Flight data changes frequently. In order to achieve an efficient and reliable publication of these changes, a pull-based approach using classical request/response web services is not sufficient.

The design of an asynchronous messaging architecture which is based on the OGC PubSub 1.0 standard and combines industry-established technologies such as AMQP 1.0 and Web Services Notification overcomes this gap and eases integration. By following a standards-based approach, an interoperable solution is available which, besides the Aviation domain, can be easily transferred to other fields that require near-real time, push-based data dissemination.

What does this ER mean for the Working Group and OGC in general

Eventing patterns and asynchronous messaging previously played an important role in OGC Testbed Aviation threads. Architecture patterns similar to the work done in the PubSub WG have been prototypically implemented based on best practices. The PubSub 1.0 standard has been approved by the OGC and an implementation of relevant parts of it will provide a valuable assessment of the design approach of the standard. Besides the implementation, a definition of a profile for aviation could serve as an example on how to integrate OGC PubSub 1.0 into a specific domain.

How does this ER relates to the work of the Working Group

The architecture documented in this ER is one of the first implementations of the OGC PubSub 1.0 standard. Hence, it is of great importance to assess and validate the specification itself, in particular as regards the following aspects:

- Extensibility of the PubSub 1.0 specification;
- Harmonization of PubSub 1.0 specification, namely the SOAP binding, with AMQP 1.0 as a delivery method;
- Integration and backwards compatibility with Harris Data Exchange (DEX) clients supporting the retrieval of data (e.g. AIXM, FIXM) via JMS;
- As noted, the ER focuses on the Aviation domain, however a definition of a profile for aviation could serve as an example on how to integrate PubSub into other fields that require near-real time, push-based data dissemination;
- Use the *OGC Filter Encoding 2.0* language to express subscription filters; and
- Plugability of filter languages, e.g. to reference the correct geometry in aviation-specific data formats, via XPath/XQuery, etc.

This ER also contributes to some of the activities in the current scope for the PubSub Standards Working Group (SWG), such as:

- Feasibility of adding PubSub capabilities to Web Feature Service (WFS) 2.0 and Web Map Service (WMS) 1.3.0;
- Definition of a generic mechanism to reuse service-specific data requests to PubSub-enable the existing Web Services; and
- Definition of lexical constants (e.g. "all" to identify all the publications offered by a Publisher).

AMQP is a broadly useful and popular delivery method, and should be considered for standardization in the PubSub SWG. The ER authors are encouraged to bring an AMQP delivery method document to the SWG for consideration and standardization.

Finally, the ER provides useful indications on future lines of work for the PubSub specification, such as:

- OGC PubSub 1.0 profile for moving objects;
- Specific mechanisms for standardizing delivery methods; and
- Analysis on the role of Message Brokers in enterprise production environments (cf. the PubSub Brokering Publisher).

Keywords

ogcdocs, testbed-12, PubSub, Eventing, Aviation, AIXM, AMXM, FIXM, Asynchronous Messaging

Chapter 1. Introduction

1.1. Scope

Currently, OGC Web Services [1: the family of OGC standards that share the same service model, OGC Web Services Common: http://www.opengeospatial.org/standards/common] only support synchronous web service request-response query capabilities. This report investigates the means to incorporate asynchronous messaging to OGC Web Services. Asynchronous messaging is a form of message delivery where the provider and consumer can be decoupled (e.g. in time and in terms of knowledge of each other). The recently released OGC Publish/Subscribe specification 1.0 defines a Publish/Subscribe model for OGC Web Services. The work within this testbed applies this specification for the retrieval of aviation data (i.e., AIXM and FIXM) information using geospatial queries and AMQP 1.0 as the underlying messaging protocol. The developed system design, the involved components as well as common workflows are documented.

This report targets the following problem statements.

- How to define an OGC compliant web service that allows the management of subscriptions and the corresponding delivery of messages?
- What protocols suite best for the dissemination of Aviation data messages from data publishers to client components?
- How to define an interoperable solution that meets all requirements in terms of geospatial filtering capabilities, data dissemination and reliability?

In particular, the design and realization of a service architecture based on an implementation of the *OGC Publish/Subscribe 1.0* specification (PubSub) in combination with the AMQP 1.0 protocol is illustrated. OGC PubSub 1.0 is a recently published standard which is agnostic of message delivery protocols. This report therefore focuses on the interface design required to define an interoperable approach for Aviation using this standard. Specific service level integrations (i.e. FAA SWIM and SESAR SWIM) have been investigated but an implementation has not been fulfilled. The Future Work section outlines work packages on corresponding efforts.

An in-depth analysis of the existing approaches for asynchronous messaging in Aviation and related available technologies was not the scope of this testbed. In particular, it focused on the prototyping of an OGC based Publish/Subscribe service architecture in correspondence with related message delivery methods and will not provide recommendations beyond this scope.

1.2. Document contributor contact points

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

Table 1. Contacts

Name	Organization
Matthes Rieke (editor, m.rieke@52north.org)	52°North

Name	Organization
Aleksandar Balaban (editor, aleksandar.balaban@m-click.aero)	m-click.aero

1.3. Future Work

1.3.1. PubSub-enablement of Data Provision Services

The current architecture uses an OGC PubSub 1.0 standalone publisher for subscription management and data dissemination. OGC PubSub 1.0 also defines an architectural approach that adds Publish/Subscribe capabilities to existing OGC services such as the Web Feature Service. Investigating the feasibility of this approach will be very useful as it may result in a simplified service architecture.

1.3.2. Profile Definition for Geospatial Queries

The current approach developed within this testbed is not fully interoperable with the referenced standards due to the manner in which the geometry part of a message/feature is referenced. The relevant data formats such as AIXM, FIXM and Aerodrome Mapping Exchange Model (AMXM) and their features often contain multiple geometries. It will therefore be of great benefit to define an common approach to define subscriptions with geospatial queries for the Aviation domain. A dedicated profile for OGC PubSub 1.0 could be a reasonable approach. Such a profile would define a solution on how to reference the correct geometry (e.g., via XPath/XQuery or alternatively constant values).

1.3.3. WFS Queries as Subscription Filters

The current architecture only supports filters based on the *OGC Filter Encoding 2.0* language. The OGC PubSub SWG has identified the reuse of service-specific data requests as valuable concept for message filtering. Future work could contribute to the SWG by targeting a WFS specific approach. A subscription could be defined using a *WFS GetFeature* query. A client would then request baseline data from a WFS using a specific query and then simply reuse that query to define a subscription at the Asynchronous Messaging Server. This will ease the integration of the Asynchronous Messaging Server into existing WFS-based architectures.

1.3.4. Integration of FAA SWIM Data Producers

Data producers within FAA SWIM currently provide means for accessing data providing certain business logic (e.g. Stored Queries to access Digital Notice To Airmen (NOTAM) within a 50 miles radius of a specific NAVAID feature). In order to re-use this business logic and to minimize the required systems engineering work dedicated work should be carried out on defining corresponding OGC PubSub 1.0 approaches. The Traffic Flow Management System (TFMS) or the Terminal Flight Data Manager (TFDM) could provide valuable use cases for such integrations.

1.3.5. Network of Messaging Brokers

Productive messaging systems often feature a network of distributed message broker nodes. In order to route messages to the correct destinations additional metadata is required. The current approach utilizes AMQP 1.0 node links to specify the destination. In a complex brokering environment this is not sufficient. Future work should take these systems into account and extend the developed OGC PubSub 1.0 profile with definitions on necessary metadata (e.g. the `to` property field of an AMQP 1.0 message).

1.3.6. Moving Object Subscription

In *OGC Web Services (OWS) Initiative - Phase 8* a scenario has been developed where a spatial subscription was defined in a dynamic way (see [OGC 11-093r2]). The position on the flight route of a given aircraft was used to determine if a message was relevant (if it intersects the remainder of the route). A prototypical design has been developed but no interoperable approach was specified. Such a use case has been identified as a valuable contribution within this testbed. Therefore work on defining an OGC PubSub 1.0 subscription profile for moving objects will be of great benefit for the Aviation domain (especially FIXM Flight object updates are a perfect match).

1.3.7. Integration of Message Broker Software

A deeper analysis of topic- and queue-based delivery patterns which involve Message Broker Software and accordingly the definition of OGC PubSub 1.0 delivery profiles should be investigated. Such Message Brokers play an important role in the establishment of enterprise production environments and are therefore a crucial aspect of asynchronous messaging architectures.

1.3.8. Definition of an ontology for relevant concepts

A specific need has been identified to define an ontology for the consistent use of asynchronous messaging terminology that lie within the scope of this work. It will help to position the relevant specifications (OGC PubSub 1.0, AMQP, OASIS WS-N) and relate it to existing ontologies (e.g. SESAR SWIM Technical Infrastructure).

1.4. Foreword

Chapter 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: **OGC 09-025r2** OGC® Web Feature Service 2.0 Interface Standard - With Corrigendum (2014)
- OGC: **OGC 09-026r2** OGC® Filter Encoding 2.0 Encoding Standard - With Corrigendum (2014)
- OGC: **OGC 11-093r2** OWS-8 Aviation Architecture Engineering Report (2011)
- OGC: **OGC 15-118** IMIS Profile Recommendations for OGC Web Services Engineering Report (planned 2017)
- OGC: **OGC 13-131** OGC® Publish/Subscribe Interface Standard 1.0 - Core (2016)
- OGC: **OGC 13-133** OGC® Publish/Subscribe Interface Standard 1.0 - SOAP Protocol Binding Extension (2016)

Chapter 3. Terms and definitions

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

Table 2. Terms and definitions

Publish/Subscribe Model	A concept where senders of messages (= publishers), do not send messages directly to specific receivers (= subscribers), but instead publish messages without knowing possible subscribers. Vice versa, subscribers express interest in (a subset of) messages and only receive messages that are of interest without knowing possible publishers. There exist different definitions of Publish/Subscribe. This report re-uses the definition given in the OGC Publish/Subscribe 1.0 specification (section 6.1 of [OGC 13-131]).
Asynchronous Messaging	Describes a communication pattern in which sending entities can deliver in an asynchronous way. No immediate response from the receiving entity is required to continue processing. Receiving entities can pick up messages directly or at a later point in time. This report uses AMQP 1.0 as the protocol for asynchronous message delivery.

3.1. SESAR SWIM-TI and OGC PubSub Taxonomy

SESAR SWIM-TI (Technical Infrastructure) specification document provides its own, exact specification for messaging/communication taxonomy. It defines a vocabulary based on the available sources and the best practices considering the fact that there is no widely accepted, international standardized communication vocabulary.

The SWIM-TI defines the communication concepts through three different kinds of communication participants decoupling, as well as through further two characteristics: the information persistence and the dissemination (discrete/streaming). Further, it specifies a list of message exchange patterns, which are the combinations of decoupling, statements about communication participants, information flow directions and cardinalities.

3.1.1. General communication characteristics

Decoupling

Decoupling describes the degree of loose coupling between the participants. Decoupling is subdivided into 3 dimensions, as follows.

- **Time**: Time decoupling means that the interacting parties do not have to be actively participating at the same time.
- **Space**: Space decoupling means that the interacting parties do not have to know each other.
- **Synchronization**: Synchronization decoupling means that the interacting parties are not blocked and can do other work.

Persistence/transient

In persistent communication, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver. In contrast, with transient communication, a message is stored by the communication system only as long as the sending and receiving applications are executing.

Discrete/streaming

Discrete/streaming describes if a unit of information exchange does contain all relevant data or the relevant data will be transmitted/received in many atomic communication steps.

3.1.2. Message exchange patterns

The SWIM information distribution is realized via the support to specific Message Exchange Patterns (MEP). MEPs are characterized through 4 groups of attributes:

Conversation direction A conversation is a series of related messages. The Conversation direction describes the sequencing and direction of the flow of messages between the interacting parties.

Cardinality describes the number of participants in the exchange of messages.

Decoupling describes the degree of loose coupling between the participants.

Push/Pull indicates whether a subscriber will receive the data at the initiative of the publisher (Push) or whether the subscriber needs to fetch the data (Pull).

The following table represents SWIM-TI message exchange patterns (MEPs), which are directly supported by OGC PubSub 1.0 standard. Additionally, it explains how OGC PubSub 1.0 requirement classes implement SWIM-TI message exchange patterns.

Table 3. Message Exchange Patterns

MEP	*Direction conversation*	*Cardinality*	*Time Decoupling*		*Synchronization Decoupling*		*Space Decoupling*	
	Consumer=C Provider=P Subscriber=S Publisher=Pu		**C / S**	**P / Pu**	**C / S**	**P / Pu**	**C / S**	**P / Pu**
Observer Push (OPUSH-MEP)	1 way (Pu → S)	1-many	No	No	Yes	Yes	No	No
Observer Pull (OPULL-MEP)	1 way (Pu → S)	1-many	No	No	Yes	Yes	No	No
Publish/Subscribe Push (PSPUSH-MEP)	1 way (Pu → S)	many-many	Yes	Yes	Yes	Yes	Yes	Yes
Publish/Subscribe Pull (PSPULL-MEP)	1 way (Pu → S) Synchronous R/R	many-many	Yes	Yes	Yes	Yes	Yes	Yes

The following MEPs are removed from the original list because they are irrelevant for the communication paradigm explored in this engineering report:

- Synchronous Request/Reply (SRR-MEP)
- Synchronous Request/Reply (SRR-MEP)
- Asynchronous Fire & Forget (AFF-MEP)
- Fully Decoupled Request/Reply (FDRR-MEP)

The OGC PubSub 1.0 core specification is organized into requirement classes. Observer patterns are specified in the "Standalone Publisher" requirement class, while the Publish/Subscribe ones are required in the "Brokering Publisher" requirement class.

From the OGC PubSub 1.0 point of view, the Push/Pull MEPs are dependent on the concrete notification delivery implementation and are not part of the core specification. They are rather specified in so called binding documents. For example, the requirements class SOAP Brokering Publisher from OGC PubSub 1.0 SOAP binding document defines a broker middleware component, which sends notifications (publications) to receivers (subscribers) using push delivery method. This MEP corresponds to PSPUSH-MEP from the table above.

3.1.3. Communication roles

The MEPs identify distinct participants in the communication with distinct roles. The classification of roles in SWIM-TI is as follows.

Common roles:

- Service provider: A generic role for providing an ATM specific service that can be consumed; and
- Service consumer: A generic role for consuming an ATM specific service that is provided.

Roles similar to those used in OGC PubSub 1.0:

- Publisher: A publisher produces information that is potentially of interest for a Publication consumer;
- Subscriber: A Subscriber subscribes interest for receiving information by a Publication consumer, negotiates the modalities for delivery of this information and manages the lifecycle of the subscription; and
- Publication consumer: A Publication consumer receives information for which the Subscriber has subscribed.

Roles irrelevant for this ER:

- Publication mediator: A Publication mediator receives the information produced by a Publisher and forwards that information to all Publication consumers for which the subscriptions match the Publication characteristics;
- Subscription handler: A Subscription handler interacts with the Subscriber and maintains the subscriptions;
- Registration handler: A Registration handler interacts with the Publisher and maintains the registrations;

- Fire & Forget Mediator: A Mediator provides time decoupling between client and server in a Fire & Forget MEP; and
- Request Reply Mediator: A Mediator provides time and synchronization decoupling between consumer and provider in a fully decoupled Request/Reply MEP.

Opposite to the detailed role schema defined in SWIM-TI, the OGC PubSub 1.0 distinguishes just a several roles for entities participating in information exchange interactions such as: Sender, Receiver, Subscriber, and Publisher.

3.1.4. OGC PubSub 1.0

The OGC PubSub 1.0 standard specification describes an asynchronous communication system based on individual requirements collected into sets called requirement classes. They extend top level major concepts from the core document gradually adding more details. The top level requirement class is called Basic Publisher. It describes a communication between publishers and subscribers without intermediary middleware components and therefore with temporal decoupling only. This schema corresponds to Observer patterns. A subclass class called Brokered Publisher introduces additional requirements for spatial decoupling and therefore corresponds to the Publish/Subscribe patterns.

The OGC PubSub 1.0 specification describes a communication system with obligatory temporal (Standalone Publisher) and optional spatial and synchronization decoupling.

If a system architecture implements OGC PubSub 1.0 Brokering Publisher requirement class, then it has to implement a message broker, which implies the spatial decoupling. The intermediary (middleware) broker component is defined as the "Brokered Publisher" in the OGC PubSub 1.0 terminology.

Synchronization decoupling is not mandatory. It is frequently provided by standard communication software libraries and application architecture implementation models.

The OGC PubSub 1.0 compatible communication system doesn’t explicitly require message persistence. However, the persistence might be specified as part of binding documents. For example, if data delivery shall be implemented using pull style, it automatically imply persistence for messages. Similar is true for discrete/streaming like data dissemination.The requirements are not included into the core specification and might be specified as part of binding specifications.

The push and pull notification/publication delivery approaches are dependent on a concrete implementation and are not part of the core specification. However, the binding documents target on these approaches. For example, the requirements class "SOAP Brokering Publisher" from the SOAP binding document requires a broker middleware component, which sends notifications (publications) to receivers (subscribers) using push delivery method. This MEP corresponds to PSPUSH-MEP from the table above.

Chapter 4. Conventions

4.1. Abbreviated terms

- **AIM** Aeronautical Information Management
- **AIXM** Aeronautical Information Exchange Model
- **AMQP** Advanced Message Queuing Protocol
- **API** Application Program Interface
- **ATM** Air Traffic Management
- **CSW** OGC Catalogue Service for the Web
- **DNOTAM** Digital NOTAM
- **EAD** Pan-European AIS Database
- **FAA** Federal Aviation Administration
- **FES** Filter Encoding Specification
- **FIXM** Flight Information Exchange Model
- **GML** Geography Markup Language
- **GUFI** Globally Unique Flight Identifier
- **HTTP** HyperText Transfer Protocol
- **ISO** International Standards Organization
- **NAS** National Airspace System
- **NEMS** NAS Enterprise Messaging Service
- **NOTAM** Notice To Airmen
- **OASIS** Organization for the Advancement of Structured Information Standards
- **OGC** Open Geospatial Consortium
- **OWS** OGC Web Service
- **PubSub** OGC Publish/Subscribe 1.0 Specification
- **QoS** Quality of Service
- **SAA** Special Activity Airspace
- **SESAR** Single European Sky ATM Research Programme
- **SOA** Service-oriented architecture
- **SOAP** Simple Object Access Protocol
- **SWIM** System-Wide Information Management
- **TCP** Transmission Control Protocol
- **WFS** Web Feature Service
- **WSDL** Web Services Description Language

- **WS-N** OASIS Web Services Notification
- **XML** Extensible Markup Language

Chapter 5. Overview

The Engineering Report starts with an overview on the related technological standards (section Related Technological Standards) in which an overview on the *AMQP 1.0* protocol, *Java Message Service* as well as *Web Services Notification* is provided.

Subsequently an analysis on the current state of asynchronous messaging as well as Publish/Subscribe approaches in the Aviation domain (section Current State of Asynchronous Messaging) is given. Existing solutions in both the FAA and SESAR SWIM architecture are described as well as previous OGC work on event-driven message dissemination.

Section Requirements provides an overview on the requirements for the system architecture. Following the targeted (and discarded) solutions are presented (section Solutions). Sections Architecture and Implementations describe the architecture design and the corresponding implementation.

Finally, the report concludes with a summary of accomplishments and lessons learned (section Conclusions).

Chapter 6. Related Technological Standards

The technologies and frameworks described in this section are relevant to the design and implementation of a Publish/Subscribe messaging architecture and a corresponding an asynchronous messaging layer for Aviation. Therefore, a brief overview is outlined to provide the necessary context.

6.1. Advanced Message Queuing Protocol 1.0

AMQP 1.0 [2] is a wire-level protocol that was standardized as by OASIS in 2012. There exists another well-adopted version (0-9-1) that is incompatible with AMQP 1.0 and thus will not be introduced in this document.

As mentioned above, AMQP 1.0 is a wire-level protocol and not an Application Programming Interface (API). Therefore, it is a description of the format of the data that is sent across the network as streams of octets as well as concepts such as reliable delivery of messages. It does not define a set of methods on how to interact with a software entity like an API.

AMQP 1.0 defines a set of message-delivery guarantees (as part of its internal "link protocol"):

- at-most-once (a message is delivered once or never);
- at-least-once (a message is certain to be delivered, but duplicates could occur); and
- exactly-once (a message will always certainly arrive without duplicates).

The structure of an AMQP 1.0 bare message is composed of a set of standard properties (id, user, time of creation, subject, etc.), a non-mandatory set of application-specific properties, and the message body. A bare message is considered as immutable, though it can be annotated with additional information (see Figure 1). In particular, the message header is a well-defined set of delivery-related annotations (time to live (TTL), priority, etc.).

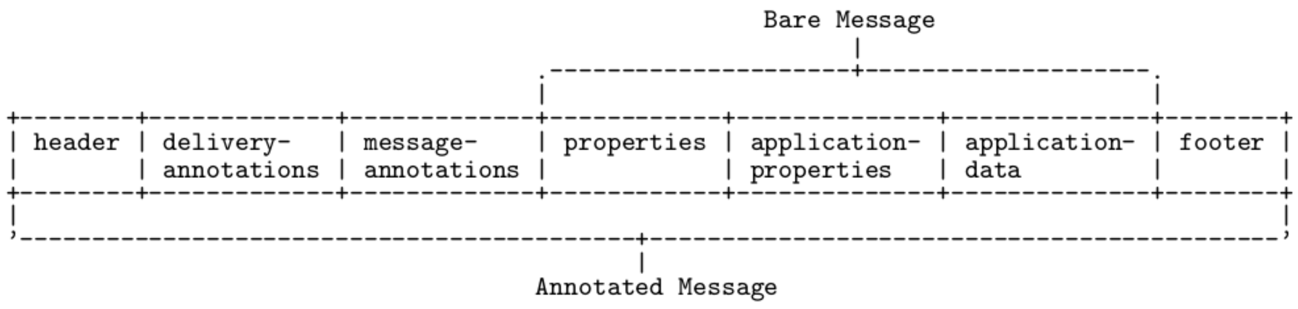

Figure 1. AMQP 1.0 Message Format (source: OASIS AMQP 1.0 Specification)

The AMQP 1.0 specification does not define a message broker architecture, e.g., JMS. Therefore a message broker can define interaction patterns in a custom manner. Still, the specification describes the concept of distributions nodes (e.g. queues and topics) that shall follow certain rules (e.g. expiration of messages after their time-to-live (TTL)).

6.2. OASIS Web Services Notification

The OASIS Web Services Notification (WS-N, [3]) family of standards defines a set of web services and their interaction following the Publish/Subscribe pattern and asynchronous messaging (for a definition of these terms see Terms and definitions). WS-N uses the HTTP protocol and SOAP 1.2 as the message layer protocol.

WS-N defines a set of roles for components of a distributed service architecture and message protocol. In particular, these roles are Publishers, Notification Producers, Notification Consumers and Notification Brokers which are briefly described as follows:

- a Publisher is responsible for formatting a Notification and disseminating it to a NotificationProducer;
- a Notification Consumer must provide an interface for receiving NotificationMessages;
- a Notification Producer must provide an interface for subscribing to a subset of messages and must be capable to deliver these subsets of NotificationMessages to the subscribing Notification Consumer; and
- a Notification Broker combines the tasks of a Notification Producer and a Notification Consumer and acts as an intermediary distributor.

6.3. Java Message Service

The JMS specifies an API for message exchange between two or more clients. JMS is developed under the Java Community Process [2: https://www.jcp.org] and is integrated within the Java Enterprise Edition (Java EE).The JMS API implements the Publish/Subscribe pattern and supports both 1-to-1 (queues) and 1-to-n (topic) delivery. As JMS is an API, it is not ensured that different *implementors* of this API interoperate seamlessly. For example, if a client software wants to interact with a JMS provider, it has to ensure that it includes provider-specific components into the software. This is most often limited to *runtime* components, therefore the business logic of the client software does not need to be adjusted.

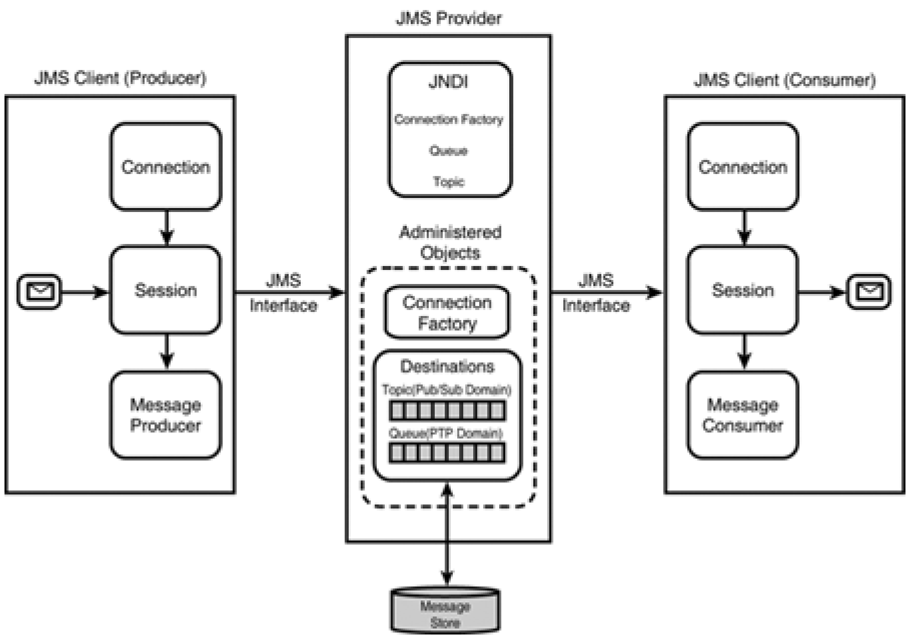

Figure 2. JMS Architecture (source: [1])

JMS defines a set of concepts and roles (see Figure 2) that interact with each other through the API. A *Producer* publishes messages to *Destinations* (queues or topics). A *Consumer* listens on these to receive messages on the other end.

Chapter 7. Current State of Asynchronous Messaging

Asynchronous messaging (see Terms and definitions) has been implemented in several ways in the Aviation domain. These approaches are often specific to a system (e.g. SWIM and SESAR) and are not designed to be interoperable. This section describes previous efforts of the OGC on an interoperable Publish/Subscribe architecture supporting asynchronous messaging.

7.1. Federal Aviation Administration

In 2007, the FAA established the SWIM Program to implement a set of Information Technology principles in the National Airspace System (NAS) and provide users with relevant and commonly understandable information [5]. The following sections provide an overview of the FAA SWIM platform.

7.1.1. SWIM Architecture

SWIM enables the sharing of information between diverse systems enabling the Next Generation Air Transportation System (NextGen) to deliver the right information to the right place at the right time. To achieve this, SWIM provides the IT Service Oriented Architecture (SOA) enterprise infrastructure necessary for NAS systems to share and reuse information and increase interoperability [6]. SWIM's approach allows software applications in the NAS to interact with one another through information services that can be accessed without knowledge of an application's underlying platform implementation. This simplifies interface requirements to existing NAS systems and ensures new systems can be built with minimum technology (hardware, software, and data definition) constraints. SWIM also enables the transition to net-centric NAS operations, and from tactical conflict management to strategic, trajectory-based operations.

At the heart of the SWIM architecture is the NAS Enterprise Messaging System (NEMS). NEMS provides some of the core services under the SWIM program including messaging, service management, service-level security, and mediation services. NEMS provides both asynchronous publish/subscribe and request/reply platforms. For asynchronous publish/subscribe transactions, NEMS uses the ActiveMQ 1.0 protocol as well as JMS. Figure 3 illustrates the JMS flow through NEMS.

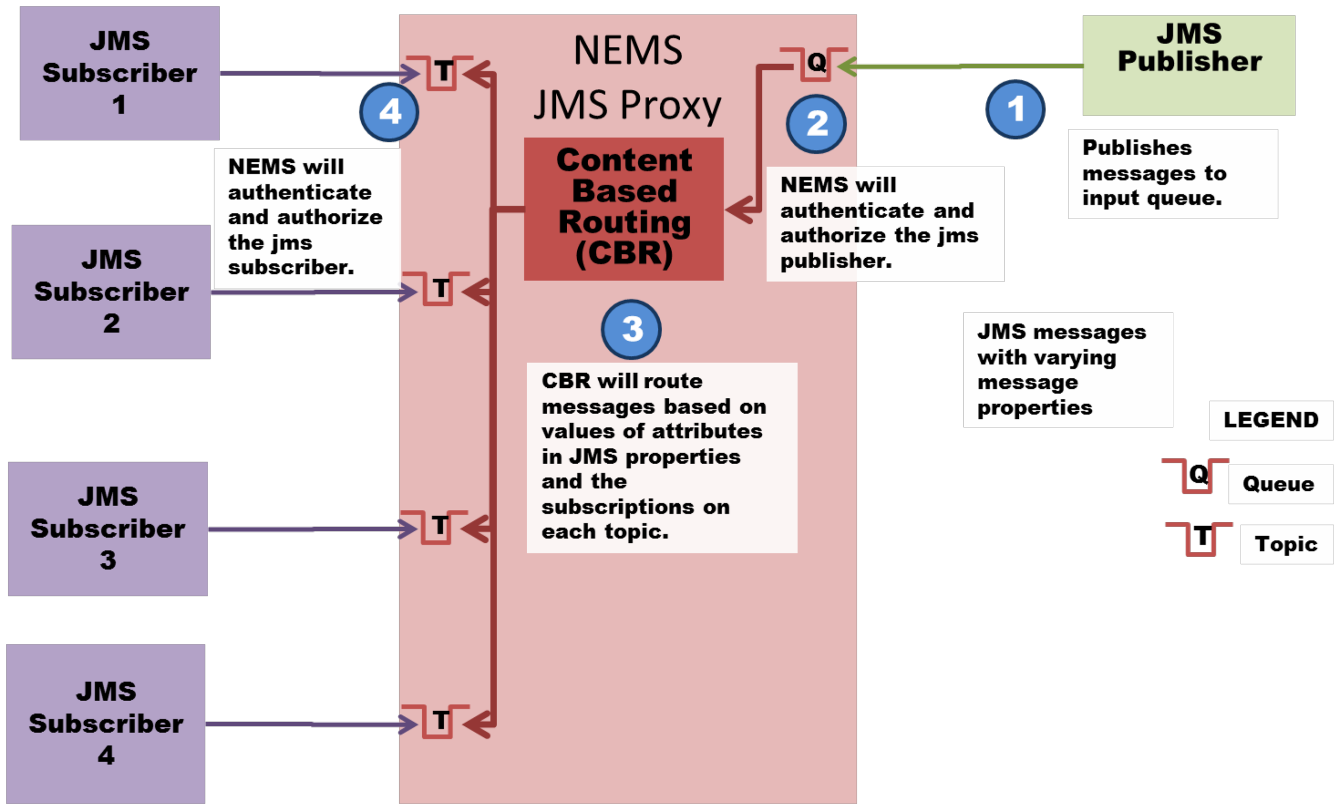
JMS Subscriber 1
JMS Subscriber 2
JMS Subscriber 3
JMS Subscriber 4
NEMS JMS Proxy
Content Based Routing (CBR)
JMS Publisher
1
Publishes messages to input queue.
2
NEMS will authenticate and authorize the jms publisher.
3
CBR will route messages based on values of attributes in JMS properties and the subscriptions on each topic.
4
NEMS will authenticate and authorize the jms subscriber.
JMS messages with varying message properties
LEGEND
Q Queue
T Topic

Figure 3. NEMS JMS Message Flow

7.1.2. SWIM in Production

The FAA uses SWIM in its production environment [7]. The following list gives some examples of what type of systems use SWIM.

- **Flight and Flow Data Systems**
  - **Time Based Flow Management System (TBMS)**: Includes Time Based Metering capability and trajectory modeler, to enhance efficiency and optimize demand and capacity.
  - **Traffic Flow Management System (TFMS)**: Data exchange system for supporting the management and monitoring of national air traffic flow. TFMS processes all available data sources such as flight plan messages, flight plan amendment messages, and departure and arrival messages.
  - **SWIM Terminal Data Distribution Systems (STDDS)**: Publishes Terminal data to NAS and non-NAS consumers.
  - **SWIM Flight Data Publication Services (SFDPS)**: Provides flight data and updates to clients for files and active flight plans.
- **Aeronautical Data**
  - **Digital NOTAM Service**: Publishes NOTAMS in AIXM 5.1.
  - **Aeronautical Information Management for the Federal NOTAM System (AIM FNS)**: System-to-system interface that enables end systems to receive digital NOTAMs from FNS.
  - **Aeronautical Information Management for Special Activity Airspace (AIM SAA)**: Provides Airport reference and configuration data, definitions and schedule information for Special Activity Airspace (SAA).

- **Weather Data**
  - **Integrated Terminal Weather System (ITWS) Data Publication**: Provides specialized weather products in the terminal area.

7.2. Previous OGC Work

Within the OGC, several approaches have been designed to implement asynchronous messaging patterns. In the following, relevant concepts are outlined.

7.2.1. Event Service

The OGC Event Service has been used from OGC Testbeds 6 to 10 to implement a Publish/Subscribe architecture. It is based on the Sensor Event Service Discussion Paper (OGC 08-133). The design of the service is heavily based on the OASIS WS-N standards and is extensible in terms of filtering capabilities.

A client can define a subscription using the OGC Filter Encoding 2.0 specification [OGC 09-026r2] which is also used by the OGC Web Feature Service 2.0 [OGC 09-025r2] as the primary filtering language. One drawback of the Event Service is the fact that it it requires clients to implement the WS-N Consumer role in order to receive messages asynchronously. It does not allow the definition of alternative dissemination methods (e.g. AMQP 1.0).

The latest state of the Event Service concept has been described in an additional Discussion Paper (OGC 11-088r1).

7.2.2. Web Event Processing Service

Within the OGC IMIS IoT Pilot an event processing architecture which relies on a OGC Web Processing Service (WPS) server has been implemented. Central element is the WPS for Event Processing (Web Event Processing Service, WEPS) which controls the overall workflow (see Figure 4).

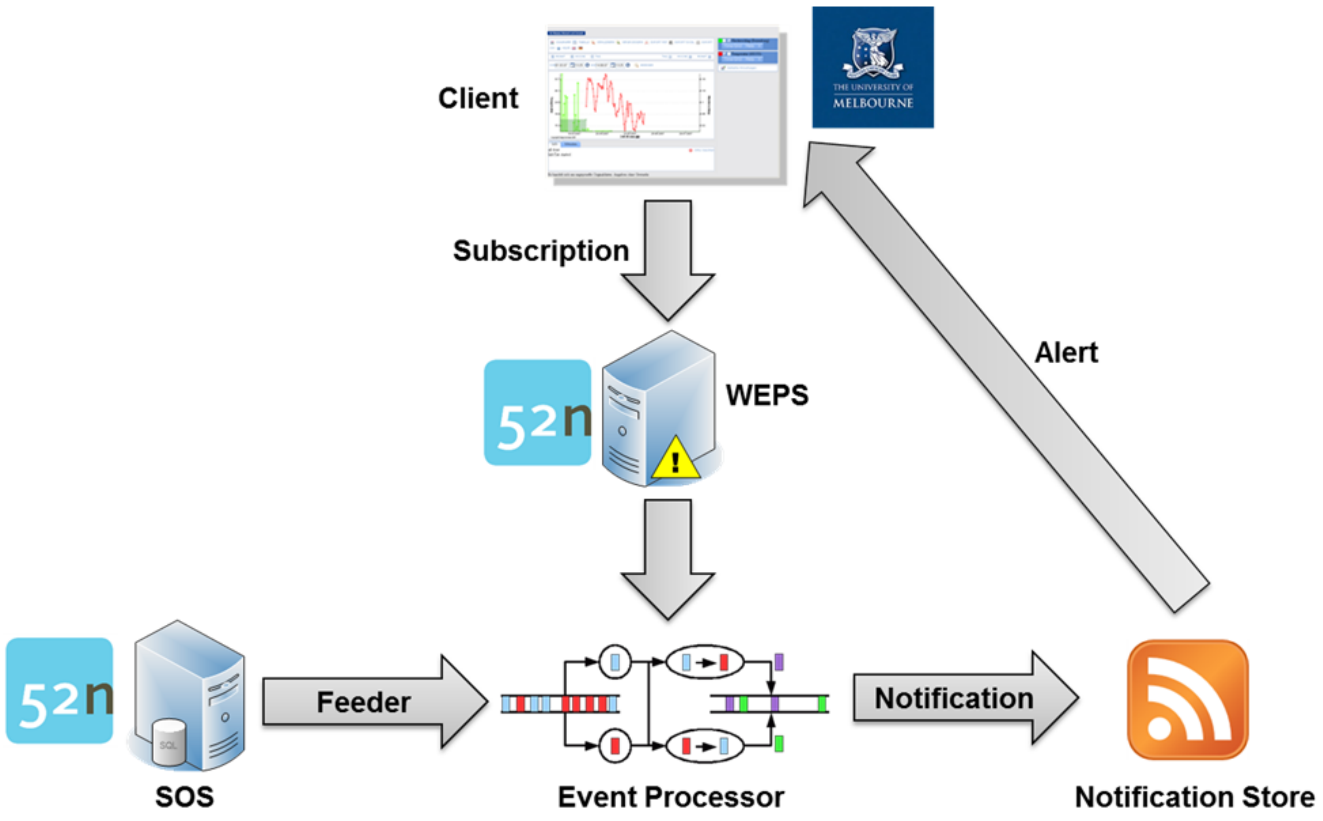

Figure 4. Web Event Processing Service (source: [OGC 15-118])

Main tasks of this WEPS are:

- Handling and managing client event subscriptions through WPS Execute requests; and
- Controlling the event processing module which performs the analysis and pattern matching of incoming sensor data streams against the event pattern rules contained in the event subscriptions.

7.2.3. OGC PubSub 1.0

The *OGC Publish/Subscribe Interface Standard 1.0* [OGC 13-131] is a recently released specification that introduces the Publish/Subscribe pattern to the OGC web services world.

> Publish/Subscribe 1.0 is an interface specification that supports the core components and concepts of the Publish/Subscribe message exchange pattern with OGC Web Services. The Publish/Subscribe pattern complements the Request/Reply pattern specified by many existing OGC Web Services. This specification may be used either in concert with, or independently of, existing OGC Web Services to publish data of interest to interested Subscribers.
>
> — OGC ® Publish/Subscribe Interface Standard 1.0 - Core

The main concepts of OGC PubSub 1.0 are:

- **Publications**: an implementing service provides information on the data it makes available for subscribing;

- **Delivery Methods**: the technological solution on how messages are transported to subscribers; and
- **Filtering Capabilities**: definition of the way a subscriber can subset data of a publication.

The following role entities are defined:

- **Publisher**: an entity that offers publications to subscribers;
- **Receiver**: an entity that receives messages from senders;
- **Sender**: an entity that sends messages to receivers;
- **Subscriber**: an entity that creates a subscription at a publisher; and
- **Subscription**: an expression of interest in (a subset of) a publication offered by a publisher.

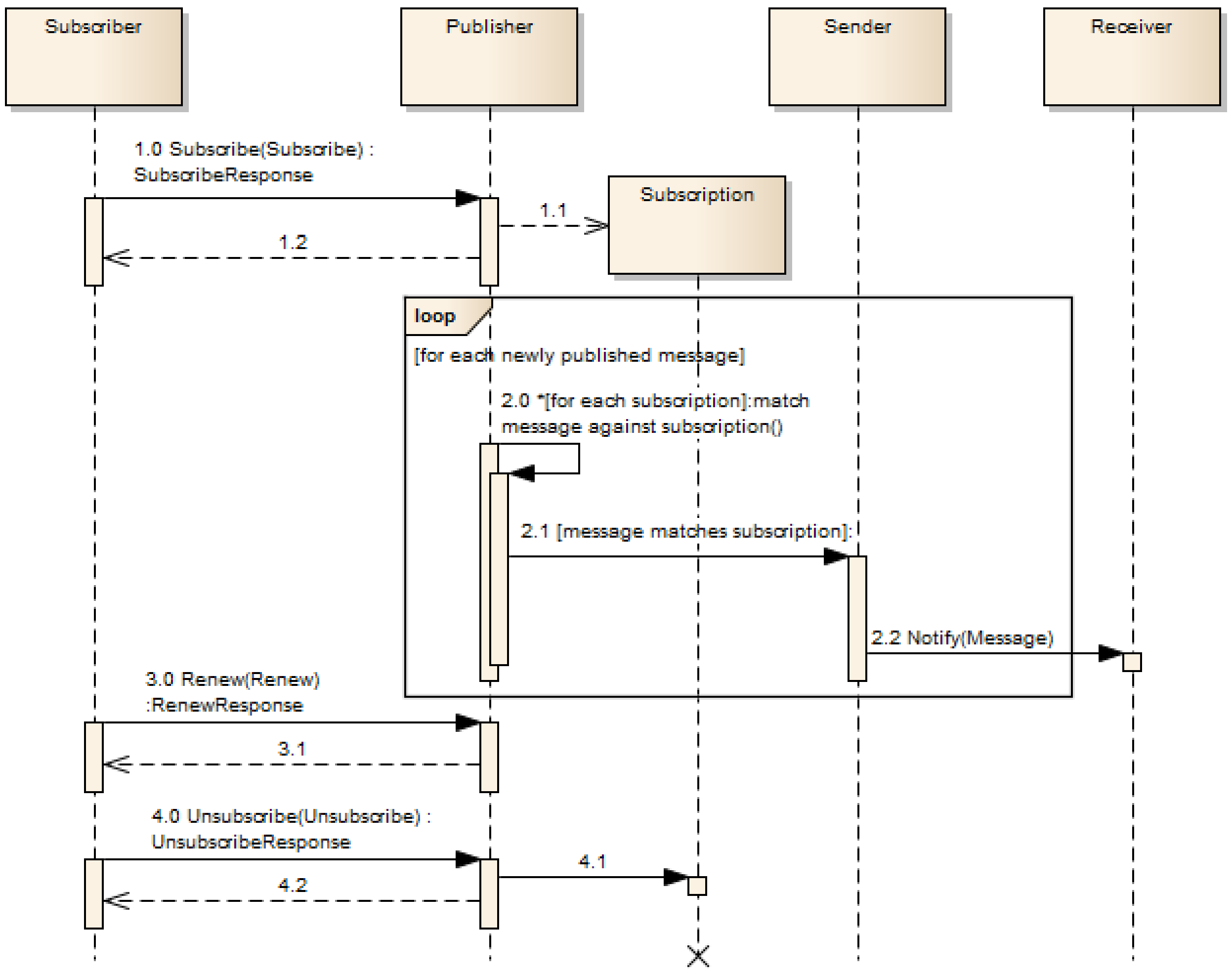
Subscriber
Publisher
Sender
Receiver
1.0 Subscribe(Subscribe) : SubscribeResponse
1.1
Subscription
1.2
loop
[for each newly published message]
2.0 *[for each subscription]:match message against subscription()
2.1 [message matches subscription]:
2.2 Notify(Message)
3.0 Renew(Renew) :RenewResponse
3.1
4.0 Unsubscribe(Unsubscribe) : UnsubscribeResponse
4.1
4.2

Figure 5. PubSub Workflow (source: OGC 13-131)

A common subscription workflow is illustrated in Figure 5. It shows how the above mentioned entities interact with each other. Within this testbed the following components have been mapped to OGC PubSub 1.0 entities.

Table 4. PubSub Mapping to Aviation components

OGC PubSub 1.0 Entity	**Aviation Architecture Component**
Subscriber	Aviation Client
Publisher	Asynchronous Messaging Server (using WFS as the backend)
Sender	Asynchronous Messaging Server
Receiver	Aviation Client

Currently only the *SOAP Protocol Binding Extension* [OGC 13-133] as a protocol-specific realization of *OGC PubSub 1.0* is available. Therefore, this binding extensions has been implemented during the testbed.

Chapter 8. Requirements

Currently, OGC Web Services only support synchronous web service request-response query capabilities. This report is required to define means to incorporate Publish/Subscribe messaging patterns for the retrieval of aviation data (i.e. AIXM and FIXM) information using geospatial queries through an AMQP 1.0 interface. The report shall demonstrate the capability using the recommended approach.

8.1. Problem Statements

Following the above requirements, these problem statements have been identified.

- How to define an OGC compliant web service that allows the management of subscriptions and the corresponding delivery of messages?
- What protocols suite best for the dissemination of Aviation data messages from data publishers to client components?
- How to define an interoperable solution that meets all requirements in terms of geospatial filtering capabilities, data dissemination and reliability?

Chapter 9. Solutions

The following sections provide an overview of the approaches that have been discussed and implemented during this testbed. The different aspects of the Publish/Subscribe architecture, such as subscription management, publication methods, message delivery, and service orchestration are outlined. Finally, recommendations for the architectural and technological approaches are described.

9.1. Targeted Solutions

9.1.1. Service Architecture

- In the analysis and requirements phase of the testbed, it has been identified that OGC PubSub 1.0 (see section OGC PubSub 1.0) and its SOAP binding extension provide a good foundation for the service-based management of subscription and dissemination of corresponding data. The specification defines means to subset data streams (using a filtering language) as well as to disseminate the resulting data streams to a client using a delivery channel of his preference. As OGC PubSub 1.0 has recently been released and fulfills all requirements, this approach has been implemented.
- Alternatives such as the OGC Event Service (see section Event Service) have been discussed. The Event Service has been used in previous testbeds' Aviation threads and also provides capabilities to manage subscriptions and subset data streams. The dissemination of data is achieved via *HTTP POST* requests. The usage of the Event Service has been discarded in favor to a OGC PubSub 1.0 service due to its lack of flexible support of different data dissemination methods.

9.1.2. Data Publication

In order to implement a Publish/Subscribe approach, a data provider (e.g. Web Feature Services, Harris Data Exchange) has to publish (updates to) data in near-real time. To achieve this, several approaches have been discussed.

- One approach is to set up a component that regularly queries a Web Feature Service using the same request and compare the response. If a change is observed, the component pushes the change to the Asynchronous Messaging Server. Such intermediary component would be the *Publisher* within the Publish/Subscribe approach. This approach has been discarded due to the high amount of business logic that would be required to identify updates to data sets. In particular, the underlying Aviation data models such as AIXM 5.1 have complex logic for dynamic features and updates to these.
- Alternatively, a WFS could implement a simple push mechanism whenever its data changes (see section Architecture). The WFS would then be the *Publisher*. This approach has been implemented within the testbed due to successful application in previous testbed and the straightforward realization (see section Patterns for Publishing Data).
- OGC PubSub 1.0 defines an architectural approach where existing data provision services can add the OGC PubSub 1.0 interface as an extension to the native service interface. This way, the service would both the *Publisher* and the *Asynchronous Messaging Server*. This approach has

been discarded as the efforts for adding such functionality to existing Web Feature Services would have exceeded the time frame of the testbed.

9.1.3. Data Dissemination

For the dissemination of data between the data publishers (e.g. Web Feature Services, Harris Data Exchange), the Asynchronous Messaging Server and clients several approaches have been discussed.

- OASIS Web Services Notifications (WS-N, see section OASIS Web Services Notification) has been successfully applied in previous testbeds and provides means to ensure reliable message transportation. A client that wants to receive data has to implement the WS-N *Consumer* role which accepts *HTTP POST* requests carrying the payload. This approach has been discarded as it has been proved working in previous testbeds already.
- AMQP 1.0 (see section Advanced Message Queuing Protocol 1.0) is a modern wire-level protocol designed for asynchronous message exchange. It has built-in support for reliable message delivery, message expiration as well as encryption and authentication. Within this testbed the message delivery using AMQP 1.0 has been chosen and implemented.

9.2. Recommendations

This report focuses on a solution which is based on OGC PubSub 1.0. The architecture and workflow described in the following sections describe the client and server components that implement against the interfaces defined by OGC PubSub 1.0. To accommodate the existing architectures (e.g. FAA SWIM's "Content Based Router") some dedicated efforts have been made to illustrates the advantages of interoperability (see FAA-specific AMQP 1.0 Profile for OGC PubSub 1.0).

9.2.1. OGC PubSub for Asynchronous Messaging

The central component for this testbeds messaging architecture is a service that implements OGC PubSub 1.0, i.e., its SOAP binding. It provides the means to manage subscriptions, evaluate these against message streams and asynchronously disseminate the messages to clients. Section Workflows and Use Cases describes the design and common workflows.

9.2.2. AMQP 1.0 Data Dissemination

To achieve asynchronous and reliable message delivery the AMQP 1.0 protocol is used (see section AMQP 1.0 Profile for OGC PubSub 1.0). The use of Message Broker software is discussed in section Message Broker Software and has been integrated into the asynchronous messaging architecture (see section Implementations).

Chapter 10. Architecture

This section provides an overview on both the overall service architecture of the Testbed-12 Aviation thread as well the parts related to the Publish/Subscribe pattern and asynchronous messaging.

10.1. General Testbed Architecture

The overall service architecture for Aviation within Testbed-12 consists of several components that can be separated into three layers.

- **Data Provider Layer**: This layer provides access to aeronautical data (AIXM, FIXM, AMXM) using *OGC Web Feature Service 2.0* (WFS) as well as *OGC Web Map Service 1.3.0* (WMS) instances.
- **Broker Layer**: The *Data Broker* service and the *Asynch Messaging Server* build the broker layer. They act as gateway services to the underlying data providers.
- **Client Layer**: The *Aviation Client* accesses the aeronautical data by communicating with the services of the broker layer.

An *OGC Catalogue Service for the Web* (CSW) service allows the client to discover relevant services and is orthogonal to the above three layers. Figure 6 illustrates the Aviation components and their interaction.

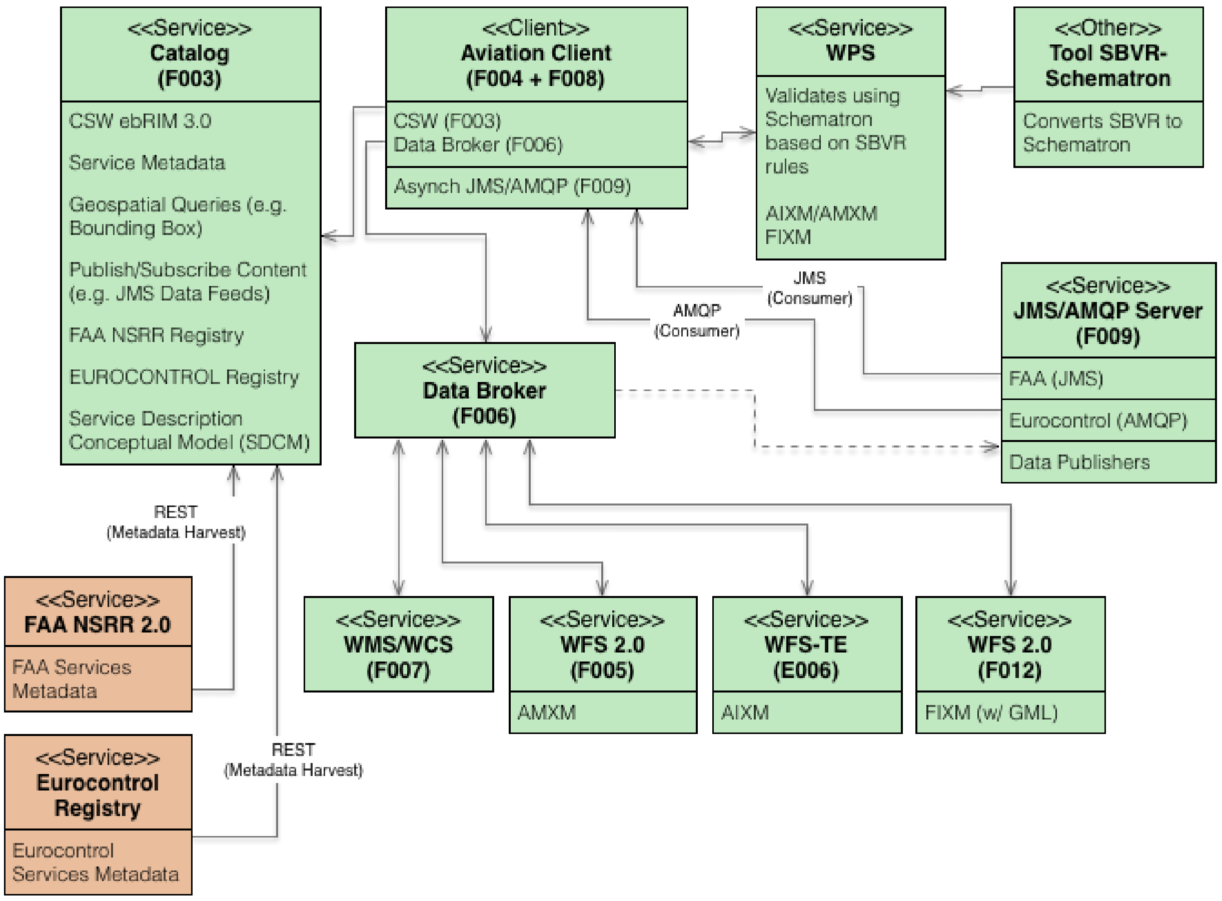

Figure 6. Aviation Service Architecture

10.2. OGC Publish/Subscribe Architecture

This document focuses on the requirements and service architecture for the asynchronous messaging parts of the overall architecture. The *Asynchronous Messaging Server* forms the basis of this system. It implements the *OGC PubSub 1.0* specification (manifestation of the its *Publisher* and *Sender* entities), allowing clients to subscribe for data using advanced filtering (e.g. spatial queries) as well as data producers to provide access to data in an event-drive fashion.

Figure 7 outlines the components and the interfaces for interaction. For message delivery the *AMQP 1.0* wire-level message protocol is used. It provides an interoperable and platform-independent way to disseminate messages.

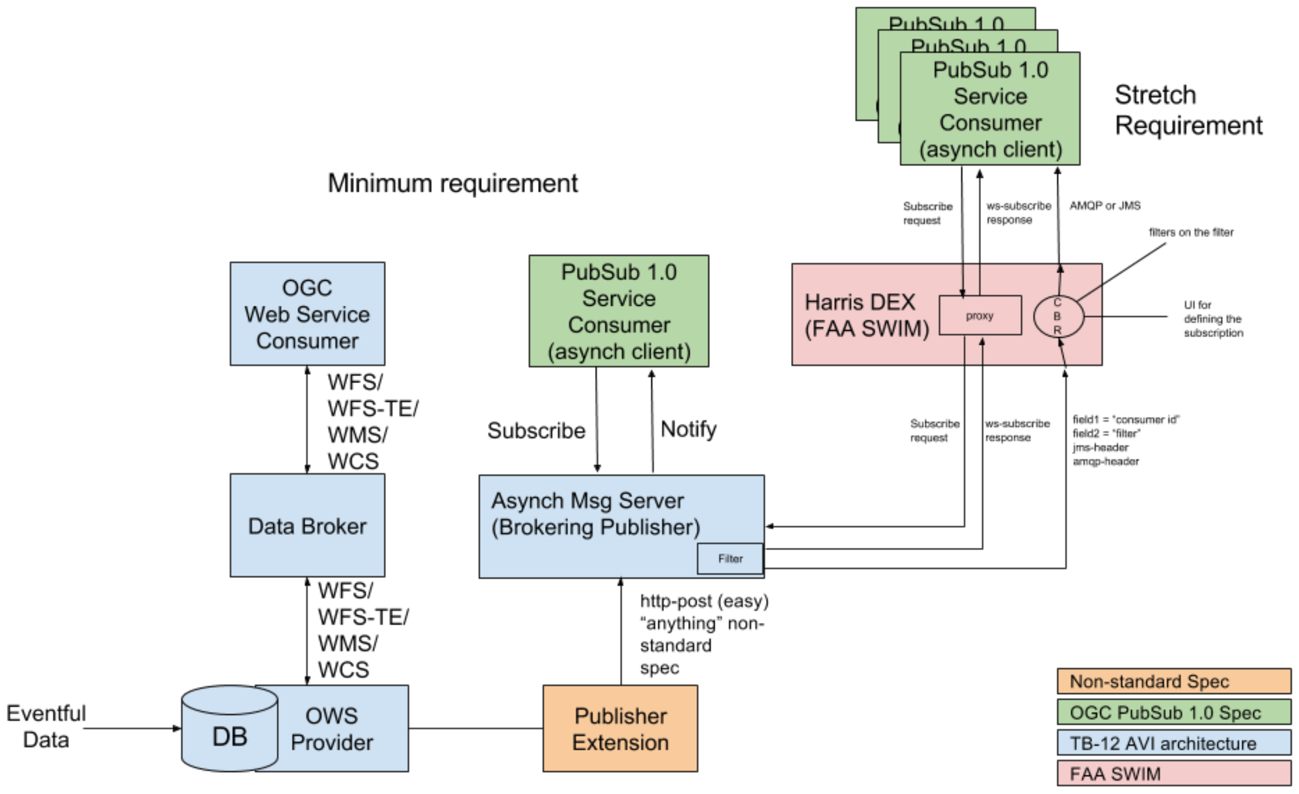

Figure 7. Publish/Subscribe and Asynchronous Messaging Architecture

Two different architectural solutions have been developed within this testbed. A basic AMQP 1.0 message delivery method and an advanced FAA-specific AMQP 1.0 method that addresses system-specific requirements.

10.3. Workflows and Use Cases

The two different approaches (basic AMQP 1.0, FAA-specific) follow different workflow patterns. The following sections provide an overview on the developed workflows and the participating software components.

10.3.1. Standalone Publisher

Figure 8 illustrates a common service-oriented setup based on OGC WFS data provisioning services. An OGC PubSub 1.0 Standalone Publisher ("PubSub Asynch Server" in the figure) acts as the brokering component (*Publisher* and *Sender* entities) in between the data provisioning and the client layer.

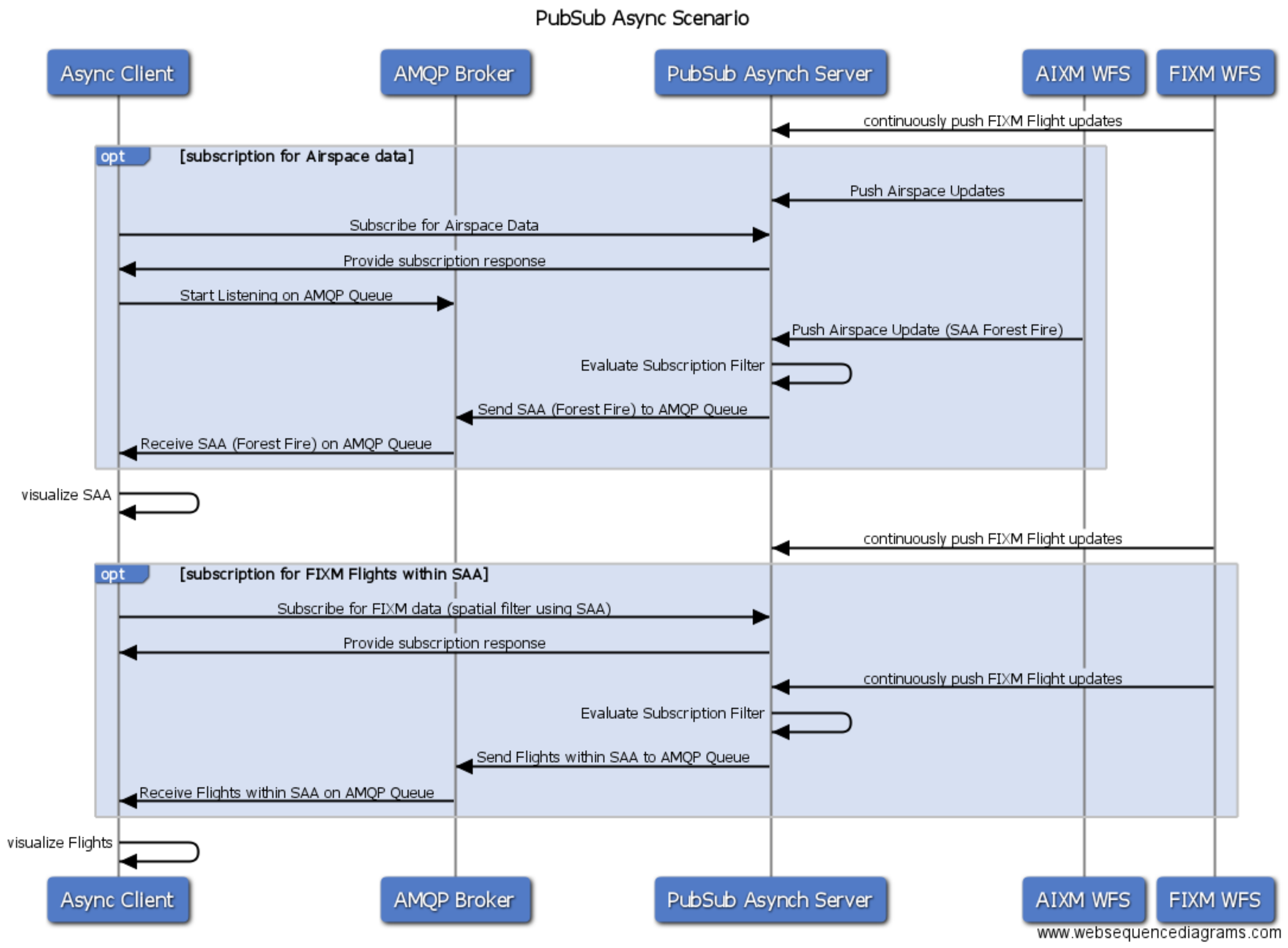
PubSub Async Scenario
Async Client
AMQP Broker
PubSub Asynch Server
AIXM WFS
FIXM WFS
continuously push FIXM Flight updates
opt
[subscription for Airspace data]
Push Airspace Updates
Subscribe for Airspace Data
Provide subscription response
Start Listening on AMQP Queue
Push Airspace Update (SAA Forest Fire)
Evaluate Subscription Filter
Send SAA (Forest Fire) to AMQP Queue
Receive SAA (Forest Fire) on AMQP Queue
visualize SAA
continuously push FIXM Flight updates
opt
[subscription for FIXM Flights within SAA]
Subscribe for FIXM data (spatial filter using SAA)
Provide subscription response
continuously push FIXM Flight updates
Evaluate Subscription Filter
Send Flights within SAA to AMQP Queue
Receive Flights within SAA on AMQP Queue
visualize Flights
www.websequencediagrams.com

Figure 8. Standalone Publisher Workflow

The *PubSub Async Server* evaluates incoming data based on the subscriptions registered by a client. In the above sequence diagram two WFS services send data updates to the *PubSub Async Server* via simple *HTTP POST* Requests. This emulates a PubSub-enabled WFS instance. OGC PubSub 1.0 defines two design patterns: the Standalone Publisher and PubSub-enabled OGC services (see section OGC PubSub 1.0). Due to some restrictions it was not possible during this testbed to develop a PubSub-enabled WFS, thus the emulation using *HTTP POST* has been established.

10.4. Harris Data Exchange Integration

DEX is a software component that is already in production within the FAA SWIM architecture. It allows the management of subscriptions and the retrieval of data (e.g. AIXM, FIXM) via JMS (see section Java Message Service). It does not support complex filtering (e.g. spatial or content-wise).

To benefit from the complex filtering capability of the *PubSub Async Server*, an integration pattern has been developed that provides both backwards compatibility for DEX clients and access to complex filtering in an interoperable way.

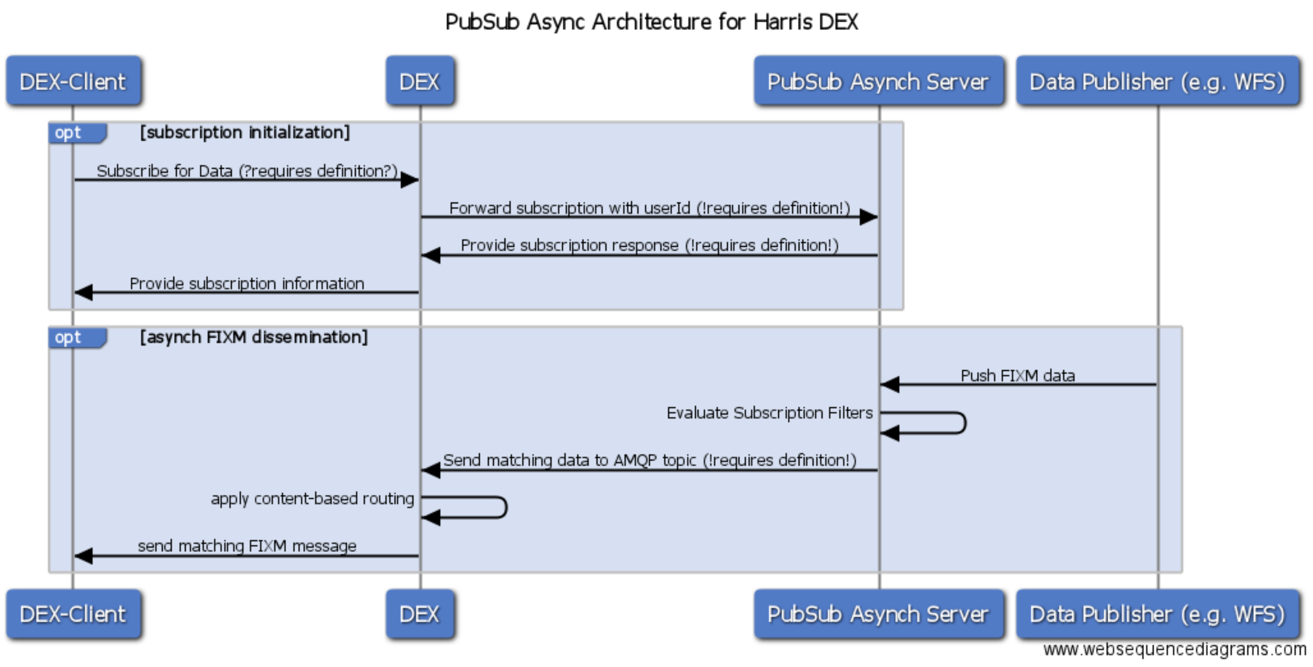
PubSub Async Architecture for Harris DEX
DEX-Client
DEX
PubSub Asynch Server
Data Publisher (e.g. WFS)
opt
[subscription initialization]
Subscribe for Data (?requires definition?)
Forward subscription with userId (!requires definition!)
Provide subscription response (!requires definition!)
Provide subscription information
opt
[asynch FIXM dissemination]
Push FIXM data
Evaluate Subscription Filters
Send matching data to AMQP topic (!requires definition!)
apply content-based routing
send matching FIXM message
www.websequencediagrams.com

Figure 9. Harris DEX Workflow

Figure 9 and Figure 7 illustrate the architecture and workflow of this integration pattern. A DEX client simply interacts with the existing DEX subscription management interface. The DEX is then responsible for converting subscriptions into an OGC PubSub 1.0 conforming structure.

As the DEX uses content-based routing (see section Harris Data Exchange) to deliver messages to the corresponding user, the *PubSub Async Server* includes the relevant information in the AMQP 1.0 message header. The details of this approach are outlined in section FAA-specific AMQP 1.0 Profile for OGC PubSub 1.0.

10.5. Subscribe Patterns

This section provides an overview on the developed patterns for subscribing to asynchronously disseminated messages. The *OGC PubSub 1.0 Core Specification* and its *SOAP binding extension* form the basis of the design. In addition, a profile for *OGC Filter Encoding 2.0* is presented, which ensures interoperability between clients and service implementations.

10.5.1. OGC PubSub 1.0

The general subscription workflow involves methods for:

- creating a subscription;
- removing a subscription; and
- renewing a subscription (updating the termination time).

The following Listing outlines the common structure of a *Subscribe* request against a OGC PubSub 1.0 service that implements the SOAP binding extension.

Creating A Subscription

Subscription using AMQP 1.0 Delivery

```xml
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:fes="http://www.opengis.net/fes/2.0"
  xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2">
   <env:Header>
      <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
   </env:Header>
   <env:Body>
      <wsn:Subscribe>
         <wsn:ConsumerReference>
           <wsa:Address>http://a.url</wsa:Address>
         </wsn:ConsumerReference>
         <wsn:Filter>
            <wsn:MessageContent Dialect="http://www.opengis.net/fes/2.0">
              ...
            </wsn:MessageContent>
         </wsn:Filter>
         <wsn:InitialTerminationTime>2016-08-18T16:30:00Z</wsn:InitialTerminationTime>
         <pubsub:PublicationIdentifier>AIXM</pubsub:PublicationIdentifier>
         <pubsub:DeliveryMethod>
            <pubsub:Identifier>http://a.delivery.method</pubsub:Identifier>
         </pubsub:DeliveryMethod>
      </wsn:Subscribe>
   </env:Body>
</env:Envelope>
```

The element `wsn:ConsumerReference` is used to define the endpoint to where matching messages shall be delivered. This depends on the delivery method chosen for the given subscription: for example a WS-N consumer would provide the *WS-N Consumer* endpoint that accepts *HTTP POST* requests, an AMQP 1.0 capable client would specify the node link URL for a topic or a queue.

A *Subscriber* can use the optional element `wsn:Filter/wsn:MessageContent` to define filter criteria which subset the stream of data of the targeted *Publication*. See section Support for Geospatial Queries for details.

The element `pubsub:PublicationIdentifier` references a valid *Publication* identifier as discovered in the services *Capabilities* document (see following section).

The response from the server is structured as follows:

Subscription Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap12:Envelope
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
    xmlns:add="http://www.w3.org/2005/08/addressing"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap12:Body>
        <wsn:SubscribeResponse>
            <wsn:SubscriptionReference>
                <add:Address>http://ows.dev.52north.org:8080/subverse-
webapp/service</add:Address>
                <add:ReferenceParameters>
                    <wsn:ConsumerReference>
                        <add:Address>http://message.endpoint.url</add:Address>
                    </wsn:ConsumerReference>
                    <pubsub:SubscriptionIdentifier>9d864279-7972-4002-bc7a-
b9db9d351a5a</pubsub:SubscriptionIdentifier>
                </add:ReferenceParameters>
            </wsn:SubscriptionReference>
            <wsn:CurrentTime>2016-07-11T09:24:44.206Z</wsn:CurrentTime>
            <wsn:TerminationTime>2016-07-11T09:30:00.000Z</wsn:TerminationTime>
        </wsn:SubscribeResponse>
    </soap12:Body>
</soap12:Envelope>
```

The `wsn:ConsumerReference` element carries the information for the AMQP 1.0 node where the client can receive the data. The identifier for later management purposes is provided in the `pubsub:SubscriptionIdentifier` element.

Removing A Subscription

Unsubscribing a previously created subscription can be achieved by sending the following request:

Removing a Subscription

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
 xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap:Body>
        <wsnt:Unsubscribe>
          <pubsub:SubscriptionIdentifier>9d864279-7972-4002-bc7a-
b9db9d351a5a</pubsub:SubscriptionIdentifier>
        </wsnt:Unsubscribe>
    </soap:Body>
</soap:Envelope>
```

The client has to provide the previously received identifier in the pubsub:SubscriptionIdentifier element.

The response from the server is a simple acknowledgment:

Unsubscribe Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap:Body>
        <wsn:UnsubscribeResponse>
            <pubsub:SubscriptionIdentifier>9d864279-7972-4002-bc7a-
b9db9d351a5a</pubsub:SubscriptionIdentifier>
        </wsn:UnsubscribeResponse>
    </soap:Body>
</soap:Envelope>
```

Publications

The PubSub specification requires that an implementing service provides a set of publications that it supports. In general, these publications are a concept to group similar message into one data stream. A client can then decide if it wants to subscribe to the whole stream of a publication or a subset of it by defining a filter.

The following publications have been defined for this testbed.

Table 5. Testbed-12 Aviation Publications

identifier	abstract
AIXM	Provides AIXM 5.1 data for OGC Testbed-12
FIXM	Provides FIXM 3.0.1 data for OGC Testbed-12
all	Provides all data (root publication)

The available *Publications* of a service are discoverable via the services *Capabilities* document. An exemplary document is appended as an Annex (see Asynchronous Messaging Server Capabilities Document) to this report.

10.5.2. Support for Geospatial Queries

To enable geospatial filtering within a subscription, a client has to provide a wsn:Filter/wsn:MessageContent (see above). To streamline the way how spatial queries are implemented across all Aviation services, the Asynchronous Messaging Server also uses the *OGC Filter Encoding 2.0* Specification. Similar to WFS services, the PubSub service describes via the *Capabilities* document which spatial operations it supports.

Filter Encoding 2.0

The following Listing shows an exemplary bounding box filter. It uses the `input/geometry` value to refer to the geometry of a feature. It is currently the responsibility of the Asynchronous Messaging Server to decide how to resolve the geometry of a feature. This approach limits the interoperability and a future work item addressing this issue is documented in section Future Work.

FES 2.0 Filter for Bounding Box

```
<fes:Filter xmlns:fes="http://www.opengis.net/fes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2">
 <fes:ValueReference>input/geometry</fes:ValueReference>
  <fes:BBOX>
     <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
        <gml:lowerCorner>-33 52</gml:lowerCorner>
        <gml:upperCorner>-32 53</gml:upperCorner>
     </gml:Envelope>
  </fes:BBOX>
</fes:Filter>
```

- supported operators (comparison, spatial, …)
- queryables (e.g. for AIXM/AMXM geometries
  - XPath, XQuery?

WFS Queries as Subscription Filters

The OGC PubSub SWG has identified the reuse of service-specific data requests as valuable concept for message filtering. A subscription could be defined using a *WFS GetFeature* query, following an analogue approach in the common request/response pattern. This support WFS Queries has been identified as out of scope for this testbed. Thus, a possible future work item has been identified and documented accordingly (see section Future Work).

10.6. AMQP 1.0 Profile for OGC PubSub 1.0

In order to achieve interoperability for AMQP 1.0 message transportation in combination with OGC PubSub 1.0, a dedicated profile is required. The current standard provides means to extend a service at the required locations. In order to interoperable interact with an OGC PubSub 1.0 service the following extensions are required:

- Extend the capabilities DeliveryCapabilities section with an AMQP 1.0 definition;
- Extend the Subscribe request with additional required parameters - or describe how to use the existing once; and
- Extend the SubscribeResponse with required information on where the AMQP 1.0 messages are delivery - or describe how to use the existing once.

This approach uses AMQP 1.0 *node links* in order to deliver messages to users. When working with complex brokering architectures (e.g. networks of messages brokers), additional metadata (i.e. the to properties field of an AMQP 1.0 message) is required. An implementation of this was out of scope within this testbed, and is considered for a future work item (see Future Work) has been documented.

10.6.1. Definitions

The profile itself requires a unique identifier. Proposal:

http://www.opengis.net/extension/PubSubAMQP/1.0/amqp10deliveryMethod

A PubSub DeliveryMethod requires and unique identifier and an abstract. Proposal:

Table 6. AMQP 1.0 Profile definitions

identifier	abstract
https://docs.oasis-open.org/amqp/core/v1.0	Advanced Message Queuing Protocol 1.0

An exemplary encoding (following the OGC PubSub 1.0 SOAP Binding) would be:

AMQP 1.0 Profile definitions XML encoding

```
<pubsub:DeliveryMethod>
    <ows:Abstract>Advanced Message Queuing Protocol 1.0</ows:Abstract>
    <pubsub:Identifier>https://docs.oasis-open.org/amqp/core/v1.0</pubsub:Identifier>
</pubsub:DeliveryMethod>
```

10.6.2. Exemplary Workflow

The following use cases illustrate possible interaction workflows between client, server and AMQP 1.0 brokers.

Use case 1: Usage of the default AMQP 1.0 broker provided by the PubSub service

1. The client retrieves the Capabilities via web service request.
2. Within the capabilities, the client identifies the existence of the AMQP 1.0 delivery method.
3. The client executes a Subscribe request by providing the AMQP 1.0 delivery method unique identifier and the default broker (as provided by the capabilities).
4. The server provides a SubscribeResponse with a dynamically created AMQP 1.0 node link that is unique among subscriptions (1-to-1 delivery).
5. The client connects to the provided AMQP 1.0 node link.
6. Once the server receives data that matches the subscription it sends this data to the AMQP 1.0 node.
7. The subscription terminates (e.g. via an Unsubscribe request, or by end of life) - no more data is send to the AMQP 1.0 node.

For step 3, additional information on the default AMQP 1.0 broker has to be provided by the capabilities. Proposal:

DeliveryMethod XML encoding extensions

```
<pubsub:DeliveryMethod>
    <ows:Abstract>Advanced Message Queuing Protocol 1.0</ows:Abstract>
    <pubsub:Identifier>https://docs.oasis-open.org/amqp/core/v1.0</pubsub:Identifier>
    <pubsub:Extension>
        <amqp:defaultHost>amqp://a.valid.amqp.node.link</amqp:defaultHost>
    </pubsub:Extension>
</pubsub:DeliveryMethod>
```

This requires a formal definition (e.g. by XML Schema). Proposal:

Namespace: http://www.opengis.net/pubsub/1.0/amqp/v1.0 **Elements**:

Table 7. DeliveryMethods extension fields

name	example values
defaultHost	ows.dev.52north.org, amqp://myserver.com, amqps://myserver.com

The client shall provide the `defaultBroker` within the Subscribe request, e.g.:

Use case 1 Subscribe Request

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:fes=
"http://www.opengis.net/fes/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:pubsub="http://www.opengis.net/pubsub/1.0" xmlns:wsa=
"http://www.w3.org/2005/08/addressing" xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
   </env:Header>
   <env:Body>
      <wsn:Subscribe>
         <wsn:ConsumerReference>
           <wsa:Address>amqp://the.default.broker.of.the.server</wsa:Address>
         </wsn:ConsumerReference>
         <wsn:InitialTerminationTime>2112-09-16T00:00:00Z</wsn:InitialTerminationTime>
         <pubsub:PublicationIdentifier>FIXM</pubsub:PublicationIdentifier>
         <pubsub:DeliveryMethod>
            <pubsub:Identifier>https://docs.oasis-
open.org/amqp/core/v1.0</pubsub:Identifier>
         </pubsub:DeliveryMethod>
      </wsn:Subscribe>
   </env:Body>
</env:Envelope>
```

The SubscribeResponse can reuse the existing elements (in particular wsn:SubscribeResponse/wsn:SubscriptionReference/add:ReferenceParameters/wsn:ConsumerReference). The server shall decide how to create a unique endpoint for the subscription. Example response:

Use case 1 Subscribe Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap12:Envelope
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
    xmlns:add="http://www.w3.org/2005/08/addressing"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap12:Body>
        <wsn:SubscribeResponse>
            <wsn:SubscriptionReference>
                <add:Address>http://ows.dev.52north.org:8080/subverse-
webapp/service</add:Address>
                <add:ReferenceParameters>
                    <wsn:ConsumerReference>
                        <add:Address>
amqp://ows.dev.52north.org/subverse.adsb.1234asdf</add:Address>
                    </wsn:ConsumerReference>
                    <pubsub:SubscriptionIdentifier>d944e8fe-6a81-4e56-883b-
c131c418bb2c</pubsub:SubscriptionIdentifier>
                </add:ReferenceParameters>
            </wsn:SubscriptionReference>
            <wsn:CurrentTime>2016-04-12T14:58:59.964Z</wsn:CurrentTime>
            <wsn:TerminationTime>2112-09-16T00:00:00.000Z</wsn:TerminationTime>
        </wsn:SubscribeResponse>
    </soap12:Body>
</soap12:Envelope>
```

The ConsumerReference would be the AMQP 1.0 node link the server and client use to exchange messages (see steps 5 and 6 above).

Use case 2: Subscriber provides an AMQP 1.0 node

If the PubSub server does not provide a default AMQP 1.0 broker or the client wants to provide an AMQP 1.0 node link on its own, the above use cannot be applied. The workflow for such situation could look like the following:

1. The client retrieves the Capabilities via web service request;
2. Within the capabilities, the client identifies the existence of the AMQP 1.0 delivery method;
3. The client executes a Subscribe by providing the AMQP 1.0 delivery method unique identifier and an AMQP 1.0 node link;
4. The server provides a SubscribeResponse, echoing the provided AMQP 1.0 node link;
5. The client is ready to accept messages on the AMQP 1.0 node link;
6. Once the server receives data that matches the subscription it sends this data to the AMQP 1.0 node link; and

7. The subscription terminates (e.g. via an Unsubscribe request, or by end of life) - no more data is published to the AMQP 1.0 node.

Example request for step 3:

Use case 2 Subscribe Request

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:fes=
"http://www.opengis.net/fes/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:pubsub="http://www.opengis.net/pubsub/1.0" xmlns:wsa=
"http://www.w3.org/2005/08/addressing" xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
   </env:Header>
   <env:Body>
      <wsn:Subscribe>
         <wsn:ConsumerReference>
           <wsa:Address>amqp://my.own.host/my-subscription</wsa:Address>
         </wsn:ConsumerReference>
         <wsn:InitialTerminationTime>2112-09-16T00:00:00Z</wsn:InitialTerminationTime>
         <pubsub:PublicationIdentifier>FIXM</pubsub:PublicationIdentifier>
         <pubsub:DeliveryMethod>
            <pubsub:Identifier>https://docs.oasis-
open.org/amqp/core/v1.0</pubsub:Identifier>
         </pubsub:DeliveryMethod>
      </wsn:Subscribe>
   </env:Body>
</env:Envelope>
```

Example response for step 4 (note the echoed ConsumerReference):

Use case 2 Subscribe Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap12:Envelope
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
    xmlns:add="http://www.w3.org/2005/08/addressing"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap12:Body>
        <wsn:SubscribeResponse>
            <wsn:SubscriptionReference>
                <add:Address>http://ows.dev.52north.org:8080/subverse-
webapp/service</add:Address>
                <add:ReferenceParameters>
                    <wsn:ConsumerReference>
                        <add:Address>amqp://my.own.host/my-subscription</add:Address>
                    </wsn:ConsumerReference>
                    <pubsub:SubscriptionIdentifier>d944e8fe-6a81-4e56-883b-
c131c418bb2c</pubsub:SubscriptionIdentifier>
                </add:ReferenceParameters>
            </wsn:SubscriptionReference>
            <wsn:CurrentTime>2016-04-12T14:58:59.964Z</wsn:CurrentTime>
            <wsn:TerminationTime>2112-09-16T00:00:00.000Z</wsn:TerminationTime>
        </wsn:SubscribeResponse>
    </soap12:Body>
```

10.7. FAA-specific AMQP 1.0 Profile for OGC PubSub 1.0

This section describes the integration pattern of the Asynchronous Messaging architecture into the FAA NEMS. The DEX is currently the component that implements the Publish/Subscribe mechanism within FAA NEMS. The concept described in this section therefore focuses on the interaction with the DEX.

The DEX uses a specialized approach to route message within the system. The DEX is a message broker implementation providing JMS as well as AMQP 1.0 delivery. It is a multi-user environment capable of routing messages based on user-specified criteria.

This routing is implemented with a content-based approach. A message sent on the internal broker contains a set of header fields that map the message to a uniquely identified user. Upon message reception the DEX analyses the header fields and forwards the message to the corresponding user.

In order to support this behavior the *Async Messaging Server* needs to implement and provide these header fields. Consequently, it can be integrated into the existing FAA/SWIM architecture. Thus, the advanced filtering methods described in this document are available within FAA/SWIM.

10.7.1. Integration Workflow

Figure 10 illustrates the integration of FAA NEMS (i.e., DEX) within the Asynchronous Messaging Architecture. In step 1 an *OGC PubSub 1.0* client performs a Subscribe request against DEX. The

Subscribe request features the specific elements required to allow the correct content-based routing of messages (i.e. the user ID as known to DEX). DEX than simply forwards the received request to the *Asynchronous Messaging Server*, thus acting as a proxy that hides the underlying architecture. The *Asynchronous Messaging Server* evaluates the received Subscribe request and acts accordingly (setting up the internal subscriptions). An acknowledgment is sent back to DEX (step 3) which is as well passed directly through to the PubSub Client (step 4).

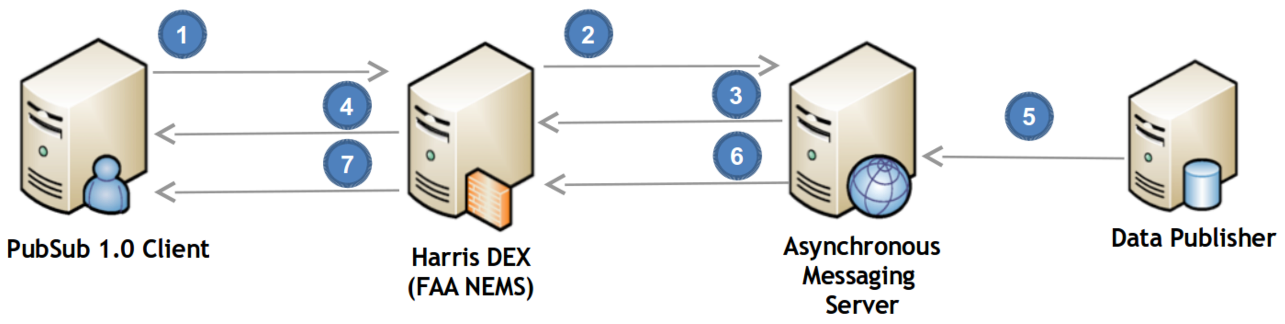

Figure 10. DEX Integration Workflow

Whenever new data from a *Data Publisher* arrives at the *Asynchronous Messaging Server* (step 5) it is evaluated against the subscription (filtering on publication, optional spatial filtering). Matching data is enriched with the header fields that are required to apply the content-based routing within DEX (i.e., the user ID) and is then pushed to DEX via AMQP 1.0 (step 6). DEX then applies its default content-based routing and delivers the message to the client via AMQP 1.0.

10.7.2. Definitions

Again, the profile itself requires a unique identifier. Proposal:

```
http://www.opengis.net/extension/PubSubAMQP/1.0/dex-over-amqp10
```

The DEX-specific profile required a dedicated DeliveryMethod defined by a unique identifier and an abstract. Proposal:

Table 8. AMQP 1.0 Profile definitions

identifier	abstract
http://www.opengis.net/extension/PubSubAMQP/1.0/dex-over-amqp10	Advanced Message Queuing Protocol 1.0 integrated via FAA NEMS (DEX)

An exemplary encoding (following the OGC PubSub 1.0 SOAP Binding) would be:

AMQP 1.0 Profile definitions XML encoding

```
<pubsub:DeliveryMethod>
    <ows:Abstract>Advanced Message Queuing Protocol 1.0 integrated via FAA NEMS
(DEX)</ows:Abstract>
    <pubsub:Identifier>http://www.opengis.net/extension/PubSubAMQP/1.0/dex-over-
amqp10</pubsub:Identifier>
</pubsub:DeliveryMethod>
```

Subscribe Definition

The subscription has to include the reference to the user ID (as known by DEX) in order to allow the *Asynchronous Messaging Server* to set the corresponding AMQP header fields. It is provided in the `dex-amqp:userId` element.

Use case 1 Subscribe Request

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:fes=
"http://www.opengis.net/fes/2.0" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:pubsub="http://www.opengis.net/pubsub/1.0" xmlns:wsa=
"http://www.w3.org/2005/08/addressing" xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
   </env:Header>
   <env:Body>
      <wsn:Subscribe>
         <wsn:ConsumerReference>
           <!-- wsa:Address left empty intentionally; allowed by schema -->
           <wsa:Address></wsa:Address>
         </wsn:ConsumerReference>
         <wsn:InitialTerminationTime>2112-09-16T00:00:00Z</wsn:InitialTerminationTime>
         <pubsub:PublicationIdentifier>FIXM</pubsub:PublicationIdentifier>
         <pubsub:DeliveryMethod>
            <pubsub:Identifier>http://www.opengis.net/extension/PubSubAMQP/1.0/dex-
over-amqp10</pubsub:Identifier>
            <pubsub:Extension>
              <dex-amqp:userId>dex-user-unique-id</dex-amqp:userId>
            </pubsub:Extension>
         </pubsub:DeliveryMethod>
      </wsn:Subscribe>
   </env:Body>
</env:Envelope>
```

As in this scenario the *Asynchronous Messaging Server* is integrated into the DEX architecture the Subscribe request does not have to state a consumer address - it shall be left blank. Example response:

Use case 1 Subscribe Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap12:Envelope
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
    xmlns:add="http://www.w3.org/2005/08/addressing"
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0">
    <soap12:Body>
        <wsn:SubscribeResponse>
            <wsn:SubscriptionReference>
                <add:Address>http://ows.dev.52north.org:8080/subverse-
webapp/service</add:Address>
                <add:ReferenceParameters>
                    <wsn:ConsumerReference>
                      <!-- wsa:Address left empty intentionally; allowed by schema -->
                      <wsa:Address></wsa:Address>
                    </wsn:ConsumerReference>
                    <pubsub:SubscriptionIdentifier>d944e8fe-6a81-4e56-883b-
c131c418bb2c</pubsub:SubscriptionIdentifier>
                </add:ReferenceParameters>
            </wsn:SubscriptionReference>
            <wsn:CurrentTime>2016-04-12T14:58:59.964Z</wsn:CurrentTime>
            <wsn:TerminationTime>2112-09-16T00:00:00.000Z</wsn:TerminationTime>
        </wsn:SubscribeResponse>
    </soap12:Body>
</soap12:Envelope>
```

10.7.3. DEX Taxonomy

The taxonomy that is required to set up the content-based routing within DEX is straight-forward. Table 9 illustrates a template for such a taxonomy. Based on this rule, the DEX is capable to route the message received from the *Asynchronous Messaging Server* (defined as the `DEX_SOURCE_TYPE`).

Table 9. Exemplary DEX Taxonomy for PubSub Integration

Taxonomy	DEX_SOURCE_TYPE	MESSAGE_TYPE	User Id
PubSub	PubSub_Async_Server	*	<the user ID as known to DEX>

10.8. Patterns for Publishing Data

This section provides details on the developed patterns for publishing data within the developed service architecture.

10.8.1. Data Providers

In order to provide publications on AIXM and FIXM, the Asynchronous Messaging Server has to have a stream of updates on these data available. As it was not designed to be a data provision service on its own, a mechanism has been developed to simulate a PubSub-enabled WFS.

The architecture-specific approach uses a simple *HTTP POST* endpoint where WFS services push updates on features to (see section Architecture). Following this approach it can be ensured that all updates are published in near-real time and in a consistent way. The data pushed to this endpoint is directly used as the stream of data for the *Publications* that are provided by the Asynchronous Messaging Server.

An alternate approach would be an intermediary component, pulling for updates in a regular interval and publishing these updates accordingly. This approach has the downside that the data stream may be inconsistent as an update could have been missed during this given interval. In addition, a consistent delay would be introduced into the architecture.

AIXM 5.1

Updates on AIXM 5.1 features are populated by pushing a Digital NOTAM (DNOTAM) message to the above described HTTP endpoint. The `AIXMBasicMessage` element is used. It contains up to three members:

1. the DNOTAM;
2. the corresponding update on the feature AIXM feature; and
3. an optional `aixm:Unit` describing the issuing institute.

An exemplary DNOTAM on a Special Activity Airspace is provided in Annex AIXM DNOTAM Document. Note that the given example is simulated data.

FIXM 3.0.1

Updates on FIXM 3.0.1 Flights are populated by pushing the given `fixm:Flight` document to the above described HTTP endpoint. A Flight object contains all relevant information such as the flight identification, the Globally Unique Flight Identifier (GUFI), the agreed route and the current position of the aircraft.

An exemplary Flight object is provided in Annex FIXM Flight Document. Note that the given example is simulated data.

10.8.2. Harris Data Exchange

A (delayed) live FIXM data stream provided by DEX has been integrated as a data publishing source. At the time of publication the implementation has not been finalized. A description of the approach will be subject of a Change Request to this report.

10.9. Delivery Methods

During the period of this testbed, different delivery methods have been defined and implemented. This section provides an overview and discusses advantages and drawbacks of each method.

10.9.1. AMQP 1.0

As AMQP 1.0 is a wire-level protocol, it does not provide application semantics or definitions on the data format. Therefore, a basic 1-to-1 communication pattern has been defined and developed during this testbed. The client provides a valid AMQP 1.0 node URL when subscribing. Matching messages are then delivered to it accordingly. The node itself could be a simple listener or a sophisticated message broker implementation such as Apache ActiveMQ or Apache Qpid.

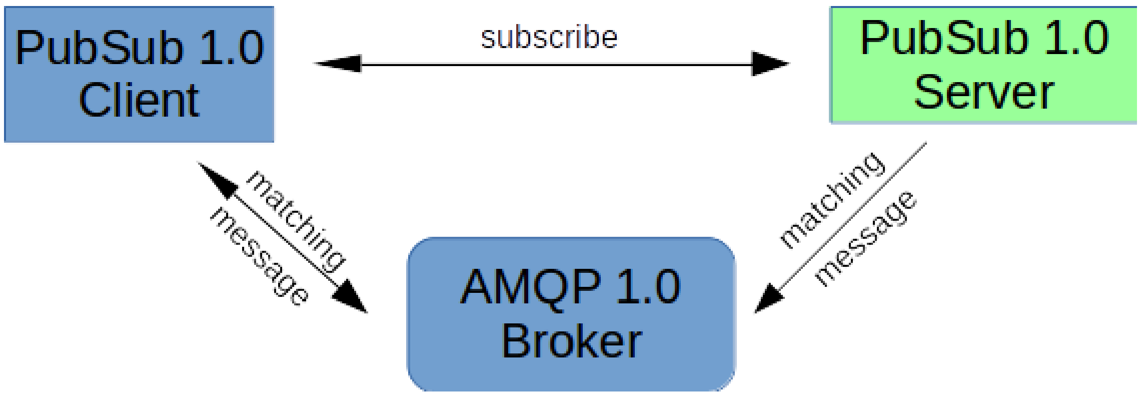

Figure 11. AMQP 1.0 Broker delivery

Figure x illustrates the interaction pattern between the client, the PubSub server, and an AMQP broker. The PubSub server does not know about the broker concept, it simply sends matching messages to the provided AMQP node URL.

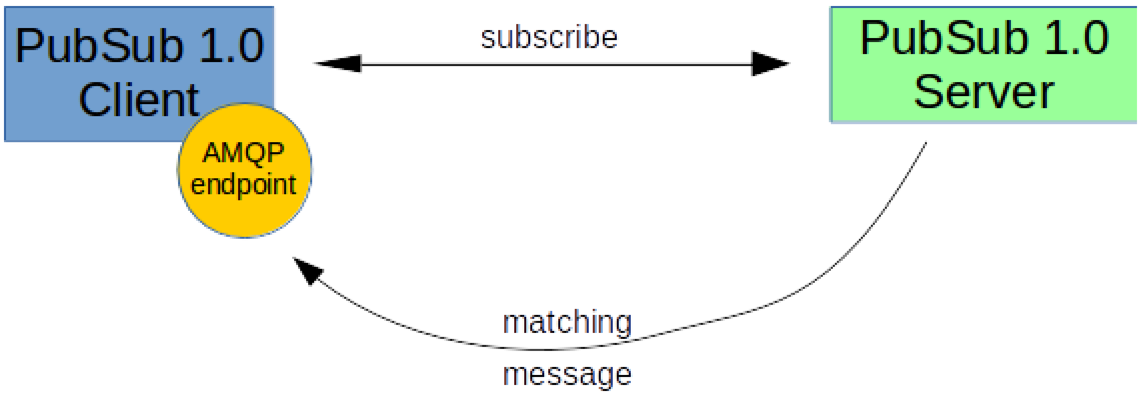

Figure 12. AMQP 1.0 Client delivery

The same behavior is illustrated in Figure x. Again, the PubSub server is agnostic about the AMQP node.

Topic Delivery

Message brokers in general provide the ability to deliver messages on topics. This can be understood as 1-to-n delivery. To support topic delivery in a certain scenario, the system design needs to ensure that the messages are send to the correct AMQP node, representing a topic.

For example, an AMQP message broker (e.g., QPID or ActiveMQ) supports the topic concept. Now, when the user subscribes to a subset of messages using the PubSub 1.0 service in conjunction with the AMQP 1.0 delivery method profile, it simply needs to provide the topic node URL.

Currently, the PubSub 1.0 AMQP profile does not support the concept on its own. Still, it might be a future work item to allow the PubSub server to communicate the ability for topic delivery, e.g., via the `PublisherCapabilities DeliveryMethods`. A dedicated field `supportsTopicDelivery` would indicate the availability of the functionality. The PubSub server has to ensure that the messages are sent to an AMQP topic node. This could be done by a service-maintained AMQP message broker instance.

10.9.2. Harris Data Exchange

The DEX system component is used within the FAA SWIM architecture. It provides AMQP 1.0 messaging capabilities. In order to support multiple users and corresponding subscriptions, a content-based routing (CBR) has been implemented. This concept requires AMQP messages to provide a set of message header fields. Using these fields, the DEX is able to route message to the correct users.

At the time of publication the implementation has not been finalized. A description of the approach will be subject of a Change Request to this report.

10.9.3. Web Services Notification

Besides the aforementioned AMQP 1.0 delivery methods, the well-established WS-N message delivery has been implemented during this testbed. Once a message matched the subscription criteria, the *PubSub Async Server* invokes the WS-N Notify method (which is the only mandatory method for a WS-N Consumer) at the provided WS-N endpoint URL. The *PubSub Async Server* again realizes the OGC PubSub 1.0 *Publisher* and *Sender* entities and the WS-N Consumer acts as the *Receiver* entity.

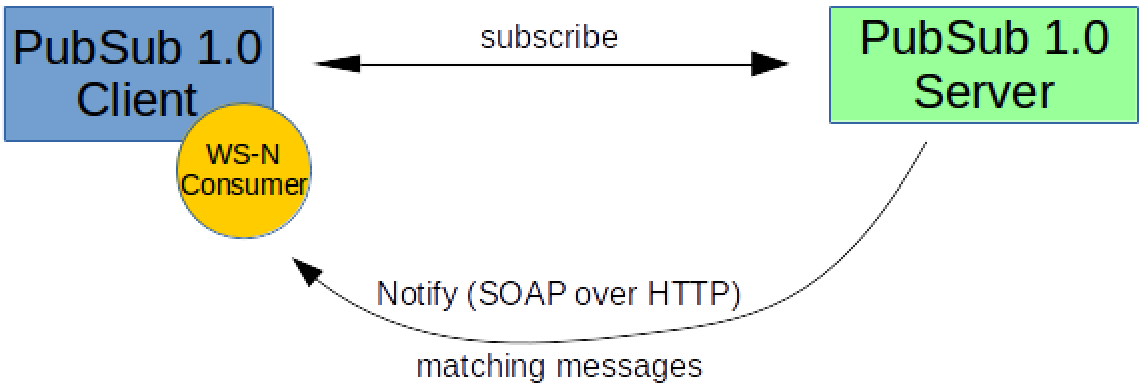

Figure 13. WS-N Client delivery

This delivery method was used by *OGC Event Service* in previous testbeds (see section Event Service). Its major downside is that a WS-N consumer has to have a publicly available endpoint URL. This is in particular problematic for client software that is run on many machines or onboard of an aircraft.

Chapter 11. Implementations

This section provides an overview on the developed software components, an analysis on the performance of the overall architecture, as well as recommendations and examples on software libraries that support the AMQP 1.0 protocol.

11.1. Services

The following services have been integrated into this testbeds system design. They play an important role in the developed architecture.

11.1.1. 52°North subverse

52°North has developed an implementation of *OGC PubSub 1.0* and its *SOAP binding extension*. The service is named `52°North subverse` [3: https://github.com/52North/subverse] and is currently designed as an OGC PubSub 1.0 standalone publisher. Thus it provides dedicated `PublisherCapabilities` and supports the required `Subscribe` and `Unsubscribe` methods.

11.1.2. Harris Data Exchange

The DEX system component is used within the FAA SWIM architecture. It provides AMQP 1.0 messaging capabilities. In order to support multiple users and corresponding subscriptions, a content-based routing (CBR) has been implemented. This concept requires AMQP 1.0 messages to provide a set of message header fields. Using these fields, the DEX is able to route message to the correct users.

During this testbed, a SWIM/DEX AMQP 1.0 message delivery profile has been developed for OGC PubSub 1.0.

Figure 14 illustrates a high-level view of how the DEX interacts with FAA systems to deliver the proper data to the proper consumers.

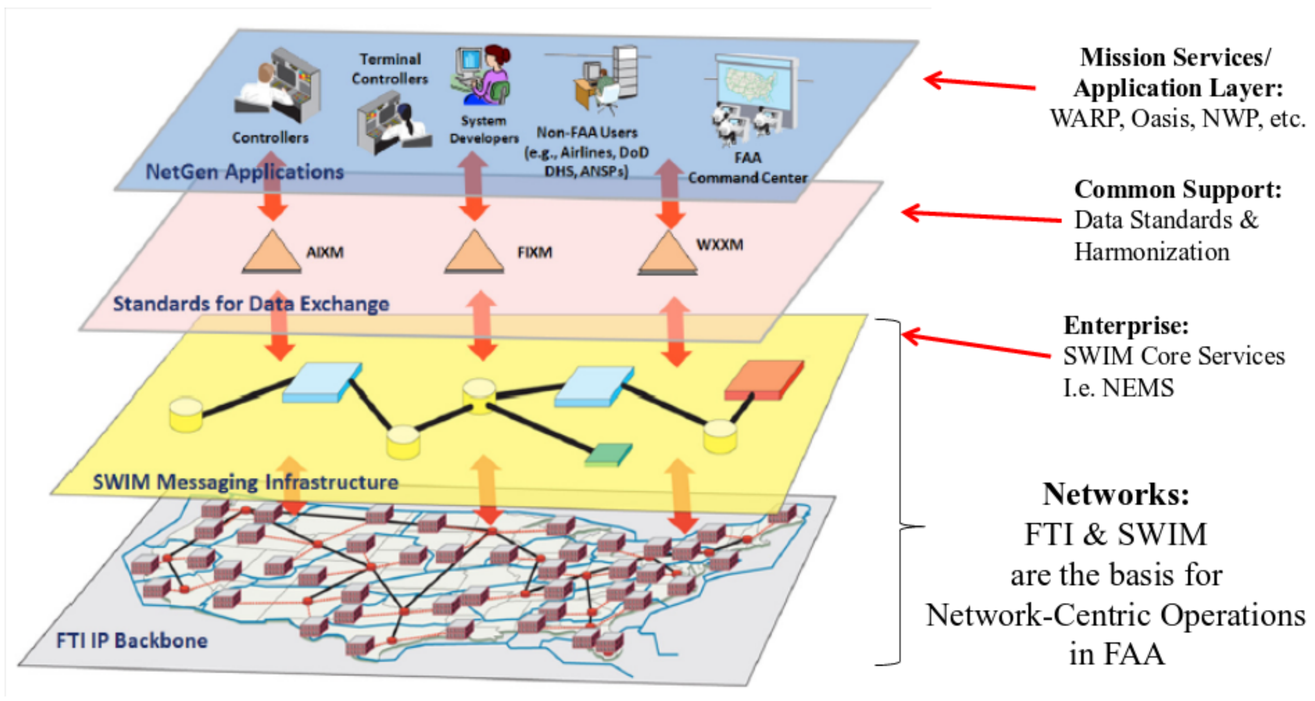
Terminal Controllers
Controllers
System Developers
Non-FAA Users (e.g., Airlines, DoD DHS, ANSPs)
FAA Command Center
NetGen Applications
AIXM
FIXM
WXXM
Standards for Data Exchange
SWIM Messaging Infrastructure
FTI IP Backbone
Mission Services/ Application Layer: WARP, Oasis, NWP, etc.
Common Support: Data Standards & Harmonization
Enterprise: SWIM Core Services I.e. NEMS
Networks: FTI & SWIM are the basis for Network-Centric Operations in FAA

Figure 14. High-Level DEX Architecture

Figure 15 illustrates the DEX architecture components.

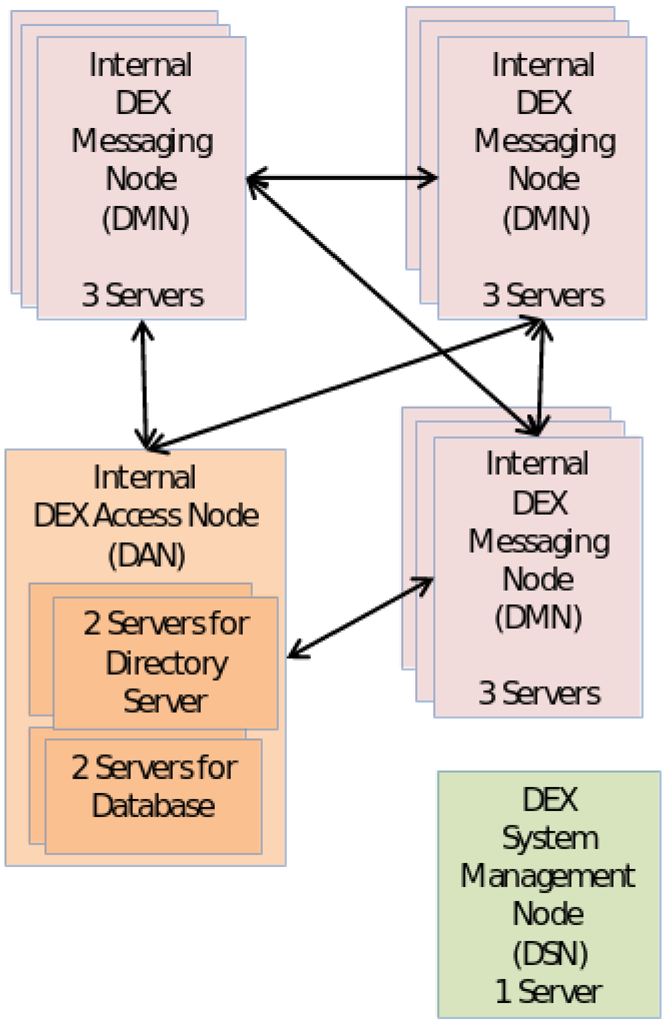
Internal DEX Messaging Node (DMN) 3 Servers
Internal DEX Messaging Node (DMN) 3 Servers
Internal DEX Access Node (DAN)
2 Servers for Directory Server
2 Servers for Database
Internal DEX Messaging Node (DMN) 3 Servers
DEX System Management Node (DSN) 1 Server

Figure 15. DEX Architecture Components

Content-Based Routing

The DEX employs a content-based routing (CBR) system to quickly route messages to their intended consumers. Publishers into DEX provide a “menu” of the data they will provide in the form of a taxonomy. Table 10 lists an example taxonomy.

Table 10. Sample DEX Producer Taxonomy

Taxonomy				**DEX_SOURCE_TYPE**	**MESSAGE_TYPE**	**ATN Type**
SFMS	*MESSAGE_TYPE*	All Messages		SFMS	*	*
		DO260B	All Messages	SFMS	DO260B	*
		ATN-DL	All Messages	SFMS	ATN-DL	*
			CM	SFMS	ATN-DL	CM
			CPDLC	SFMS	ATN-DL	CPDLC
			ADS-C	SFMS	ATN-DL	ADS-C

DEX taxonomies are customizable, allowing for fine-granular filtering of data for potential consumers. For example, the above sample taxonomy allows consumers the following options for consuming this data:

- All messages published by the SFMS system
- All DO260B messages
- All messages of type ATN-DL
- Just ATN-DL CM messages
- Just ATN-DL CPDLC messages
- Just ATN-DL ADS-C messages

When the SFMS publisher publishes messages, the messages contain header values corresponding to these taxonomy entries. Thus, CBR reads the headers, scans consumers to find those that subscribe to those header values, and routes the messages accordingly. Because CBR only needs to read the header values and not the message contents, this process is very efficient, able to process thousands of messages per second.

11.2. Clients

11.2.1. 52°North aviationFX

For assessing the architecture of the asynchronous messaging, 52°North developed a basic client for aviation: `52°North aviationFX` [4: https://github.com/52North/aviationFX]. The client supports the OGC PubSub 1.0 SOAP extensions and is thus able to interact with `52°North subverse`.

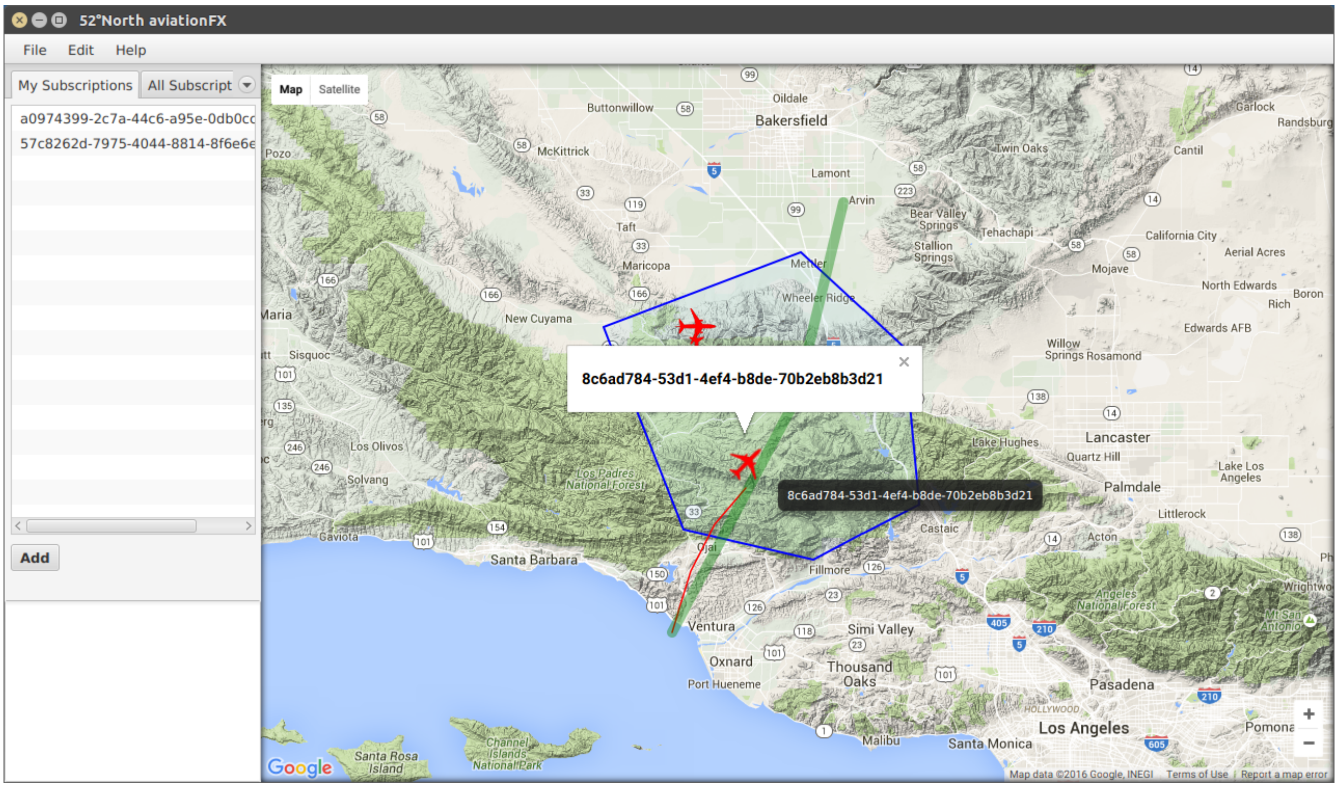

Figure 16. 52°North aviationFX Client

Figure 16 visualizes the scenario of a Special Activity Airspace with intersecting flights.

11.3. Message Broker Software

As AMQP 1.0 is a wire-level protocol the usage of a software responsible for efficient and convenient message delivery is highly beneficial. There exist different message brokering software that support the AMQP 1.0 protocol. Below is a non-exhaustive list of software projects and products.

- **Apache ActiveMQ** - an open source message broker software that supports (besides other protocols) both JMS 1.1 and AMQP 1.0 and therefore is a suitable solution for the Aviation domain (http://activemq.apache.org/).
- **Apache Qpid** - a project that offers two AMQP 1.0 message broker implementations, one written in C++, the other in Java (https://qpid.apache.org/).
- **Oracle WebLogic** - a multi-purpose COTS Java EE application server which comes with native support for JMS and AMQP 1.0 support via extensions (https://www.oracle.com/middleware/weblogic/)
- **Microsoft Azure** - a cloud computing platform that supports AMQP 1.0 in its Service Bus, thus allowing components that are built using different languages and frameworks running on different operating systems to interact with the Azure environment.

11.4. AMQP 1.0 Software Solutions

This section gives a brief overview on existing software libraries that support the AMQP 1.0 protocol. Some examples source code is provided to illustrate the workflow.

11.4.1. Apache Qpid

Besides the AMQP Broker software the Apache Qpid project also provides a client library for Java. It implements the AMQP 1.0 protocol and represents the AMQP 1.0 concepts such as messages and connections.

The Apache Qpid client library provides valuable high-level features such as connection management. The library is able to detect disconnects and reacts accordingly by reestablishing the connection to a given AMQP 1.0 broker software. This is very important for the Aviation domain in particular as reliable connections cannot be ensured for e.g., onboard client software.

The following listing illustrates the steps required to set up a consumer using the Qpid library:

Qpid Consumer

```java
import java.io.IOException;
import org.apache.qpid.proton.amqp.messaging.AmqpValue;
import org.apache.qpid.proton.amqp.messaging.ApplicationProperties;
import org.apache.qpid.proton.amqp.messaging.Section;
import org.apache.qpid.proton.message.Message;
import org.apache.qpid.proton.messenger.Messenger;

public class QpidConsumer {

    public static void main(String[] args) throws IOException {
        Messenger messenger = Messenger.Factory.create();
        messenger.start();
        while (true) {
            messenger.subscribe("amqp://localhost/my-test-queue");

            while (!messenger.stopped()) {
                System.out.println("start receiving");
                messenger.recv();
                while (messenger.incoming() > 0) {
                    System.out.println("starting receiving loop");
                    Message msg = messenger.get();
                    System.out.println("New Message with subject: "+msg.getSubject());
                    Section body = msg.getBody();
                    if (body instanceof AmqpValue) {
                        System.out.println(((AmqpValue) body).getValue());
                    }

                    ApplicationProperties ap = msg.getApplicationProperties();
                    if (ap != null && ap.getValue() != null) {
                        ap.getValue().forEach((Object key, Object value) -> {
                            System.out.println(key +"="+ value);
                        });
                    }
                }
            }
        }
    }

}
```

Publishing data to an AMQP 1.0 node is straightforward as well. The following listing gives an exemplary solution:

Qpid Publisher

```java
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import org.apache.qpid.proton.amqp.Symbol;
import org.apache.qpid.proton.amqp.messaging.AmqpValue;
import org.apache.qpid.proton.message.Message;
import org.apache.qpid.proton.messenger.Messenger;

public class QpidPublisher {

    public static void main(String[] args) throws IOException {
        Map<String, String> messageAnnotations = new HashMap<>();
        Map<String, String> deliveryAnnotations = new HashMap<>();

        Messenger messenger = Messenger.Factory.create("my-id");
        messenger.start();

        Message message = Message.Factory.create();
        message.setAddress("amqp://localhost/my-test-queue");

        message.setSubject("testing-amqp");
        message.setContentType("text/plain");

        //set message annotations
        messageAnnotations.forEach((String k, String v) -> {
            message.getMessageAnnotations().getValue().put(Symbol.valueOf(k), v);
        });

        //set delivery annotations
        deliveryAnnotations.forEach((String k, String v) -> {
            message.getDeliveryAnnotations().getValue().put(Symbol.valueOf(k), v);
        });

        message.setBody(new AmqpValue("Hello AMQP!"));

        messenger.put(message);
        messenger.send();
    }

}
```

The above source code snippets have been verified working with both Apache ActiveMQ and Apache Qpid Broker.

11.4.2. Microsoft Azure AMQP.Net Lite

AMQP.Net Lite is a lightweight AMQP 1.0 library for the .Net Framework. The library includes both a client and listener to enable peer to peer and broker based messaging. It is available as Open

Source under the Apache 2.0 license [5: https://github.com/Azure/amqpnetlite].

Chapter 12. Conclusions

The following sections describe the accomplishments achieved within this testbed work item as well as an overview on lessons learned. Details on candidate future work items are presented in a separate section (see Future Work).

12.1. Accomplishments

The requirements described in section Requirements have been addressed as follows:

- the design of a service architecture with a foundation of the *OGC PubSub 1.0* Specification: this architecture involves data publishers (WFS services), the Asynchronous Messaging Server (OGC PubSub 1.0 standalone publisher) and a client that interacts with the Asynchronous Messaging Server;
- the definition of an AMQP 1.0 delivery profile for OGC PubSub 1.0 that allows the interoperable management of subscriptions and the corresponding delivery of messages using the AMQP 1.0 protocol;
- the analysis of existing architectures that involve asynchronous messaging with a focus on the usage of Message Broker Software (such as Apache ActiveMQ);
- the development of a prototypical version of the designed architecture to proof its concept and functionality;
- the support for geospatial subscriptions using *OGC Filter Encoding 2.0* in combination with data publications for AIXM 5.1 and FIXM 3.0.1; and
- the integration of the developed solutions with existing established systems such as the Harris Data Exchange (DEX).

Appendix A: Asynchronous Messaging Server Capabilities Document

PubSub Capabilities Document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pubsub:PublisherCapabilities
    xmlns:pubsub="http://www.opengis.net/pubsub/1.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:xlin="http://www.w3.org/1999/xlink"
    xmlns:ns="http://www.opengis.net/fes/2.0"
    xmlns:amqp="http://www.opengis.net/pubsub/1.0/amqp/v1.0">
    <ows:ServiceIdentification>
        <ows:Title xml:lang="eng">Asynchronous Messaging Server</ows:Title>
        <ows:Abstract xml:lang="eng">52°North subverse Service</ows:Abstract>
        <ows:ServiceType codeSpace="">OGC:PubSub</ows:ServiceType>
        <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>

<ows:Profile>http://www.opengis.net/extension/PubSubAMQP/1.0/amqp10deliveryMethod</ows
:Profile>
        <ows:Fees>NONE</ows:Fees>
        <ows:AccessConstraints>NONE</ows:AccessConstraints>
    </ows:ServiceIdentification>
    <ows:ServiceProvider>
        <ows:ProviderName>52°North</ows:ProviderName>
        <ows:ProviderSite xlin:href="http://52north.org"/>
        <ows:ServiceContact>
            <ows:IndividualName>Matthes Rieke</ows:IndividualName>
            <ows:PositionName>Software Architect</ows:PositionName>
            <ows:ContactInfo>
                <ows:Phone>
                    <ows:Voice>+49(0)251/396 371-0</ows:Voice>
                    <ows:Facsimile>+49(0)251/396 371-11</ows:Facsimile>
                </ows:Phone>
                <ows:Address>
                    <ows:DeliveryPoint>Martin-Luther-King-Weg 24</ows:DeliveryPoint>
                    <ows:City>Münster</ows:City>
                    <ows:AdministrativeArea>North Rhine-
Westphalia</ows:AdministrativeArea>
                    <ows:PostalCode>48155</ows:PostalCode>
                    <ows:Country>Germany</ows:Country>
                    <ows:ElectronicMailAddress>
info@52north.org</ows:ElectronicMailAddress>
                </ows:Address>
            </ows:ContactInfo>
        </ows:ServiceContact>
    </ows:ServiceProvider>
    <ows:OperationsMetadata>
        <ows:Operation name="GetCapabilities">
            <ows:DCP>
                <ows:HTTP>
                    <ows:Get xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service?">
                        <ows:Constraint name="Content-Type">
                            <ows:AllowedValues>
                                <ows:Value>application/x-kvp</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                    </ows:Get>
                </ows:HTTP>
            </ows:DCP>
            <ows:Parameter name="AcceptVersions"/>
        </ows:Operation>
        <ows:Operation name="GetSubscription">
            <ows:DCP>
                <ows:HTTP>
                    <ows:Post xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service">
                        <ows:Constraint name="Content-Type">
                            <ows:AllowedValues>
                                <ows:Value>application/xml</ows:Value>
                                <ows:Value>text/xml</ows:Value>
                                <ows:Value>application/soap+xml</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                        <ows:Constraint name="PostEncoding">
                            <ows:AllowedValues>
                                <ows:Value>SOAP</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                    </ows:Post>
                </ows:HTTP>
            </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Notify">
            <ows:DCP>
                <ows:HTTP>
                    <ows:Post xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service">
                        <ows:Constraint name="Content-Type">
                            <ows:AllowedValues>
                                <ows:Value>application/soap+xml</ows:Value>
                                <ows:Value>application/xml</ows:Value>
                                <ows:Value>text/xml</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                        <ows:Constraint name="PostEncoding">
                        <ows:AllowedValues>
                            <ows:Value>SOAP</ows:Value>
                        </ows:AllowedValues>
                    </ows:Constraint>
                </ows:Post>
            </ows:HTTP>
        </ows:DCP>
    </ows:Operation>
    <ows:Operation name="Renew">
        <ows:DCP>
            <ows:HTTP>
                <ows:Post xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service">
                    <ows:Constraint name="Content-Type">
                        <ows:AllowedValues>
                            <ows:Value>application/soap+xml</ows:Value>
                            <ows:Value>application/xml</ows:Value>
                            <ows:Value>text/xml</ows:Value>
                        </ows:AllowedValues>
                    </ows:Constraint>
                    <ows:Constraint name="PostEncoding">
                        <ows:AllowedValues>
                            <ows:Value>SOAP</ows:Value>
                        </ows:AllowedValues>
                    </ows:Constraint>
                </ows:Post>
            </ows:HTTP>
        </ows:DCP>
    </ows:Operation>
    <ows:Operation name="Subscribe">
        <ows:DCP>
            <ows:HTTP>
                <ows:Post xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service">
                    <ows:Constraint name="Content-Type">
                        <ows:AllowedValues>
                            <ows:Value>application/soap+xml</ows:Value>
                            <ows:Value>application/xml</ows:Value>
                            <ows:Value>text/xml</ows:Value>
                        </ows:AllowedValues>
                    </ows:Constraint>
                    <ows:Constraint name="PostEncoding">
                        <ows:AllowedValues>
                            <ows:Value>SOAP</ows:Value>
                        </ows:AllowedValues>
                    </ows:Constraint>
                </ows:Post>
            </ows:HTTP>
        </ows:DCP>
    </ows:Operation>
    <ows:Operation name="Unsubscribe">
```

```
            <ows:DCP>
                <ows:HTTP>
                    <ows:Post xlin:href="http://ows.dev.52north.org:8080/subverse-
webapp/service">
                        <ows:Constraint name="Content-Type">
                            <ows:AllowedValues>
                                <ows:Value>application/soap+xml</ows:Value>
                                <ows:Value>application/xml</ows:Value>
                                <ows:Value>text/xml</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                        <ows:Constraint name="PostEncoding">
                            <ows:AllowedValues>
                                <ows:Value>SOAP</ows:Value>
                            </ows:AllowedValues>
                        </ows:Constraint>
                    </ows:Post>
                </ows:HTTP>
            </ows:DCP>
        </ows:Operation>
        <ows:Parameter name="GetCapabilities"/>
        <ows:Parameter name="PubSub"/>
        <ows:Parameter name="1.0.0"/>
    </ows:OperationsMetadata>
    <pubsub:FilterCapabilities>
        <pubsub:FilterLanguage>
            <ows:Abstract>OGC Filter Encoding Spec 2.0</ows:Abstract>
            <pubsub:Identifier>http://www.opengis.net/fes/2.0</pubsub:Identifier>
            <pubsub:SupportedCapabilities>
                <ns:Filter_Capabilities>
                    <ns:Conformance>
                        <ns:Constraint name="ImplementsMinSpatialFilter">
                            <ows:NoValues/>
                            <ows:DefaultValue>true</ows:DefaultValue>
                        </ns:Constraint>
                        <ns:Constraint name="ImplementsTemporalFilter">
                            <ows:NoValues/>
                            <ows:DefaultValue>true</ows:DefaultValue>
                        </ns:Constraint>
                    </ns:Conformance>
                </ns:Filter_Capabilities>
            </pubsub:SupportedCapabilities>
        </pubsub:FilterLanguage>
    </pubsub:FilterCapabilities>
    <pubsub:DeliveryCapabilities>
        <pubsub:DeliveryMethod>
            <ows:Abstract>Advanced Message Queuing Protocol 1.0</ows:Abstract>
            <pubsub:Identifier>https://docs.oasis-
open.org/amqp/core/v1.0</pubsub:Identifier>
            <pubsub:Extension>
                <amqp:defaultHost>ows.dev.52north.org</amqp:defaultHost>
            </pubsub:Extension>
        </pubsub:DeliveryMethod>
        <pubsub:DeliveryMethod>
            <ows:Abstract>WS-BaseNotification allows a NotificationConsumer to receive
a Notification in one of two forms:

 1. The NotificationConsumer MAY simply receive the "raw" Notification (i.e. the
application-specific content).

 2. The NotificationConsumer MAY receive the Notification data as a Notify message as
described below.</ows:Abstract>
            <pubsub:Identifier>http://docs.oasis-open.org/wsn/b-
2/NotificationConsumer</pubsub:Identifier>
        </pubsub:DeliveryMethod>
    </pubsub:DeliveryCapabilities>
    <pubsub:Publications>
        <pubsub:Publication>
            <ows:Abstract>provides all data (root publication)</ows:Abstract>
            <pubsub:Identifier>all</pubsub:Identifier>
        </pubsub:Publication>
        <pubsub:Publication>
            <ows:Abstract>provides AIXM 5.1 data as DNOTAM</ows:Abstract>
            <pubsub:Identifier>AIXM</pubsub:Identifier>
        </pubsub:Publication>
        <pubsub:Publication>
            <ows:Abstract>provides FIXM 3.0.1 data</ows:Abstract>
            <pubsub:Identifier>FIXM</pubsub:Identifier>
        </pubsub:Publication>
    </pubsub:Publications>
</pubsub:PublisherCapabilities>
```

Appendix B: AIXM DNOTAM Document

Exemplary DNOTAM for a Special Activity Airspace

```
<?xml version='1.0' encoding='UTF-8'?>
<message:AIXMBasicMessage xmlns:message="http://www.aixm.aero/schema/5.1/message"
xmlns:gts="http://www.isotc211.org/2005/gts" xmlns:gco=
"http://www.isotc211.org/2005/gco" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:gss="
http://www.isotc211.org/2005/gss" xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:gsr="http://www.isotc211.org/2005/gsr" xmlns:gmd=
"http://www.isotc211.org/2005/gmd" xmlns:event="http://www.aixm.aero/schema/5.1/event"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.aixm.aero/schema/5.1/message
http://www.aixm.aero/schema/5.1/message/AIXM_BasicMessage.xsd
http://www.aixm.aero/schema/5.1/event
http://www.aixm.aero/schema/5.1/event/version_5.1-e/Event_Features.xsd" gml:id="
M00001">
  <message:hasMember>
    <event:Event gml:id="uuid.cc1bdbc7-476c-4b04-81c0-8f5e7ccb454b">
      <gml:identifier codeSpace="urn:uuid:">cc1bdbc7-476c-4b04-81c0-
8f5e7ccb454b</gml:identifier>
      <event:timeSlice>
        <event:EventTimeSlice gml:id="ID_51">
          <gml:validTime>
            <gml:TimeInstant gml:id="ID_52">
              <gml:timePosition>2016-08-28T06:00:00Z</gml:timePosition>
            </gml:TimeInstant>
          </gml:validTime>
          <aixm:interpretation>PERMDELTA</aixm:interpretation>
          <aixm:sequenceNumber>1</aixm:sequenceNumber>
          <aixm:featureLifetime>
            <gml:TimePeriod gml:id="ID_53">
              <gml:beginPosition>2016-08-28T06:00:00Z</gml:beginPosition>
              <gml:endPosition>2016-08-30T22:30:00Z</gml:endPosition>
            </gml:TimePeriod>
          </aixm:featureLifetime>
          <event:name>EBD04_20140630</event:name>
          <event:encoding>DIGITAL</event:encoding>
          <event:scenario>SAA.ACT</event:scenario>
          <event:version>2.0</event:version>
          <event:revision>2016-08-28T07:49:54</event:revision>
          <event:textNOTAM>
            <event:NOTAM gml:id="ID_54">
              <event:text>DANGER AREA LEB04 LEBEC ACTIVE, FOREST FIRE</event:text>
              <event:lowerLimit>SFC</event:lowerLimit>
```

```
                <event:upperLimit>FL170</event:upperLimit>
                <event:publisherNOF xlink:href="#uuid.6bdb97aa-3f04-41bc-ac10-
64cede76b9ba" xlink:title="STATE NOTAM OFFICE" xsi:nil="true"/>
              </event:NOTAM>
            </event:textNOTAM>
          </event:EventTimeSlice>
        </event:timeSlice>
        <event:timeSlice>
          <event:EventTimeSlice gml:id="ID_56">
            <gml:validTime>
              <gml:TimePeriod gml:id="ID_57">
                <gml:beginPosition>2016-08-28T06:00:00Z</gml:beginPosition>
                <gml:endPosition>2016-08-30T22:30:00Z</gml:endPosition>
              </gml:TimePeriod>
            </gml:validTime>
            <aixm:interpretation>BASELINE</aixm:interpretation>
            <aixm:sequenceNumber>1</aixm:sequenceNumber>
            <aixm:featureLifetime>
              <gml:TimePeriod gml:id="ID_58">
                <gml:beginPosition>2016-08-28T06:00:00Z</gml:beginPosition>
                <gml:endPosition>2016-08-30T22:30:00Z</gml:endPosition>
              </gml:TimePeriod>
            </aixm:featureLifetime>
            <event:name>LEB04_20160828</event:name>
            <event:encoding>DIGITAL</event:encoding>
            <event:scenario>SAA.ACT</event:scenario>
            <event:version>2.0</event:version>
            <event:revision>2016-08-28T07:49:54</event:revision>
            <event:textNOTAM>
              <event:NOTAM gml:id="ID_55">
                <event:text>DANGER AREA LEB04 LEBEC ACTIVE, FOREST FIRE</event:text>
                <event:lowerLimit>SFC</event:lowerLimit>
                <event:upperLimit>FL170</event:upperLimit>
                <event:publisherNOF xlink:href="#uuid.6bdb97aa-3f04-41bc-ac10-
64cede76b9ba" xlink:title="STATE NOTAM OFFICE" xsi:nil="true"/>
              </event:NOTAM>
            </event:textNOTAM>
          </event:EventTimeSlice>
        </event:timeSlice>
      </event:Event>
    </message:hasMember>
    <message:hasMember>
      <aixm:Airspace gml:id="uuid.28e6905-f99a-4ca7-a736-2c0787cdcf57">
        <gml:identifier codeSpace="urn:uuid:">028e6905-f99a-4ca7-a736-
2c0787cdcf57</gml:identifier>
        <aixm:timeSlice>
          <aixm:AirspaceTimeSlice gml:id="ID_65">
            <gml:validTime>
              <gml:TimeInstant gml:id="ID_66">
                <gml:timePosition>2016-08-28T07:49:54Z</gml:timePosition>
              </gml:TimeInstant>
          </gml:validTime>
          <aixm:interpretation>SNAPSHOT</aixm:interpretation>
          <aixm:type>D</aixm:type>
          <aixm:designator>LEB04</aixm:designator>
          <aixm:name>LEBEC</aixm:name>
          <aixm:geometryComponent>
            <aixm:AirspaceGeometryComponent gml:id="ID_67">
              <aixm:operationSequence>1</aixm:operationSequence>
              <aixm:theAirspaceVolume>
                <aixm:AirspaceVolume gml:id="ID_68">
                  <aixm:upperLimit uom="FL">170</aixm:upperLimit>
                  <aixm:upperLimitReference>OTHER</aixm:upperLimitReference>
                  <aixm:lowerLimit uom="FT">0</aixm:lowerLimit>
                  <aixm:lowerLimitReference>SFC</aixm:lowerLimitReference>
                  <aixm:horizontalProjection>
                    <aixm:Surface srsName="urn:ogc:def:crs:EPSG::4326" gml:id="ID_69">
                      <gml:patches>
                        <gml:PolygonPatch>
                          <gml:exterior>
                            <gml:LinearRing>
                              <gml:posList>34.54163119530972 -118.67980957031249
                                34.420504880133834 -118.95996093749999
                                34.4861839632883 -119.30465698242186
                                34.92760087214065 -119.5147705078125
                                35.08957427943165 -118.99291992187499
                                34.88142481679758 -118.71826171875
                                34.54163119530972 -118.67980957031249</gml:posList>
                            </gml:LinearRing>
                          </gml:exterior>
                        </gml:PolygonPatch>
                      </gml:patches>
                    </aixm:Surface>
                  </aixm:horizontalProjection>
                </aixm:AirspaceVolume>
              </aixm:theAirspaceVolume>
            </aixm:AirspaceGeometryComponent>
          </aixm:geometryComponent>
        </aixm:AirspaceTimeSlice>
      </aixm:timeSlice>
      <aixm:timeSlice>
        <aixm:AirspaceTimeSlice gml:id="ID_70">
          <gml:validTime>
            <gml:TimePeriod gml:id="ID_71">
              <gml:beginPosition>2016-08-28T06:00:00Z</gml:beginPosition>
              <gml:endPosition>2016-08-30T22:30:00Z</gml:endPosition>
            </gml:TimePeriod>
          </gml:validTime>
          <aixm:interpretation>TEMPDELTA</aixm:interpretation>
          <aixm:sequenceNumber>1</aixm:sequenceNumber>
          <aixm:type>P</aixm:type> <!-- see coding rule referring to terms used in the
information from originator that the area is prohibited -->
```

```
          <aixm:activation>
            <aixm:AirspaceActivation gml:id="ID_72">
              <aixm:annotation>
                <aixm:Note gml:id="ID_74">
                  <aixm:purpose>REMARK</aixm:purpose>
                  <aixm:translatedNote>
                    <aixm:LinguisticNote gml:id="ID_75">
                      <aixm:note>SAR activities during fire fighting</aixm:note>
                    </aixm:LinguisticNote>
                  </aixm:translatedNote>
                </aixm:Note>
              </aixm:annotation>
              <aixm:annotation>
                <aixm:Note gml:id="ID_76">
                  <aixm:purpose>REMARK</aixm:purpose>
                  <aixm:translatedNote>
                    <aixm:LinguisticNote gml:id="ID_77">
                      <aixm:note>Prohibited for all manned military and civilian
aircraft during fire fighting activity.</aixm:note>
                    </aixm:LinguisticNote>
                  </aixm:translatedNote>
                </aixm:Note>
              </aixm:annotation>
              <aixm:annotation>
                <aixm:Note gml:id="ID_78">
                  <aixm:purpose>REMARK</aixm:purpose>
                  <aixm:translatedNote>
                    <aixm:LinguisticNote gml:id="ID_79">
                      <aixm:note>More info can be obtained via EBMIZGZF.</aixm:note>
                    </aixm:LinguisticNote>
                  </aixm:translatedNote>
                </aixm:Note>
              </aixm:annotation>
              <aixm:activity>SAR</aixm:activity>
              <aixm:status>ACTIVE</aixm:status>
              <aixm:levels>
                <aixm:AirspaceLayer gml:id="ID_73">
                  <aixm:upperLimit uom="OTHER">CEILING</aixm:upperLimit>
                  <aixm:lowerLimit uom="OTHER">FLOOR</aixm:lowerLimit>
                </aixm:AirspaceLayer>
              </aixm:levels>
            </aixm:AirspaceActivation>
          </aixm:activation>
          <aixm:extension>
            <event:AirspaceExtension gml:id="ID_80">
              <event:theEvent xlink:href="#uuid.cc1bdbc7-476c-4b04-81c0-8f5e7ccb454b"
xsi:nil="true"/>
            </event:AirspaceExtension>
          </aixm:extension>
        </aixm:AirspaceTimeSlice>
      </aixm:timeSlice>
```

```xml
    </aixm:Airspace>
  </message:hasMember>
  <message:hasMember>
    <aixm:Unit gml:id="uuid.uuid.6bdb97aa-3f04-41bc-ac10-64cede76b9ba">
      <gml:identifier codeSpace="urn:uuid:">uuid.6bdb97aa-3f04-41bc-ac10-
64cede76b9ba"</gml:identifier>
      <aixm:timeSlice>
        <aixm:UnitTimeSlice gml:id="ID_61">
          <gml:validTime>
            <gml:TimeInstant gml:id="ID_62">
              <gml:timePosition>2016-08-28T07:49:54Z</gml:timePosition>
            </gml:TimeInstant>
          </gml:validTime>
          <aixm:interpretation>SNAPSHOT</aixm:interpretation>
          <aixm:name>STATE NOTAM OFFICE</aixm:name>
          <aixm:type>NOF</aixm:type>
          <aixm:contact>
            <aixm:ContactInformation gml:id="ID_63">
              <aixm:phoneFax>
                <aixm:TelephoneContact gml:id="ID_64">
                  <aixm:voice>(12) 345 67 89</aixm:voice>
                </aixm:TelephoneContact>
              </aixm:phoneFax>
            </aixm:ContactInformation>
          </aixm:contact>
        </aixm:UnitTimeSlice>
      </aixm:timeSlice>
    </aixm:Unit>
  </message:hasMember>
</message:AIXMBasicMessage>
```

Appendix C: FIXM Flight Document

Exemplary FIXM Flight Object

```
<fx:Flight xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:fx="http://www.fixm.aero/flight/3.0"
xmlns:fb="http://www.fixm.aero/base/3.0" xmlns:sdss="
http://www.fixm.aero/ext/sdss/3.0" xmlns:ff="http://www.fixm.aero/foundation/3.0"
xmlns:faa="http://www.fixm.aero/ext/faa/3.0" centre="ZNY" flightFiler="" flightType=
"NON_SCHEDULED" remarks="" source="CL" system="FDPS1" timestamp="2016-08-
29T21:57:46.546Z">
  <fx:agreed>
    <fx:route routeText="GRD LBT 3500N/07400W 3726N/06937W 3745N/06720W 3900N/05000W
4000N/04000W DETOX BARDI UQ424 LUMAS UM985 STP ASTIG ASTI1H" flightDuration=
"PT50M0.000S" initialFlightRules="IFR">
      <fx:expandedRoute>
        <fx:routePoint estimatedTime="2016-08-29T21:01:00.000Z">
          <fx:point xsi:type="fb:FixPointType" fix="LBT"/>
        </fx:routePoint>
        <fx:routePoint estimatedTime="2016-08-29T21:32:00.000Z">
          <fx:point xsi:type="fb:LocationPointType">
            <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
              <ff:pos>34.31281554905527 -118.89953613281249</ff:pos>
            </fb:location>
          </fx:point>
        </fx:routePoint>
        <fx:routePoint estimatedTime="2016-08-29T22:05:00.000Z">
          <fx:point xsi:type="fb:LocationPointType">
            <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
              <ff:pos>35.15135442846945 -119.38980102539062</ff:pos>
            </fb:location>
          </fx:point>
        </fx:routePoint>
      </fx:expandedRoute>
      <fx:initialCruisingSpeed uom="KNOTS">480.0</fx:initialCruisingSpeed>
      <fx:requestedAltitude ref="FLIGHT_LEVEL" uom="FEET">
33000.0</fx:requestedAltitude>
      <fx:segment>
        <fx:routePoint>
          <fx:point xsi:type="fb:FixPointType" fix="GRD"/>
        </fx:routePoint>
      </fx:segment>
      <fx:segment>
        <fx:routePoint>
          <fx:point xsi:type="fb:FixPointType" fix="LBT"/>
        </fx:routePoint>
      </fx:segment>
      <fx:segment>
        <fx:routePoint>
          <fx:point xsi:type="fb:LocationPointType">
          <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
            <ff:pos>34.31281554905527 -118.89953613281249</ff:pos>
          </fb:location>
        </fx:point>
      </fx:routePoint>
    </fx:segment>
    <fx:segment>
      <fx:routePoint>
        <fx:point xsi:type="fb:LocationPointType">
          <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
            <ff:pos>34.6015631772409 -119.1522216796875l</ff:pos>
          </fb:location>
        </fx:point>
      </fx:routePoint>
    </fx:segment>
    <fx:segment>
      <fx:routePoint>
        <fx:point xsi:type="fb:LocationPointType">
          <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
            <ff:pos>35.15135442846945 -119.38980102539062</ff:pos>
          </fb:location>
        </fx:point>
      </fx:routePoint>
    </fx:segment>
    <fx:segment>
      <fx:routePoint>
        <fx:point xsi:type="fb:FixPointType" fix="DETOX"/>
      </fx:routePoint>
    </fx:segment>
  </fx:route>
</fx:agreed>
<fx:aircraftDescription wakeTurbulence="H">
  <fx:aircraftType>
    <fx:icaoModelIdentifier>B744</fx:icaoModelIdentifier>
  </fx:aircraftType>
  <fx:capabilities>
    <fx:communication>
      <fx:communicationCode>V</fx:communicationCode>
    </fx:communication>
    <fx:navigation>
      <fx:navigationCode>D O</fx:navigationCode>
    </fx:navigation>
    <fx:surveillance>
      <fx:surveillanceCode>S</fx:surveillanceCode>
    </fx:surveillance>
  </fx:capabilities>
</fx:aircraftDescription>
<fx:arrival>
  <fx:arrivalAerodrome xsi:type="fb:IcaoAerodromeReferenceType" code="LIMC"/>
</fx:arrival>
<fx:departure standardInstrumentDeparture="DOOLY7">
```

```xml
      <fx:departureAerodrome xsi:type="fb:IcaoAerodromeReferenceType" code="KATL"/>
      <fx:runwayPositionAndTime runwayName="27R">
        <fb:runwayTime>
          <fb:actual time="2016-08-29T20:46:00.000Z"/>
          <fb:target time="2016-08-29T20:53:41.258Z"/>
        </fb:runwayTime>
      </fx:runwayPositionAndTime>
      <fx:standPositionAndTime standName="D7">
        <fb:standTime>
          <fb:actual time="2016-08-29T20:49:36.258Z"/>
          <fb:target time="2016-08-29T20:49:36.258Z"/>
        </fb:standTime>
      </fx:standPositionAndTime>
    </fx:departure>
    <fx:enRoute system="TGF">
      <fx:beaconCodeAssignment>
        <fx:currentBeaconCode>2003</fx:currentBeaconCode>
      </fx:beaconCodeAssignment>
      <fx:position positionTime="2016-08-29T21:14:06.926Z" source="FAA" system="TGF">
        <fx:actualSpeed>
          <fx:surveillance uom="KNOTS">0.0</fx:surveillance>
        </fx:actualSpeed>
        <fx:altitude ref="FLIGHT_LEVEL" uom="FEET">28000.0</fx:altitude>
        <fx:position xsi:type="fb:LocationPointType">
          <fb:location srsName="urn:ogc:def:crs:EPSG::4326">
            <ff:pos>${lat} ${lon}</ff:pos>
          </fb:location>
        </fx:position>
        <fx:track uom="DEGREES">${bearing}</fx:track>
      </fx:position>
    </fx:enRoute>
    <fx:flightIdentification aircraftIdentification="MNG200D"/>
    <fx:gufi codeSpace="urn:uuid">8c7995c5-1a65-430c-96d8-a8347b9ed2a3</fx:gufi>
  </fx:Flight>
```

Appendix D: Revision History

Table 11. Revision History

Date	**Editor**	**Revision**	**Primary clauses modified**	**Descriptions**
February 11, 2016	M. Rieke	0.1	all	initial version
February 29, 2016	A. Balaban, M. Rieke	0.2	all	Outline, Abstract
April 29, 2016	M. Rieke	0.3	all	Refine Outline, Architecture, AMQP 1.0 Profile → Draft Engineering Report
July 12, 2016	M. Rieke	0.4	all	Major content for all sections
August 15, 2016	A. Balaban, M. Rieke	0.5	all	Revision based on feedback
August 25, 2016	A. Balaban, M. Rieke	0.9	all	Revision based on feedback
October 24, 2016	A. Balaban, M. Rieke	1.0	Terms, FAA-specific AMQP 1.0 Profile	section on SWIM-TI and PubSub taxonomy, documentation of FAA NEMS integration

Appendix E: Bibliography

1. Prem, J., Ciconte, B., Devgan, M., Dunbar, S., Go, P.: BEA WebLogic Platform 7. Sams Publishing (2003)
2. OASIS Open: Advanced Message Queuing Protocol (AMQP) Version 1.0. (2012)
3. OASIS Open: Web Services Base Notification 1.3. (2006)
4. Eurocontrol: Network Manager 19.5 B2B Reference Manual. (2015)
5. FAA: SWIM Program Overview, `http://www.faa.gov/nextgen/programs/swim/overview/`
6. FAA: SWIM Technology & Implementation, `http://www.faa.gov/nextgen/programs/swim/overview/technology/`
7. FAA: System Wide Information Management (SWIM) Information Access to Transform the Aviation Community (2016), `https://www.faa.gov/nextgen/programs/swim/documentation/media/briefings/SWIM-Product-Portfolio.pdf`